Anti-Aliasing Issues in Image Composition

Christopher D Shaw
Mark Green
Jonathan Schaeffer

Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2H1
{ cdshaw, mark, jonathan } @alberta.uucp

Abstract

This paper describes a new parallel architecture for performing high-speed raster graphics. This architecture relies on an extension of the Z-buffer algorithm called *Composition*, due to Duff. Duff's algorithm proves to be too complex for our implementation technology, so modifications of Duff's algorithm are introduced and evaluated experimentally. The two major issues addressed are anti-aliasing performance of the new algorithm where the rasters intersect in Z, and the anti-aliasing performance of composition in scenes of high depth complexity.

Résumé

Cet article décrit une nouvelle architecture parallèle pour exécuter des graphics "rasters" a grande vitesse. Cette architecture est en fait une extension de l'algorithme Z-buffer appelé *Composition* proposé par Duff. L'algorithme de Duff s'avère trop complexe pour notre application; On y introduit des modifications qui sont évaluées expérimentalement. Les deux points importants sont les performances d'anti-aliasing du nouvel algorithme ou les "rasters" intersectent en Z, et les performances d'anti-aliasing pour les compositions de scènes complexes de grande profondeur.

Keywords: Image Quality, Parallelism, Computer Hardware Architecture, VLSI Implementation.

1. Introduction

With the steady increase in demand for higher and higher performance graphics, traditional designs utilizing a geometry pipeline and a fast rendering chip are starting to meet severe technological limits. Parallelism offers an escape from this performance limit. The method of graphics parallelization that we propose breaks up the graphics production task by object. The modeling task in a host processor distributes graphical objects (collections of polygons) to a number of independent general-purpose Graphics Processors (*GP*). Each GP performs the geometry and rendering tasks on one graphical object without communicating 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.

The advantage of an object-based system is that the modeling and rendering computations can be all be performed independently on an object-by-object basis, thus allowing the possibility of linear speedup over N processors. 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 *GPs*. We have developed a VLSI architecture which solves this problem by implementing an anti-aliasing variation of Z-buffer called Composition [Duff85, Porter84]. This architecture utilizes parallelism in a way that has not been adequately explored to date. In particular, while the Host-GP setup shown in figure 1 is not new, other methods that have been proposed until recently for combining the resultant rasters have been unsatisfactory.

The object-level approach has been proposed before, namely by Weinberg [Weinberg81], and by Fussel & Rathi [Fussel82]. A system like Weinberg's has been implemented recently [Deering88], and an extension of Weinberg's system has appeared in [Schneider88]. Each of the systems has 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 does not suffer unduly from either of these problems.

Section 2 describes the Composition architecture, while sections 3 and 4 discuss what algorithms might be used to perform the Composition task. In section 5 we discuss various methods of simplifying Duff's composition algorithm for VLSI implementation. Sections 6, 7, and 8 report on experiments performed to evaluate in terms of image quality the various approximations. Section 9 discusses the effect of random depth order of independent rasters, and section 10 offers some opinion on the quality of an image given the statistical results reported in section 8.

2. The Composition Architecture

We have designed a 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, and composes this pair of rasters into one raster of the same format. Since the composition operation produces output in the same format as its input, we can take a pair of composed frames and compose them also. This composition process continues until one final raster is produced. Thus, for N rasters, 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 1. 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. A concise description of the system is available in [Shaw88b], with more details in [Shaw88a]. 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 Compositor passes its results to the frame buffer. The advantage would be easy scalability to any number of processors without much effort. The problem potentially 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 $\log_2 N$ with the tree arrangement. Since each Compositor approximates Duff's algorithm, errors may build up after a number of composition steps. From an error point of view, the fewer steps, the better, which is what the tree offers. We will address the error accretion issue later in this paper.

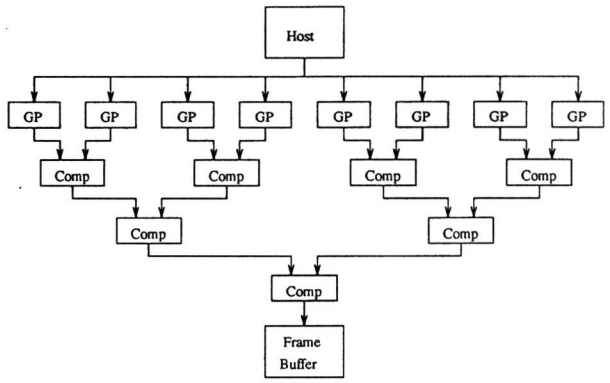

Figure 1: High-Level View of the Composition Architecture

3. 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 [Carpenter84] 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 [Catmull78] and Crow [Crow81], which suggest the use of subpixel information to perform anti-aliasing. Similarly, work by Fiume et al. [Fiume83] advocates enhancing Z-buffer with subpixel resolution information for the purposes of parallel implementation on a shared-memory machine.

