Title: A Comparison of Pixel Complexity in Composition Techniques for Sort-Last Rendering

Author: Y.C. Tay

Affiliation: National University of Singapore

Email: `tay@acm.org`

Abstract: Computer graphics has some of the most compute-intensive applications. Sort-last rendering is a method of parallelizing such applications; specifically, the primitives (e.g. triangles) that describe a scene are first allocated to a set of renderers, and the rendered images are then composited (i.e. combined) to give the final image. There are six such techniques, based on centralized, pipeline, tree, hypercube, mesh and bus architectures. Naturally, one would like to compare these techniques, but standard performance measures for graphics (throughput, latency and frametime) are too dependent on hardware and software to offer a meaningful comparison.

This paper presents a comparison based on *active pixels*, i.e. pixels that are covered by at least one primitive. Active pixels offer a uniform way of accounting for the time, space and bandwidth costs in sort-last rendering, giving complexity measures that are independent of hardware and software. Since graphics architectures usually operate on full frames (including inactive pixels), these measures can be viewed as lower bounds.

The comparison highlights the strengths of each technique. For example, the tree has minimum work, binary-swap (hypercube) has minimum composition latency, direct pixel forwarding (mesh) has minimum bandwidth latency, and snooping (bus) has minimum bandwidth volume; binary-swap's bandwidth and composition latencies decrease, whereas the bandwidth volumes for direct pixel forwarding and snooping are constant, when the number of renderers increase; etc.

The analysis uses an object-based (instead of an image-based) model that matches the rendering techniques, and provides closed-form approximations for the complexity measures. The accuracy for three of these approximations is evaluated by comparison to previous measurements.

A Comparison of Pixel Complexity in Composition Techniques for Sort-Last Rendering

1 Introduction

Computer graphics has an insatiable demand for computing power. (E.g. Pixar uses hundreds of multi-processor machines to produce animated movies.) To satisfy this demand, much work was done on parallel rendering architectures. The first attempts focused on *image-parallel* techniques, in which each renderer in a collection is assigned a region of screen space; the final image is obtained by collecting the regions (subimages) from the renderers and tiling them together.

More recently, there was a burst of activity on *object-parallel* architectures [8, 11, 13, 25, 33, 41, 42, 44], in which the objects that constitute a scene are partitioned and each partition assigned to a renderer. The renderers' images are then composited to form the final image; Molnar et al. described this as *sort-last* [32]. There is already a handful of sort-last architectures in research laboratories and on the market.

The object-parallel architectures are based on five different schemes — pipeline [33, 38, 42, 46], tree [17, 27, 30, 36, 37, 43, 44, 45], hypercube (binary-swap [20, 28, 29, 40]), mesh (direct pixel forwarding [24]) and bus (snooping [9]) — so a natural question arises: What are their relative merits? The usual performance measures in computer graphics are throughput (the number of primitives rendered per second), latency (the time taken to compute a single frame) and frametime (the time lapse between successive frames) [31]. Such measures are very hardware- and software-dependent. For example, they depend on the hardware support for the geometric calculations, the connection topology for the modules, what the graphics primitives are, and whether shading is done before or after composition.

Hence, it may not be meaningful to compare the techniques with such performance measures, and an abstract comparison based on complexity measures may be more appropriate. However, standard complexity measures like number of rounds and messages [23] are also unsatisfactory, since rounds vary in time and messages vary in size among the algorithms. The complexity comparison should therefore use a more refined and fundamental unit of measure. For parallel graphics algorithms, the natural candidate for a measurement unit is the pixel: We can use pixels to measure and compare the time, space and bandwidth requirements of these algorithms. Such a uniform treatment of complexity measures may be unique to graphics.

Cox and Hanrahan have previously used pixels as a measure for object-parallel rendering. However, their analysis is restricted to buffer usage and snooping. Moreover, they assume that *pixel depth* — i.e. the number of objects that render to a pixel — is independently distributed over the image. This image-based model is a mismatch for the object-parallel algorithms, and it contradicts *area coherence*, i.e. pixel depths are not independent, since each object typically renders to multiple pixels. The mismatch also requires further assumptions that trivialize their analysis (e.g. the z-values to be broadcast in the snooping protocol are assumed independent of the ordering among renderers). Furthermore, they use a generating function as the tool for their analysis. The function captures the distribution of pixel depth, but does not yield closed-form formulas for the performance measures, so the technique cannot be used to compare different algorithms.

Our contributions in this paper are as follows:

(1) We introduce various complexity measures that are defined in terms of *active pixels* — for a particular image, the active pixels are the ones covered by at least one object. Four such measures are: *total active bandwidth volume* B, which measures the bandwidth requirement; *active bandwidth latency* L_B, which measures the time required for sending the pixels in parallel; *active composition latency* L_C, which measures the time required for compositing the pixels in parallel; and *active work* W, which measures the total computational effort. In practice, graphics hardware usually process a full frame at a time, i.e. including the inactive pixels. By using active pixels, we are thus measuring lower bounds on the time, space and bandwidth requirements. This is consistent with (say) *communication complexity*, which measures the number of bits that must be communicated for a set of processors to jointly compute a function [35]. Besides, renderers in sort-last architectures may have a large percentage of inactive pixels [20], so costs may be reduced by storing, computing and sending only active pixels — Molnar et al. called this *SL-sparse*, and such techniques have been implemented in research prototypes [14, 19] and commercial products [15, 16],

(2) We analyze and compare the complexity of the five existing parallel composition techniques. The results are summarized in Table 1, together with those for centralized composition. They show the relative strengths (and weaknesses) of these techniques, thus answering the question posed above (in the third paragraph). In particular, the tree has minimum work W, binary-swap (hypercube) has minimum active composition latency L_C, direct pixel forwarding (mesh) has minimum active bandwidth latency L_B, and snooping (bus) has minimum active bandwidth volume B.

(3) The complexity analysis assumes objects are randomly distributed to the renderers. We thus present an object-based alternative to Cox and Hanrahan's image-based model for buffer usage and snooping analysis; our analytic model is a better match, and makes their questionable assumptions (e.g. the pixel depth at any renderer is at most 1) unnecessary.

We begin in Section 2 (Parallel Rendering) by introducing the object model, analyzing pixel depth at the renderers, and re-examining some issues raised by Cox and Hanrahan [7, 8, 9] about frame buffer design in sort-last architectures. The analytic technique yields closed-form approximations for measuring buffer usage, and we provide a comparison to Cox's data so the reader can judge the accuracy of these approximations.

Section 3 (Parallel Composition) analyzes and compares the six composition techniques. We assume the composition operator is commutative and associative, so the results apply to z-buffering (i.e. choosing the pixel with the smallest z-value from among those with the same (x, y)-coordinates), but not to transparent objects (e.g. in volume rendering). Given the space constraint, we can only present some derivations for binary-swap and snooping. The comparison brings some new insights into these algorithms: composition reduces by half the bandwidth requirement B in binary-swap but has no effect in the case of direct pixel forwarding, binary-swap does the same amount of work W as the balanced tree but the former reduces the active composition volume exponentially (as the algorithm proceeds), and direct pixel forwarding — like snooping — has total active bandwidth volume B that does not vary with the number of renderers. We also point out where our complexity results are supported by empirical observations.

2 Parallel Rendering

In this section, we introduce the object model and analyze the pixel depth at renderers. The analysis begins with the images already generated at the renderers. We thus factor out the differences in complexity among various scene descriptions, lighting models and rasterization techniques.

We assume the architecture consists of renderers and compositors — each with a frame buffer — connected together. Objects are initially allocated to the renderers, each of whom can render a full image (if necessary). The role of the compositors in Section 3 is to help composite the renderers' images into the final image.

2.1 Objects, Buffers and Active Pixels

A pixel may contain color, depth, screen coordinates, object identifier, coverage, opacity, surface normals, texture coordinates, etc. [5, 12, 33, 46]. A pixel may also be a subpixel, as in supersampling [6, 33]. Suppose each frame buffer (possibly virtual [1, 8]) has S pixels indexed by a set I, so $|I| = S$. (Table 2 contains a glossary for the variables we use.)

A *scene* is a collection of *objects*; e.g. polygons, triangle strips or CSG primitives [25, 34, 39]. We say an object renders to a pixel if the object — when rendered *alone* — contributes to the color of the pixel. Thus, for our purpose, an object $\mathcal{O}$ is a mapping $\mathcal{O} : I \to \{0, 1\}$, where $\mathcal{O}(i) = 1$ if and only if $\mathcal{O}$ renders to pixel i. Consider a fixed set of objects $\mathcal{O}_1, \ldots, \mathcal{O}_J$. The *original pixel depth* at pixel i is $d_i = \sum_{j=1}^{J} \mathcal{O}_j(i)$, i.e. the number of objects that render to i if all J objects are rendered with a *single* buffer.

For a given scene, the original pixel depth depends on the viewing transformation. Although d_i is not a random variable, we adapt the idea of mean and variance for d_i, as follows: Let $f_m = |\{i : d_i = m\}|/S$. Since $d_i \le J$ for all i, we have $f_m = 0$ for $m > J$, so $\sum_{m=0}^{J} f_m = 1$. Define *average original pixel depth* (also called *depth complexity* [8, 18, 31]) $\mu = \sum_{m=0}^{J} m f_m$ and *variation in original pixel depth* $\sigma^2 = \sum_{m=0}^{J} (m - \mu)^2 f_m$. Table 3 shows μ and σ for some scenes, which were chosen for their variety in the number and size of objects, screen coverage, average original pixel depth [8] as well as realism [31]. The maximum μ in these scenes is about 25 (for *Galaxy*), but scenes with μ exceeding a hundred are plausible [18].