However, the key restriction with A-Buffer is that each pixel's contributions must be sorted in order of depth, which is impossible in our architecture, since the Compositor combines arbitrary pairs of pixels. 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 [Duff85] 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..1.0], where a value of zero indicates no coverage and one indicates full 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 centre 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. 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. We will elaborate a little more on this issue later.

The lack of a sort requirement means that composition does not suffer the unboundedness of A-Buffer. In fact, A-Buffer cannot handle unsorted data at all, so Composition is the clear choice for our architecture. There is a trade-off however, since Duff's coverage measure does not include any positional information.

4. Duff's Composition Algorithm

With two rasters named **Ras1** and **Ras2**, two pixels $pixel_{Ras1}$ and $pixel_{Ras2}$ are composed 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 2, some pixels will intersect: that is, the Z comparison in some corners will be of the opposite sign of the Z comparison in other corners. In this case, we must determine the fraction β, which is the coverage ratio between the two pixels.

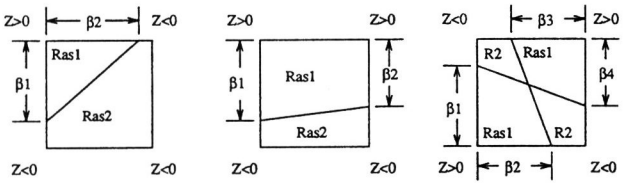

Figure 2: Three Examples of Intersecting Pixel Contributions

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 $pixel_{Ras1}$ and the contribution of $pixel_{Ras2}$. The number β is the fraction of the pixel taken up by $pixel_{Ras1}$. In the left example in figure 2, β would be the proportion of the pixel labeled **Ras1**, 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.

$$R_{comp} = \beta \times (R_{Ras1} + (1-\alpha_{Ras1}) \times R_{Ras2}) + (1-\beta) \times (R_{Ras2} + (1-\alpha_{Ras2}) \times R_{Ras1}) \quad (1)$$

$$\alpha_{comp} = \alpha_{Ras2} + \alpha_{Ras1} - \alpha_{Ras2} \times \alpha_{Ras1} \quad (2)$$

$$Z_{comp} = Min(Z_{Ras1}, Z_{Ras2}) \quad (3)$$

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. None of these operations are difficult to implement in VLSI, and with clever pipeline scheduling, two small pipelined multipliers can perform all the required multiplications at the same rate that data is input to the chip.

However, β must be calculated from two or four interpola-

tions of the Z intersection point along each edge of differing corner sign. An edge interpolation is called edgeβ, and each is the proportion of the edge taken by **Ras1**, calculated as follows:

$$edge\,\beta = \frac{|diff_{Ras1}|}{|diff_{Ras1}| + |diff_{Ras2}|} \quad (4)$$

Here $diff_{Ras1}$ is the difference between **Ras1** and **Ras2** Z's in the corner where **Ras1** is nearest to the viewer. Conversely for $diff_{Ras2}$.

5. Estimating β Without Floating Point

The β calculation is substantial, since at least two edgeβs must be evaluated to form β, meaning that at least two divisions must be performed for each pixel. Gate limitations in the gate array technology used to implement the Compositor preclude a full floating point calculation. Division is a complex operation to implement, 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 remainder of the paper will describe how the experiments were performed, the types of approximations used, and the statistical analysis used to evaluate the results.

There are three aspects to estimating β without using Duff's algorithm. These aspects may be considered as independent axes and there is a Cartesian product of estimation procedures in which one "value" from each axis is chosen to form an estimation *candidate algorithm*. The purpose of a candidate algorithm is to perform the linear interpolation algorithm without using division.

The first axis is the *pattern* of estimation, which classifies where Z data is to be sampled in the pixel. One of these sampling patterns has been discussed in the literature [Catmull78] [Crow81]; we will introduce a different pattern specialized for the β estimation problem.

The second axis is the *quantity* of data collected per sample. Thus, for a given pattern, more data can be collected to give more accurate results.

The third axis is the *model* of estimation. A model abstractly describes how the estimation is to be calculated, but does not concern itself with implementation details. That is, a model tells how a particular set of pixel Z samples are to be combined into an estimate of β, while a pattern tells where the pixel is to be sampled. For a given model, there may be many patterns collecting different amounts of data.