Consider a collection of buffers indexed by a set F, and let $X_j = b$ if and only if $\mathcal{O}_j$ is assigned to buffer $b \in F$. In our analysis, F always refers to a set of renderers or compositors for which, if the objects are independently and uniformly distributed to the renderers, then $X_1, \ldots, X_J$ are independent and identically distributed random variables. Let $\Pr(X_j = b) = p_b$. Define the *pixel depth* at i in buffer b, denoted $D_b(i)$, as the number of objects, among those assigned to b, that render to i, i.e. $D_b(i) = \sum_{j=1}^{J} \mathcal{O}_j(i)\delta(X_j, b)$ where $\delta(a, b) = 1$ if $a = b$ and 0 otherwise. Since $D_b(i)$ is defined in terms of X_j, it is also a random variable. In fact, it is binomially distributed:

Claim 1
$$\Pr(D_b(i) = k) = \binom{d_i}{k} p_b^k (1 - p_b)^{d_i - k} .$$

Proof
Suppose $\mathcal{O}_{r_1}, \ldots, \mathcal{O}_{r_{d_i}}$ are the d_i objects that are active at i, so $\mathcal{O}_{r_1}(i) = \cdots = \mathcal{O}_{r_{d_i}}(i) = 1$, and $\mathcal{O}_r(i) = 0$ for $r \neq r_1, \ldots, r_{d_i}$. Then $D_b(i) = \delta(X_{r_1}, b) + \cdots + \delta(X_{r_{d_i}}, b)$, where $\delta(X_{r_1}, b), \ldots, \delta(X_{r_{d_i}}, b)$ are independent Bernoulli random variables with $\Pr(\delta(X_r, b) = 1) = \Pr(X_r = b) = p_b$. Therefore $D_b(i)$ is binomially distributed, and the claim follows. □

Thus, although the randomization is by distributing each object as a whole, the probability is as if each original pixel depth d_i is distributed among the buffers; the spatial coherence in the objects, however, imposes a correlation between $D_b(i)$ and $D_b(i')$ for any i and i'. Fortunately, this correlation does not affect our comparison in Section 3 (which is focused on the first moments).

We say pixel i is *active* in buffer b if and only if $D_b(i) > 0$. However, during the composition process, an active pixel in a buffer becomes inactive in that buffer once the pixel is sent to another buffer. By Claim 1, the probability that pixel i is active in buffer b is $\Pr(D_b(i) > 0) = 1 - (1 - p_b)^{d_i}$.

2.2 Pixel Depth at Renderers

In the rendering phase, we assume objects are allocated uniformly and independently among N renderers. This assumption plays two alternative roles: (1) It is a load-balancing scheme, i.e. the objects are in fact randomly allocated to the renderers. (2) The assumption is a model for studying deterministic allocation strategies. Cox had shown that applying random allocation and round-robin allocation to his scenes gave virtually the same results for the average, variance and range of the performance measures [7].

To examine pixel depth at renderers, index the set of renderers by $F = \{1, \ldots, N\}$. The density function for pixel depth k at any location i in buffer b is then given by Claim 1, with $p_b = 1/N$. We now use the following to truncate various expansions: Given two functions g and h on real numbers, we say $h(x) = \Theta(g(x))$ if and only if $C_1 \leq \lim_{x \to 0} \left| \frac{h(x)}{g(x)} \right| \leq C_2$ for some positive constants C_1 and C_2. Thus, the expected fraction of active pixels in a renderer's buffer is

$$\alpha = \frac{1}{S} \sum_{i \in I} (1 - (1 - \frac{1}{N})^{d_i}) = \frac{\mu}{N} - \frac{\sigma^2 + \mu^2 - \mu}{2N^2} + \Theta\Big(\frac{1}{N^3}\Big); \tag{1}$$

this measures how much of a frame buffer is actually used. Hence, a small α means renderers can use fractional buffers. Note that the contribution of variation σ^2 is negative; in this sense, the more uneven the original pixel depth distribution d_i, the smaller α becomes (cf. Cox and Hanrahan's result that uneven depth is worst for z-buffering by snooping [9]). The expected fraction of pixels with pixel depth exceeding 1 is

$$\beta = \frac{1}{S} \sum_{i \in I} \Pr(D_b(i) > 1) = \frac{\sigma^2 + \mu^2 - \mu}{2N^2} + \Theta(\frac{1}{N^3}) \; ; \tag{2}$$

this measures usage of the z-buffer. Again, renderers do not need full-size z-buffers if β is small. The leading term for β is quadratic in $1/N$, so it drops rapidly for small σ and μ. Cox did observe that, at $N = 8$, $\alpha < 0.3$ and $\beta < 0.15$ for most of his scenes. The expected fraction of pixels that need not be sent for compositing if renderers do z-buffering is

$$\gamma = \frac{1}{S}\sum_{i\in I}\sum_{k\geq 2}(k-1)\Pr(D_b(i)=k) = \frac{\sigma^2+\mu^2-\mu}{2N^2} + \Theta\Big(\frac{1}{N^3}\Big). \tag{3}$$

This measures the savings in bandwidth from z-buffering at the renderers. Cox and Hanrahan noted from their scenes that there was a "discrepancy" between β being small (say, less than 0.05) and γ being large (say, more than 0.20); i.e. although there was little z-buffering at the renderers, a significant amount of traffic was saved by these z-buffers [7]. To a first approximation, (2) and (3) show that γ is in fact the same as β. This agrees with one's intuition that, if most of z-buffering involves just two pixels rendered to the same location, then the fraction of z-buffering (β) is also the fraction of traffic saved (γ). The "discrepancy" observed by Cox is the contribution from the higher order terms in $\Theta(\frac{1}{N^3})$ for large pixel depths.

If we drop the $\Theta(.)$ terms, we get closed-form approximations for α, β and γ. To check their accuracy, these approximations (bounded with 0 and 1) are plotted in Figures 1 and 2, together with Cox's measurements. The comparison shows that the approximations are good only when N is large enough that the curves are decreasing and below 0.3. As illustrated for α in Figure 1, the accuracy improves if more terms from $\Theta(.)$ are included in the approximations.

3 Parallel Composition

In this section, we compare the six composition techniques and present some derivations for binary-swap (hypercube) and snooping (bus). We point out where these results are supported by empirical observations, and also discuss some implications of the results.

The composition of two images is done pixelwise, and we make two assumptions: (1) If two pixels are composited to give a third pixel, then all three pixels have the same size; this rules out pixels that have unbounded size, as in A-buffers [41, 46] and object buffers [2, 42]. (2) The operator is commutative and associative, so that the order in which pixels are composited does not affect the result; examples of such operators include voxel z-buffering and a modified version of Duff's composition operator [3, 12, 21, 44], but the most important example is z-buffering.

3.1 Rounds and Metrics

Each composition technique can be considered as proceeding in *rounds*; in each round, pixels are sent and received by some buffers, and the received pixels are composited (either with pixels already in a buffer, or with each other).

We use three metrics: *active area* A_r is the expected number of active pixels in all buffers just *after* round r, *active bandwidth volume* B_r is the expected number of active pixels sent (and received) during round r by all compositors, and *active composition volume* C_r is the expected number of active pixels processed at any buffer during round r. B_r is summed over all rounds to give *total active bandwidth volume* B, and the active bandwidth from one compositor per round is summed over all rounds to give *active bandwidth latency* L_B; these correspond to Neumann's redistribution size and redistribution time [34]. *Active composition latency* L_C is similarly determined by C_r. C_r is summed over all rounds and compositors to give *active work* W, which is the expected total number of active pixels processed over all rounds.

3.2 Centralized, Pipeline and Tree Composition

A straightforward way of compositing the images is to number the renderers and have renderer r send in round r its pixels to a central compositor. The compositor stores the pixels in the first round, and updates its buffer in each round by compositing its pixels with those received in that round. Variations on this technique were used in two commercial architectures [15, 22].

Another way is via a pipeline of compositors. We number the renderers and compositors as follows: renderers 1 and 2 send their images in round 1 to compositor 1 and, for $r = 2, \ldots, N-1$, compositor $r-1$ and renderer $r+1$ send their images in round r to compositor r, so the final image is in compositor $N-1$ (see Figure 3).

The natural generalization of a pipeline is a tree. Consider a complete binary tree with $N = 2^n$ renderers as leaves at level 0 and a root at level n. The composition proceeds from leaves to root as follows: In round r ($r = 1, 2, \ldots, n$), each compositor at level r receives images from its two children (see Figure 4).

The first-order approximations for the various metrics are listed in Table 1. Their derivation is straightforward and omitted here. The constant ρ arises from the fact that, for positive integer m, $\sum_{k=1}^{N} \left(\frac{k}{N}\right)^m = \frac{N}{m+1} + \Theta(1)$. The bandwidth requirement B for centralized composition is about the lowest, but the bottleneck causes the technique to scale poorly in latency L_C and work W, thus motivating the parallel composition techniques. For the tree, the bandwidth and composition latencies are given by $L_B = L_C = 2\sum_{i\in I}\sum_{r=1}^{n}(1-(1-\frac{1}{2^r})^{d_i})$, which can be bounded by $2\xi(\log N)S$, where $\xi = 1 - \sum_{m=0}^{J}\left(\frac{1}{2}\right)^m f_m$.

While it is not surprising that the tree's latencies L_B and L_C (measured along one branch of the tree) are at worst linear in $\log N$, it is interesting that the total active bandwidth volume B and active work W are also linear in $\log N$ (to a first approximation), although they are measured over the entire tree. Intuitively, both B and W are evenly spread among the $\log N$ levels of the tree, so each round has the same active bandwidth requirement. Molnar has also observed that, for scanline A-buffer systems, the bandwidth requirement is balanced [31]. In contrast, pipelining has all four metrics B, L_B, L_C and W linear in N.