5.1. Patterns of Data Collection

All of the patterns of data collection are based upon using corner Z comparisons, and the comparison of various combinations of corner Z values. We will call these combinations of corner Z values *aggregate comparisons*, and the corner Z comparisons *fundamental samples*. For example, two types of aggregate comparisons considered were edge-midpoint and centre-pixel. The centre-pixel comparison is as follows:

$$(Z(x,y)_{Ras1}+Z(x-1,y)_{Ras1}+Z(x,y-1)_{Ras1}+Z(x-1,y-1)_{Ras1}) - (Z(x,y)_{Ras2}+Z(x-1,y)_{Ras2}+Z(x,y-1)_{Ras2}+Z(x-1,y-1)_{Ras2}) \quad (5)$$

Similarly, the left edge-midpoint comparison is generated by the following:

$$Left\ Edge = Z(x-1,y)_{Ras1}+Z(x-1,y-1)_{Ras1} - Z(x-1,y)_{Ras2}-Z(x-1,y-1)_{Ras2} \quad (6)$$

The point to mention here is that each aggregate is a sample of the *difference* between Z contours of the two source pixels at a particular point on the pixel. Each sample therefore requires the weighted sum of the corner Z values. Also, each aggregate sample is simple to calculate since it is simply the sum of at least two more fundamental samples. The easiest ones to calculate are those that are a sum of two fundamental samples.

In terms of implementation, most patterns will use only the sign of the aggregate comparisons to generate β values. More data per sample can be collected, but the pattern remains the same. The pattern classes that were experimentally evaluated are:

1) The $N \times N$ pattern, in which N^2 samples are collected at equally-spaced grid points within one pixel. For example, if N=2, the four corner samples are used. If N=3, the four corners, the four edge-midpoint aggregates, and the whole pixel aggregate is used.
2) The edge-only pattern, in which samples in the $N \times N$ pattern which lie only on a pixel edge are used. This implies $4N-4$ samples, which are fairly simple to calculate from fundamental samples.

5.2. Quantity of Data Collected Per Sample

Traditional graphics algorithms sample a geometrical model in some sort of grid pattern to find which geometrical primitive is visible at the sample point. In this context, quantity of information is unimportant. In the context of sampling the Z contour, however, gathering more information per sample is useful since we are sampling to find the line of intersection of exactly two planes by interpolation. Thus, instead of simply using the sign of the comparisons at each sample point (i.e. the most significant bit), we can use the most significant N bits of each sample as a basis for approximation.

However, a naive implementation of multi-bit samples will not work unless the **Ras1** and **Ras2** Z values differ in the N most significant bits. In other words, there is a magnitude problem. This can be solved by shifting all the differences left until the largest difference is fully "shifted in". Circuitry for this task is found in the mantissa normalization section of all floating point processors. However, integrating similar circuitry in a Compositor will be impractical, since we need to simultaneously normalize four numbers by the same amount.

5.3. Models of Approximation

There are two basic models of evaluating β given a certain collection of samples. Each model rests on the concept of attaching a weight to a sample or set of samples, then manipulating the weighted samples in some way. In the following sections, 3×3 or 3×3 edge-only patterns with one bit samples will be used to illustrate each model.

5.3.1. Contribution Model

The contribution model simply assigns weights to each sample. If a given comparison results in **Ras1** being closer, then the weight for that sample is added to the total. For a whole pixel, the sum of all weights equals one. The advantage of this model is that it is simple to implement. The challenge is to pick meaningful weighting values, as evaluated by experimentation. Note that the contribution model performs a pixel filtering at subpixel resolution in much the same way as Crow mentions in [Crow81].

5.3.2. Linear Intersection Model

At its simplest, the linear intersection model derives the final β value from a Duff evaluation using various edgeβ's. That is, imagine a line intersection that would yield corner, edge-midpoint, and whole pixel comparisons of the particular given type, then generate the β value by using canonical edgeβ values that suit the given situation. To maintain consistency, the canonical values chosen for the edgeβ's are always the

same for a given (left, edge-midpoint, right) set of comparisons per edge.

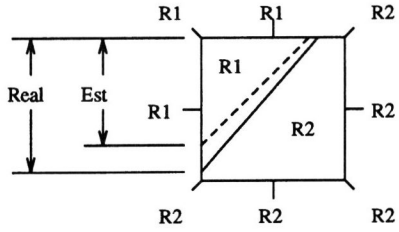

Figure 3: Real vs. Approximation Using Linear Intersection Model

For example, figure 3 shows a pixel which has been sampled in a 3×3 edge-only pattern with one bit samples. In this pixel, **Ras1** is nearer in the top left corner, and **Ras2** is nearer in the rest of the corners. The solid line shows the intersection of the two rasters. The R{1,2} labels at the four edges of the pixel show the results ot the edge-midpoint comparisons. The dashed line shows the estimated intersection. In this case, the canonical *Left edge* β=3/4, and *Top edge* β=3/4. This is used to generate β=9/32.

This is not to imply that we must use *edge* β=3/4 whenever (left, edge-midpoint, right) = (**Ras1, Ras1, Ras2**). Experiments were done to evaluate the performance of composing various pictures using other canonical values for edgeβ.

Clearly, the linear intersection model is best suited to edge-only sample patterns, since it is an analogue of the algorithm that Duff uses to generate β. *N*×*N* sample patterns do not fit this model.

6. Experimentation

As mentioned in section 5, a number of candidate algorithms were developed by picking one "value" from each of the three "axes" of *pattern*, *quantity*, and *model*. Each candidate was then evaluated by running it on a representative sample of raster pairs with intersecting Z values. The composed candidate raster was then compared with a reference raster composed by Duff's algorithm. However, a visual comparison is not sufficient, since colour effects and other differences distort the picture enough that small differences in picture quality are impossible to reliably distinguish.

Therefore, a "raster difference" was performed, in which corresponding R, G and B pixel values are compared in a pair of rasters. If any of the values are different at a given location, then the absolute value of the difference is stored in a histogram data structure. From this, the minimum and maximum difference and standard deviation are reported. Standard deviation is measured by taking the sum of the squares of the differences divided by the number of pixels which differed in the picture. Standard deviation proved to be the most accurate performance metric. That is, the best pictures had the least standard deviations when compared to rasters composed by Duff's algorithm.

7. Experimental Rasters

The experiment performed was fairly simple. For each candidate algorithm, try the algorithm on the following sets of experimental rasters. Two classes of experimental rasters were used:

The first, called the chevron class, was simply a pair of monochromatic rasters that were composed together to observe a candidate algorithm's performance with respect to the slope of the line of intersection. Both images consisted of one square monochromatic polygon which filled a frame buffer measuring 64×64 pixels. The first source raster was a white square with Z = 10000, Red, Green and Blue = 255, and α=1.0.

The second chevron source raster was a blue square with 5000 < Z < 15000, Red = 0, Green = 0, Blue = 120, and α=1.0. The varying Z's created intersections in a chevron pattern, with the upper half of the 64×64 raster having intersections of positive slope, and the bottom half being a mirror image of negative slope. The left half of figure 4 shows an example composed image, where the viewer is looking down the Z axis at the X-Y plane. The right half of figure 4 shows a profile of the image, where Z is the horizontal axis and Y is the vertical axis. The Z contour of the blue square, shown by the solid lines, is "corrugated", while the Z contour of the white square shown by the dashed line is flat. This image exposes all possible types of intersections: positive and negative slope, and **Ras1** above and below the intersection line.

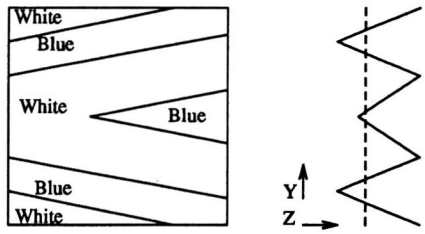

Figure 4: The Chevron Test Image

The slopes of intersection used ranged from 0.0875 (acute chevron) to 11.4 (obtuse), in equal intervals of 5 degrees. Care was taken to avoid "nice" slopes like 1, since even Z-buffer looks good at slope 1. The slopes used are listed in table 1.

Table 1: Chevron Slopes Tested

Degrees	Slope	Degrees	Slope
5.00	0.087489	50.00	1.191754
10.00	0.176327	55.00	1.428148
15.00	0.267949	60.00	1.732051
20.00	0.363970	65.00	2.144507
25.00	0.466308	70.00	2.747477
30.00	0.577350	75.00	3.732051
35.00	0.700208	80.00	5.671282
40.00	0.839100	85.00	11.430052

The second test raster set was a group of 8 monochromatic rasters which were composed in arbitrary order so as to demonstrate the algorithm's ability to handle both the intersection of random depth-ordered cases and the intersection of more than two rasters in one pixel. The final composed result of the 8 mono rasters was a pinwheel.

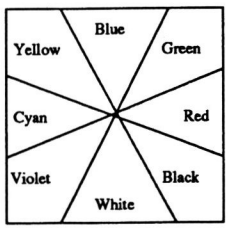

Figure 5: The Pinwheel Test Image

Each of the 8 rasters was a unique colour generated by setting R, G, and B to either 0 or maximum (255). This colour choice guarantees that any two rasters will have at least one colour channel that differs by 255, which is important for statistical purposes. The Z contour in this case was a simple plane

set up in such a way that maximum gradient from the centre of the raster was in a unique direction for each raster. Figure 5 shows an example of the final composed result of these 8 rasters.