3.3 Binary-swap: Composition with a Hypercube

One disadvantage of the tree is that, as the rounds proceed, fewer and fewer compositors are involved. The active work can be more evenly spread out by a technique Ma et al. called *binary-swap*. (It is also called *recursive halving* in the context of the *global combine* operation [4].) For this technique, there are $N = 2^n$ renderers (which also act as compositors), and each renderer's buffer is partitioned into 2^n disjoint regions. For example, if $n = 3$, each buffer may be divided by scanlines into 8 regions $R_{000}, R_{001}, \cdots, R_{111}$, as illustrated in Figure 5.

We index the buffers by binary strings of length n, so $F = \{b_1b_2\cdots b_n : b_i \in \{0,1\}\}$ and there are $N = |F| = 2^n$ buffers. Each buffer is partitioned into 2^n disjoint regions R_b, $b \in F$, so $I = \bigcup_{b\in F} R_b$ and $R_b \cap R_{b'} = \emptyset$ for $b \neq b'$. These regions may be of different sizes (i.e. $|R_b| \neq |R_{b'}|$ for $b \neq b'$). We adopt the usual meaning of the character '$*$' in pattern matching. Thus, for $n = 3$, $R_{01*} = R_{010} \cup R_{011}$ and $R_{101*} = R_{101}$. Also, $\overline{0} = 1$ and $\overline{1} = 0$. The algorithm has n rounds — in round r, buffer $b_1b_2\cdots b_{n+1-r}\cdots b_n$ sends pixels in $R_{b_nb_{n-1}\cdots\,\overline{b_{n+1-r}}*}$ to buffer $b_1b_2\cdots\,\overline{b_{n+1-r}}\cdots b_n$; and receives pixels in $R_{b_nb_{n-1}\cdots\,b_{n+1-r}*}$ from buffer $b_1b_2\cdots\,\overline{b_{n+1-r}}\cdots b_n$. One can view the 2^n buffers as located at the vertices of a virtual hypercube of dimension n; in round r, every renderer exchanges one half of its image with its neighbor along the r-th axis, and composites the half it has left with the half it receives. After the n-th round, each renderer holds a fraction $(1/2^n)$ of the final image, to be tiled together. This is illustrated in Figure 6.

Just after round r, the active pixels in buffer $b_1b_2\cdots b_n$ are in regions $R_{b_nb_{n-1}\cdots\,b_{n+1-r}*}$, and pixels have been collected (directly or indirectly via another buffer) from renderers $b_1b_2\cdots b_{n-r}*$. By Claim 1, the expected number of active pixels is

$$\sum_{i\in R_{b_nb_{n-1}\cdots\,b_{n+1-r}*}} \Pr(D_{b_1b_2\cdots\,b_{n-r}*}(i) > 0) = \sum_{i\in R_{b_nb_{n-1}\cdots\,b_{n+1-r}*}} \left(1 - \left(1 - \frac{1}{2^{n-r}}\right)^{d_i}\right), \tag{4}$$

so the active area just after round r is

$$A_r = \sum_{b_1,\cdots,b_n} \sum_{i\in R_{b_nb_{n-1}\cdots\,b_{n+1-r}*}} \left(1-\left(1-\frac{1}{2^{n-r}}\right)^{d_i}\right) = 2^{n-r}\sum_{i\in I}\left(1-\left(1-\frac{1}{2^{n-r}}\right)^{d_i}\right) \quad \text{for } r = 1,\cdots,n-1. \tag{5}$$

This is the same as for a balanced tree. By (4), the expected number of active pixels sent by buffer $b_1b_2\cdots b_n$ is $\sum_{i\in R_{b_nb_{n-1}\cdots\,\overline{b_{n+1-r}}*}}\left(1 - \left(1 - \frac{1}{2^{n+1-r}}\right)^{d_i}\right)$, so the active bandwidth volume is

$$B_r = \sum_{b_1,\cdots,b_n} \sum_{i\in R_{b_nb_{n-1}\cdots\,\overline{b_{n+1-r}}*}} \left(1 - \left(1 - \frac{1}{2^{n+1-r}}\right)^{d_i}\right) = \frac{1}{2}A_{r-1} \qquad \text{for } r = 1,\cdots,n. \tag{6}$$

Here, B_r is half that of the balanced tree. To estimate the number of active pixels sent by each buffer, we average B_r over all buffers to give

$$\frac{B_r}{N} = \frac{A_{r-1}}{2N} = \frac{2^{n-r}}{2N}\left(\frac{\mu}{2^{n-r}} + \Theta\left(\left(\frac{1}{2^{n-r}}\right)^2\right)\right)S = \frac{\mu}{2N}S + \Theta\left(\frac{1}{2^{2n-r}}\right). \tag{7}$$

Therefore, each round of pixel swapping is about twice as fast when the N is doubled — a desirable property for scalability that was empirically observed by Li et al. [26]. The latency in each round is partly determined by the composition of active pixels received by each buffer with those remaining (i.e. unsent) in the buffer. But what is received in a round is also what is sent in that round, and the latter together with the remaining pixels make up the active area in the previous round. Therefore, the active composition volume is

$$C_r = \frac{1}{2^n}A_{r-1} = \frac{1}{2^{r-1}}\sum_{i\in I}\left(1 - \left(1 - \frac{1}{2^{n+1-r}}\right)^{d_i}\right) \quad \text{for } r = 1,\cdots,n. \tag{8}$$

This is exponentially $(1/2^r)$ smaller than that for a balanced tree. A_r, B_r and C_r are then aggregated to give B, L_B, L_C and W.