Since there is 8-way symmetry, the range of slopes for the upper intersection of the red section of the pinwheel lies from 5 to 40 degrees in steps of 5 degrees. The first column of table 1 shows the slopes used. These slopes represent a range of data that one can expect in real pictures.

8. The Candidate Algorithms

We will group the candidates according to model. The first model to consider is Linear Intersection. The pattern in this case is edge-only, with N equal to 2 or 3 for a total of four or eight samples respectively. Each sample pattern was tried with 1, 2, and 3 bits per sample. This yields 6 major candidates. However the 3×3 edge-only sample pattern with one bit per sample can be subdivided into a number of minor candidates which differ in the values of the canonical values of edgeβs used. A total of six variant edgeβs were used. The purpose of these variants was to test for the best values for the canonical edgeβs.

We did not use canonical edgeβ values for the 2 and 3 bit sample because with more bits per sample, the linear interpolation algorithm can be used directly but with reduced precision. For the 2×2 pattern, the result is similar to Duff's algorithm with N-bit Z values. With the 3×3 pattern, the interpolation can take place on either the left or right half of a pixel edge, since the Z planes cross in one half only.

Table 2 shows the performance of all of the candidate algorithms except for the 3×3 pattern with one bit samples. The numbers listed for each candidate are the minimum, maximum, and arithmetic mean of the standard deviations for the chevron and pinwheel pictures. Table 3 shows the performance of the candidate algorithms with the 3×3 pattern using one bit samples. The two extra columns list the canonical edgeβs used. For the pinwheel figures, about 90 pairs of rasters were composited for each candidate algorithm. For the chevron picture, 16 raster pairs were tested. The algorithms labelled in the Name column will be referred to again later.

Table 2: Picture Performance - Linear Model

		Bits per	Chevron			Pinwheel		
Name	Pattern	Sample	Min	Max	Mean	Min	Max	Mean
	2×2	1	22.22	35.58	26.51	21.87	35.63	25.02
	2×2	2	12.29	20.32	16.04	12.68	21.66	15.38
	2×2	3	6.85	12.53	9.38	5.52	17.30	8.91
A	3×3	2	6.76	15.36	11.15	7.75	15.87	11.04
B	3×3	3	2.48	9.35	6.24	4.99	13.43	6.80

Table 3: Picture Performance - Linear Model Using Canonical edgeβs, 3×3 Pattern, 1 bit per sample.

			Chevron			Pinwheel		
Name	edgeβ1	edgeβ2	Min	Max	Mean	Min	Max	Mean
	0.375	0.625	17.07	40.35	24.83	17.14	38.13	22.17
	0.333	0.667	14.68	34.40	21.53	14.95	32.61	19.77
C	0.25	0.75	13.97	28.12	18.44	13.37	27.50	18.44
	0.20	0.80	15.42	29.90	20.13	15.49	30.50	20.05
	0.15	0.85	19.35	35.33	24.01	19.45	36.93	23.24
	0.125	0.875	21.69	39.93	26.89	22.26	41.32	25.85

Not too surprisingly, the best performance is achieved by the algorithm that takes the most and deepest samples. The best variant algorithm in table 3 has the canonical edgeβs at the midpoint between the samples. This corresponds to the situation where the linear intersection algorithm is given two Z differences of equal magnitude and opposite sign.

Using the best candidate of table 3 to compare with table 2, the means of the standard deviations reduce by a factor of approximately 0.6 for every extra bit added to sample depth in the 2×2 pattern. For the 3×3 pattern, the same effect holds: add a bit to each sample to reduce the mean error to 0.6 times its current value.

Interestingly enough, increasing the number of samples is only marginally less effective: Going from 2×2 to the 3×3 edge-only pattern reduced the mean standard deviation by a factor of 0.695 for one or two bits per sample, and by 0.668 for three bits per sample. In terms of hardware trade-offs, it is clear that unless a narrow bit-width divider can be built for less than 15% more cost than an adder, collecting more samples per bit is not worth the bother. Considering the requirement for normalization circuitry, the divider option is clearly not viable.

The second model to consider is the Contribution model. Two patterns are useful in this context: either edge-only pattern with N equal to 2 or 3 for a total of four or eight samples, or the square pattern with N equal to 3 for nine samples. Each sample pattern was tried with only 1 bit per sample since the contribution model does not do any interpolation between samples.

There are therefore three major candidates: the first is the 2×2 pattern, the second is the 3×3 edge-only pattern with 5 variants, and the third major candidate is the 3×3 square pattern with 5 variants. Each variant uses a different set of weights for the three sample positions. Table 4 shows the performance of all of the candidate algorithms in a similar manner to table 3. The "Corner", "Edge", and "Centre" columns indicate the weights used.

Again, more samples give better results, although it is a mild surprise that adding the centre weight should decrease the best-case mean by a factor of 0.659, especially since the improvement by going from the 2×2 pattern to the edge-only 3×3 pattern is 0.592.

There are two candidates tied for first place with the 3×3 square pattern, based on the chevron results. The slightly worse candidate (named D) implements a 3×3 Bartlett filter [Crow81], while the better candidate increases its centre weight at the expense of the corner weights. The third place candidate expands the centre weight at the expense of the edge weights in comparison to the Bartlett scheme, but to less effect. The fourth-place candidate is a box filter with equal weights everywhere, and the last candidate has the greatest weights on the pixel periphery.

In terms of hardware, the easiest candidate to implement is the Bartlett filter, since all the weights are negative powers of two, and all the samples are 1 bit. Once the samples have been generated, a compact adder circuit can calculate the approximate β. The best one-bit-sample linear intersection candidate (algorithm C in table 3) had a mean only 1 less than the Bartlett filter candidate. Since some nontrivial hardware must calculate the linear intersection from the two edgeβs, clearly the Bartlett filter is best given tight hardware constraints.

9. The Effect of Depth Order

There are three aspects to consider with respect to how approximations of the Composition operator handle randomly-ordered polygon data. The first is the problem of error accumulation mentioned in section 2. The second aspect is how pixels or groups of pixels in areas of high depth complexity might be wrongly calculated. The third issue is how the algorithm handles anti-aliased edges on a transparent black background in

Table 4: Picture Performance - Contribution Model

					Chevron			Pinwheel		
Name	Pattern	Corner	Edge	Centre	Min	Max	Mean	Min	Max	Mean
	2×2	.25	-	-	40.49	67.19	49.61	40.38	64.61	44.13
	3×3	.0625	.1875	-	26.14	35.62	29.38	27.00	34.56	29.51
	3×3	.0833	.1667	-	27.61	38.08	30.14	26.25	35.93	28.86
	3×3	.111	.1388	-	27.36	43.36	32.03	27.42	39.52	29.75
	3×3	.125	.125	-	27.34	43.85	32.84	27.38	41.28	30.09
	3×3	.1667	.0833	-	29.83	51.19	37.57	30.0	48.55	33.91
	3×3	.05	.125	.3	14.13	32.21	19.36	13.84	32.04	16.66
D	3×3	.0625	.125	.25	14.70	31.34	19.40	14.65	30.57	17.16
	3×3	.0625	.09375	.375	16.40	32.68	21.29	15.93	33.28	18.82
	3×3	.111	.111	.111	21.47	37.57	26.33	21.24	35.32	24.13
	3×3	.125	.1035	.0858	23.18	40.61	28.81	22.92	38.71	25.58

the source raster in areas of high depth complexity.

The first issue is easy to solve, since error buildup can occur in only two situations: either there is a pixel intersection in Z, causing an approximate β to be used, or there is an anti-aliased edge of partial coverage being composed over a non-transparent pixel. Clearly the β approximation is our only concern, since this is the only source of error; our algorithm does accurate calculations with α, so therefore erroneous new pixel values will not arise due to the numerical inaccuracy of α-based calculations.

So, the only source of error is the β approximation. This can hurt us if the final resultant pixel has more than one surface visible in it. In other words, three or more surfaces must intersect at the same pixel(s). If only two surfaces are visible in the final raster, then they could only have undergone one pixel blending step as the component rasters passed from Compositor to Compositor. Therefore error will build up in pixels where three or more planes intersect. However, these cases will be rare, and the only way that it will be noticed is if the three or more surfaces intersect over a large body of pixels, say in a straight line.

The pinwheel test case helps illustrate the situation with full-coverage high depth complexity pixels. An experiment was performed to compose the 8 source rasters in 8 arbitrary orders. The algorithms used were Duff's linear intersection, plus the approximate algorithms A through D from the above tables.

The result was visually indistinguishable no matter which order the 8 source rasters were composed in. Statistics were collected to compare all of the composed rasters which used the same algorithm. The reason for this being that the same algorithm should yield similar results and therefore small raster differences. Significantly, the mean standard deviation was similar for the 5 algorithms tested, as shown in table 5.