The exponential reduction in C_r from that for a tree is evident from Table 1: while W for binary-swap is the same as that for a balanced tree, the active composition latencies L_B and L_C for binary-swap actually decrease with N. Among the six techniques, only binary-swap and direct pixel forwarding (Section 3.4) have decreasing active latencies as more renderers are used. In fact, as N increases, the active latencies would decrease to 0; this seems too good to be true. Indeed, there are two caveats: first, active latencies do not include overheads (which are never zero) and second, L_B implicitly assumes all buffers can communicate in parallel for each round. If binary-swap is executed over a mesh, say, traffic contention among the buffers becomes unavoidable, and can cause latencies to increase with N, as was the case in Lee et al.'s experiment [24].

How much bandwidth is saved by composition? Since each renderer starts with $\mu S/N$ active pixels (1), if these are sent in all $\log N$ rounds without composition, the total active bandwidth volume would be $\mu(\log N)S$. It follows that, to a first approximation, composition reduces the bandwidth requirement B by half. (Composition similarly halves B for the tree.)

3.4 Direct Pixel Forwarding: Composition with a Mesh

Many parallel architectures use a two-dimensional mesh to connect the processors (e.g. the Intel Delta and Fujitsu AP1000), and Lee et al. have proposed some SL-sparse composition schemes for such machines [24]. One of them, called *Direct Pixel Forwarding*, is similar to hypercube composition, in that the renderers exchange pixels first along one axis of the mesh, then along the other axis; this is faster than two of their other schemes.

The architecture consists of renderers-cum-compositors arranged in an $n_{\text{row}} \times n_{\text{col}}$ mesh. We index the buffers by their mesh coordinates, so $F = \{(i,j) : 0 \le i < n_{\text{row}}, 0 \le j < n_{\text{col}}\}$ and there are $N = |F| = n_{\text{row}} n_{\text{col}}$ buffers. Each buffer is partitioned into $n_{\text{row}} n_{\text{col}}$ disjoint regions R_b, $b \in F$, as illustrated in Figure 7. As before, we adopt the usual meaning for '$*$'; for example, $R_{(i,*)} = \bigcup_j R_{(i,j)}$. There are $n = (n_{\text{row}} - 1) + (n_{\text{col}} - 1)$ rounds. The composition has two phases:
For round r in Phase 1, $r = 1, \ldots, n_{\text{row}} - 1$ (index addition/subtraction is modulo n_{row}),
buffer (i,j) sends pixels in region $R_{(i+r,*)}$ to buffer $(i+r,j)$; and
for round $(n_{\text{row}} - 1) + r$ in Phase 2, $r = 1, \ldots, n_{\text{col}} - 1$ (index addition/subtraction is modulo n_{col}),
buffer (i,j) sends pixels in region $R_{(i,j+r)}$ to buffer $(i, j+r)$.
This is illustrated in Figure 8 for a 3×4 mesh.

An analysis shows that, although (to a first approximation) active bandwidth volume B_r is constant for each round within a phase, and the number of rounds increases with the number of renderers N, the total active bandwidth volume B is approximately constant with respect to N (see Table 1). This is an advantage for scalability over composition with a pipeline, tree or hypercube. (Interestingly, composition does not save much bandwidth — if every active pixel is sent in each of the two phases without composition, the total active bandwidth volume would be $2\mu S$ approximately, like B in Table 1. This is unlike the reduction by half for the hypercube, and is because there are only 2 phases for the 2D mesh but $\log N$ rounds for the hypercube.)

But a bigger advantage is in active bandwidth latency L_B, which decreases with $1/N$, so this latency is halved when N is doubled. Among the six techniques, this is the best performance for L_B. Like binary-swap, direct pixel forwarding has an active composition latency L_C that decreases with N: If n_{row} and n_{col} are of order $\sqrt{N}$, then L_C decreases with $1/\sqrt{N}$, which is not as fast as the $(\log N)/N$ decrease for binary-swap. Similarly, the active work W increases with $\sqrt{N}$, which is slower than the pipeline, but still faster than the balanced tree and the hypercube.

3.5 Z-buffering by Snooping

The composition techniques discussed so far are applicable to any operator that is commutative and associative, i.e. they do not exploit any special feature of z-buffering. One such feature is that, if multiple objects render to the same pixel, then only one of them determines its color. This fact was used by Cox and Hanrahan to reduce bandwidth requirement with a technique similar to cache coherence by snooping in tightly-coupled systems [9].