Table 5: Performance In Areas Of High Depth Complexity

Pattern	Bits	Model	Algorithm	Mean StdDev
Duff			Duff	9.75
3×3	2	Linear	A	13.89
3×3	3	Linear	B	13.85
3×3	1	Linear	C	16.27
3×3	1	Contribution	D	13.12

The final issue is how composition performs when a number of rasters are combined in arbitrary order that have anti-aliased edges with $\alpha<1$. Consider three 3×3 pixel rasters A, B, C in order from nearest to farthest. A has an anti-aliased edge through the middle column, and B and C are mono-

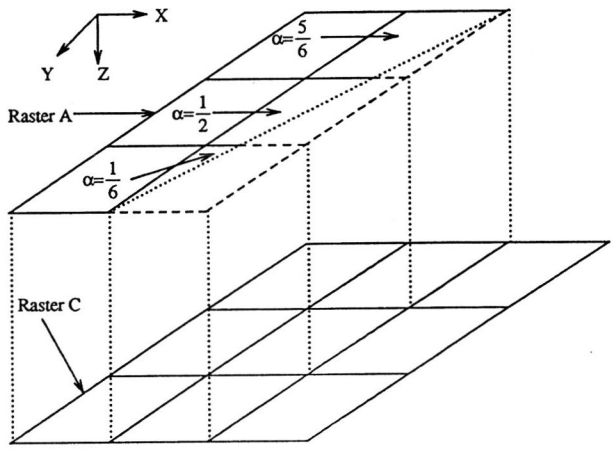

Figure 6: Two rasters A and C. A has two columns by three rows and is at least depth. The right column is an anti-aliased edge. The dashed lines about A show Z=∞. The dotted diagonal line is the edge being sampled. Raster C is at greatest depth, with Z = constant throughout.

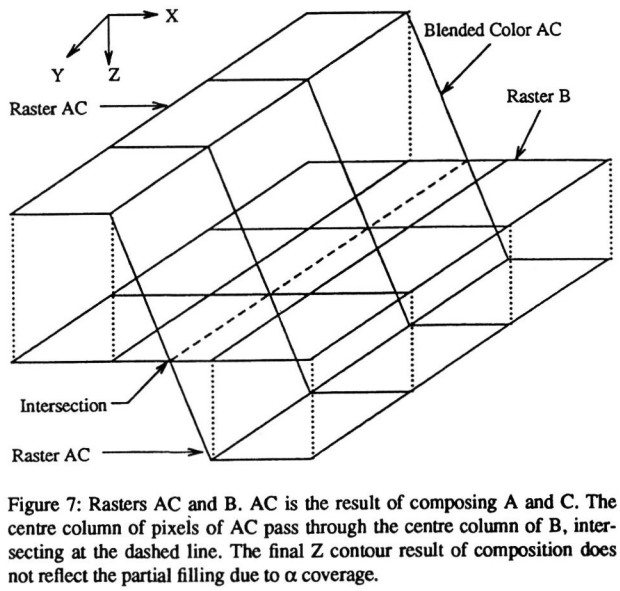

Figure 7: Rasters AC and B. AC is the result of composing A and C. The centre column of pixels of AC pass through the centre column of B, intersecting at the dashed line. The final Z contour result of composition does not reflect the partial filling due to α coverage.

chromatic at constant depth. Figure 6 shows rasters A and C before composition. There are two columns of shaded pixels in A, with the right edge of the right column shown dashed to

indicate that it is at infinite depth. The α's indicate coverage for the three partially-shaded pixels of A. The vertical dotted lines are parallel to the Z axis on the periphery of raster A.

Figure 7 shows raster AC, formed by composing rasters A and C. AC has a cliff-like Z contour with Z values equal to those for A on the left and Z values equal to those for C on the right of the cliff. The cliff face is a blended colour determined by using A's α values. After the α-blending process, the new α=1 for all raster AC. However, the cliff is parallel to the Y axis, which means that when the anti-aliased edge of A continues into the next column, the cliff suddenly jumps into that column.

In figure 7, when B is composed with AC, the resulting raster will form a straight intersection along the cliff face of AC, shown as a dashed line across the cliff face. This arises from picking up constant Z differences between A and B on the left and C and B on the right, yielding constant βs for all three cliff pixels. In situations where the anti-aliased edge of A continues into the next column, the intersection line between AC and B suddenly changes, giving rise to a jagged artifact at the pixel of the column change.

The solution is to only compose objects that are in adjacent depth order, which requires that the modeling task in the host processor be prepared to swap object descriptions around from GP to GP. However, our experience indicates that the aliasing artifacts are not terrible, so in most applications this can be ignored. That is, the aliasing is noticeable, but not nearly as bad as what Z-buffer would produce. Secondly, as more and more rasters are composed between A and C such that each new raster is visible at the edge of A, the aliasing does not get too much worse.

A second problem also exists with pre-anti-aliased edges, and that is co-ordinated edges. If two anti-aliased edges meet, problems occur at the meeting point. Normally this is not a problem, except if the two edges are part of the same object that is being rendered by two separate GP's, then the aliasing will be apparent. A similar high-level solution is used: ensure that the modeling task informs the GP's of any coordinated edges, and the GP's can output coordinated edges at full alpha coverage and colour. This way, any intervening rasters will not disturb the coordinated edge, and the other coordinated edge will blend properly since it is at the same depth, colour and coverage.