As in the case of pipelining, the renderers are ordered $1, 2, \ldots, N$, with another z-buffer in a compositor labeled $N+1$. In round 1, renderer 1 broadcasts all its active pixels, say, on a bus shared by all buffers (see Figure 9). Buffer $N+1$ stores these pixels while, for $b = 2, \ldots, N$, each z-buffer b in a renderer listens to the broadcast; for each active pixel i in buffer b, if pixel i is also in the broadcast and the latter has smaller depth than the former, then the former is deleted from buffer b. This deletion distinguishes z-buffering from other composition operators that may conceivably use snooping. Subsequently, in round r, buffer r broadcasts its undeleted active pixels; buffer $N+1$ stores any pixel that is new or hides one of its active pixels, while buffers $r+1, \cdots, N$ discard hidden pixels as described. When buffer N finishes broadcasting, buffer $N+1$ has the composited image. We first prove the following:

Claim 2 The expected number of pixels broadcast in round r of z-buffering by snooping is

$$\sum_{i\in I} \frac{1}{r}\left(1 - \left(1 - \frac{r}{N}\right)^{d_i}\right) \quad \text{for } r = 1, \ldots, N, \quad \text{where } 1 - \left(1 - \frac{N}{N}\right)^0 \overset{\text{def}}{=} 0.$$

Proof
Let $<_i$ be the total order (by depth) on all the objects that render to pixel i (i.e. $\mathcal{O}_j(i) = 1$), and define z_i on $\{1, \ldots, J\}$ by

$$z_i(j) = \begin{cases} \infty & \text{if } \mathcal{O}_j(i) = 0 \\ k & \text{if } \mathcal{O}_j(i) = 1 \text{ and there are exactly } k-1\ \mathcal{O}_{j'} \text{ such that } \mathcal{O}_{j'} <_i \mathcal{O}_j. \end{cases}$$

Thus, for $d_i \neq 0$, $z_i^{-1} : \{1, \ldots, d_i\} \to \{1, \ldots, J\}$ is well-defined.

Let the buffers be numbered $1, \ldots, N$, according to their broadcast order. For $d_i = 0$, the probability is 0 that any buffer must broadcast for pixel i. For $d_i \neq 0$, the probability that buffer r must broadcast for pixel i is

$$\begin{aligned}
&\Pr(X_{z_i^{-1}(1)} = r) + \sum_{k=2}^{d_i} \Pr(X_{z_i^{-1}(k)} = r \text{ and } X_{z_i^{-1}(h)} > r \text{ for } h = 1, \ldots, k-1) \\
&= \frac{1}{N} + \sum_{k=2}^{d_i} \Pr(X_{z_i^{-1}(k)} = r) \prod_{h=1}^{k-1} \Pr(X_{z_i^{-1}(h)} > r) \quad \text{since } X_1, \ldots, X_J \text{ are independent} \ . \\
&= \frac{1}{N} + \sum_{k=2}^{d_i} \frac{1}{N}\left(1 - \frac{r}{N}\right)^{k-1} = \frac{1}{N} + \frac{1}{N}\left(1 - \frac{r}{N}\right)\frac{1 - \left(1 - \frac{r}{N}\right)^{d_i - 1}}{1 - \left(1 - \frac{r}{N}\right)}
\end{aligned}$$

Therefore, for any i, the probability that buffer r must broadcast for that pixel is $\frac{1}{r}\left(1 - \left(1 - \frac{r}{N}\right)^{d_i}\right)$. The claim follows. □

Now, just after round r, the z-buffered result of the images in renderers $1, \ldots, r$ will be in buffer $N+1$. The other active pixels are in buffers $r+1, \ldots, N$, who (by symmetry) have the same expected number of active pixels left. This number is what renderer $r+1$ will broadcast in round $r+1$. By Claim 2, we therefore have active area

$$A_r = \sum_{i \in I}\left(1 - \left(1 - \frac{r}{N}\right)^{d_i}\right) + (N - r)\sum_{i \in I} \frac{1}{r+1}\left(1 - \left(1 - \frac{r+1}{N}\right)^{d_i}\right) \quad \text{for } r = 1, \ldots, N-1. \tag{9}$$

$(A_N = S(1 - f_0).)$ Claim 2 gives directly

$$B_r = \frac{1}{r}\sum_{i \in I}\left(1 - \left(1 - \frac{r}{N}\right)^{d_i}\right) \quad \text{for } r = 1, \ldots, N. \tag{10}$$

Hence, B_r decreases as r increases, unlike the case of pipelining. In each round, buffer $N+1$ has (on average) at least as many active pixels as any other buffer. Since, in round r, each of buffers $r+1, \ldots, N+1$ must process the received pixels as well as their local active pixels, buffer $N+1$ determines the active composition volume, thus:

$$C_r = \sum_{i \in I}\left(1 - \left(1 - \frac{r-1}{N}\right)^{d_i}\right) + \frac{1}{r}\sum_{i \in I}\left(1 - \left(1 - \frac{r}{N}\right)^{d_i}\right) \quad \text{for } r = 1, \ldots, N. \tag{11}$$

Here, C_r increases with r, but this increase is slowed down by the decrease in B_r (second summation). A_r, B_r and C_r are then aggregated to give B, L_B, L_C and W in Table 1.