10. Picture Quality

We stated earlier that picture quality judgment was too hit-and-miss to be left to one's visual capabilities. While this is still true, it is worth noting that there is a rough correspondence between the standard deviation and jaggedness when an approximation raster is compared against a Duff-produced raster. Rules of thumb, which seem to work well in this context, are that standard deviations less than 15 are "perfect" visually, deviations of 15-30 yields pictures that are "good" but have flaws on close examination, deviations of 30-45 indicate pictures with readily detectable flaws, while 45 or more indicates a more or less aliased picture. Z-buffer yields a standard deviation of 128.

We are in the process of preparing a video tape which illustrates the effects of out-of-depth-order compositing in a dynamic environment, multiple planes visible in a pixel, and the pinwheel picture spinning.

11. Conclusion

We have addressed the major anti-aliasing issues that confront our Compositor architecture. They are the β approximation, random depth order intersection, and the issue of anti-aliased edges in the source rasters.

The β approximation we will use is a 3×3 Bartlett scheme to sample the Z contour of two intersecting raster planes. This algorithm has the advantages of fairly high picture quality, and an inexpensive implementation. Higher-quality solutions would take up too much VLSI resources for the small improvement in picture quality.

Except for the rare case when multiple planes intersect in a line of pixels, our approximation is believed to be immune from error buildup. In our opinion, the probability of three or more planes intersecting in more than one pixel is extremely low, and therefore not worth worrying about unduly. In the more common case of three or more planes intersecting at a point, only the one pixel at that point can be wrong, probably in an area of high scene complexity.

Future directions for research lie in two areas. The first area is the exploration of object-level parallelism for graphics modeling and production. The other area is for possible generalizations of the Compositor architecture.

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

References

Carpenter84.
L. Carpenter, The A-Buffer, an Antialiased Hidden Surface Method, *Computer Graphics (Proceedings of SIGGRAPH '84) Vol18, No3*, (July 1984), pp. 103-108, ACM SIGGRAPH.

Catmull78.
E. C. Catmull, A Hidden-Surface Algorithm with Anti-Aliasing, *Computer Graphics (Proceedings of SIGGRAPH '78) Vol12, No3*, (August 1978), pp. 6-11, ACM SIGGRAPH.

Crow81.
F. C. Crow, A Comparison of Antialiasing Techniques, *IEEE Computer Graphics & Applications Vol1, No1*, (January 1981), pp. 40-48, IEEE Computer Society.

Deering88.
M. Deering, S. Winner, B. Schediwy, C. Duffy and N. Hunt, The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics, *Computer Graphics (Proceedings of SIGGRAPH '88) Vol22, No4*, (August 1988), pp. 21-30, ACM SIGGRAPH.

Duff85.
T. Duff, Compositing 3-D Rendered Images, *Computer Graphics (Proceedings of SIGGRAPH '85) Vol19, No3*, (July 1985), pp. 41-44, ACM SIGGRAPH.

Fiume83.
E. Fiume, A. Fournier and L. Rudolph, A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General-Purpose Ultracomputer, *Computer Graphics (Proceedings of SIGGRAPH '83) Vol17, No3*, (July 1983), pp. 141-150, ACM SIGGRAPH.

Fussel82.
D. Fussel and B. D. Rathi, A VLSI-Oriented Architecture for Real-Time Raster Display of Shaded Polygons, *Graphics Interface '82*, 1982, pp. 373-380.

Porter84.
T. Porter and T. Duff, Compositing Digital Images, *Computer Graphics (Proceedings of SIGGRAPH '84) Vol18, No3*, (July 1984), pp. 253-259, ACM SIGGRAPH.

Schneider88.
B. Schneider and U. Claussen, PROOF: An Architecture for Rendering in Object Space, *Third Eurographics Workshop on Graphics Hardware*, Sophia-Antipolis, France, September 1988, pp. 31-40.

Shaw88a.
C. D. Shaw, The Image Composition Architecture: A Highly Parallel Graphics System, *University of Alberta Master's Thesis*, Edmonton, Alberta, August 1988.

Shaw88b.
C. D. Shaw, M. Green and J. Schaeffer, A VLSI Architecture for Image Composition, *Third Eurographics Workshop on Graphics Hardware*, Sophia-Antipolis, France, September 1988, (Available as U of A Tech. Report 89-1).

Weinberg81.
R. Weinberg, Parallel Processing Image Synthesis and Anti-Aliasing, *Computer Graphics (Proceedings of SIGGRAPH '81) Vol15, No3*, (August 1981), pp. 55-61, ACM SIGGRAPH.