In particular, the expression $\eta S = S\sum_m H_m f_m$ for B is similar to the one obtained by Cox and Hanrahan, which was $S\sum_m H_m p_m$, where p_m was the probability that pixel depth was m in their model. (Cox's measurements showed that B was indeed roughly constant for large N.) Their approximation was based on three assumptions: (1) the pixel depth at any renderer was at most 1, (2) the z-values of the objects that render to a given pixel were randomly and independently distributed, and (3) the z-values to be broadcast were independent of the order of the renderers that must broadcast them. These assumptions, while astute, are hard to justify, may be violated (as in the case of Zinnia [9]), and trivialize the analysis. We have thus shown that, if one uses an object-based analytic technique instead, then their questionable assumptions are, in fact, unnecessary.

	total active bandwidth volume B	active bandwidth latency L_B	active composition latency L_C	active work W
Centralized Composition	μS	μS	$\rho N S$	$\rho N S$
Pipeline	$\rho N S$	$\rho N S$	$\rho N S$	$\rho N S$
Balanced Tree	$\mu(\log N)S$	$< 2\xi(\log N)S$	$< 2\xi(\log N)S$	$\mu(\log N)S$
Binary-Swap (Hypercube)	$\frac{\mu}{2}(\log N)S$	$\frac{\mu}{2}\frac{\log N}{N}S$	$\mu\frac{\log N}{N}S$	$\mu(\log N)S$
Direct Pixel Forwarding (Mesh)	$2\mu S$	$\frac{2\mu}{N}S$	$< \left(\frac{\mu}{2}+\rho\right)\frac{1}{\sqrt{N}}S$	$< \left(\frac{\mu}{2}+\rho\right)\sqrt{N}S$
Snooping (Bus)	ηS	ηS	$\rho N S$	$(2\eta - \rho)NS$

$\xi = 1 - \sum_{m=0}^{J}(\frac{1}{2})^m f_m$
$\eta = \sum_{m=0}^{J} H_m f_m$ where $H_m = 1 + \frac{1}{2} + \cdots + \frac{1}{m}$ $(H_0 = 1)$, $H_m \approx \log m$ for large m
$\rho = \sum_{m=0}^{J} \frac{m}{m+1} f_m$

Table 1 A comparison of the pixel complexity in six composition techniques.
(N, S, μ, J, f_m are defined in Table 2;
L_C and W for direct pixel forwarding are stated for a square mesh.)

I	index set for pixels
S	number of pixels in a frame buffer, $S = \|I\|$
$\mathcal{O}$	object, $\mathcal{O} : I \to \{0, 1\}$
J	number of objects
d_i	original pixel depth at pixel i
f_m	fraction of pixels with original pixel depth m, $f_m = \|\{i : d_i = m\}\|/S$
μ	average original pixel depth, $\mu = \sum_{m=0}^{J} m f_m$
σ^2	variation in original pixel depth, $\sigma^2 = \sum_{m=0}^{J} (m - \mu)^2 f_m$
F	index set for frame buffers
N	number of renderers
X_j	random variable for buffer to which $\mathcal{O}_j$ is assigned
p_b	probability that $\mathcal{O}_j$ is assigned to (renderer or compositor) buffer b, $p_b = \Pr(X_j = b)$
$D_b(i)$	pixel depth at i in buffer b, $D_b(i) = \sum_{j=1}^{J} \mathcal{O}_j(i)\delta(X_j, b)$
α	expected fraction of active pixels in a renderer's buffer
β	expected fraction of pixels in a renderer's buffer with pixel depth exceeding 1
γ	expected fraction of pixels that need not be sent for compositing if renderers do z-buffering
A_r	active area just after round r
B_r	active bandwidth volume during round r
C_r	active composition volume during round r
B	total active bandwidth volume
C	active composition latency
W	active work

Table 2 Glossary

	Bike	*Cube*	*Zinnia*	*Roses*	*Wash*	*Brooks*	*Galaxy*	*Rad*
μ	2.41	8.12	0.53	4.88	3.14	3.40	24.59	1.12
σ	1.99	7.63	1.22	6.07	3.18	5.24	34.82	1.03

(a) Cox and Hanrahan's scenes.

	Space	*Poliovirus*	*Lobby*	*House*	*Earth*	*Pipes*
μ	2.60	18.38	5.58	7.98	4.51	6.66
σ	2.45	14.25	4.09	9.04	4.32	7.54

(b) Molnar's scenes.

Note: Objects are defined differently for different scenes and by different rendering systems.

Table 3 Average and variation in original pixel depth for some scenes.

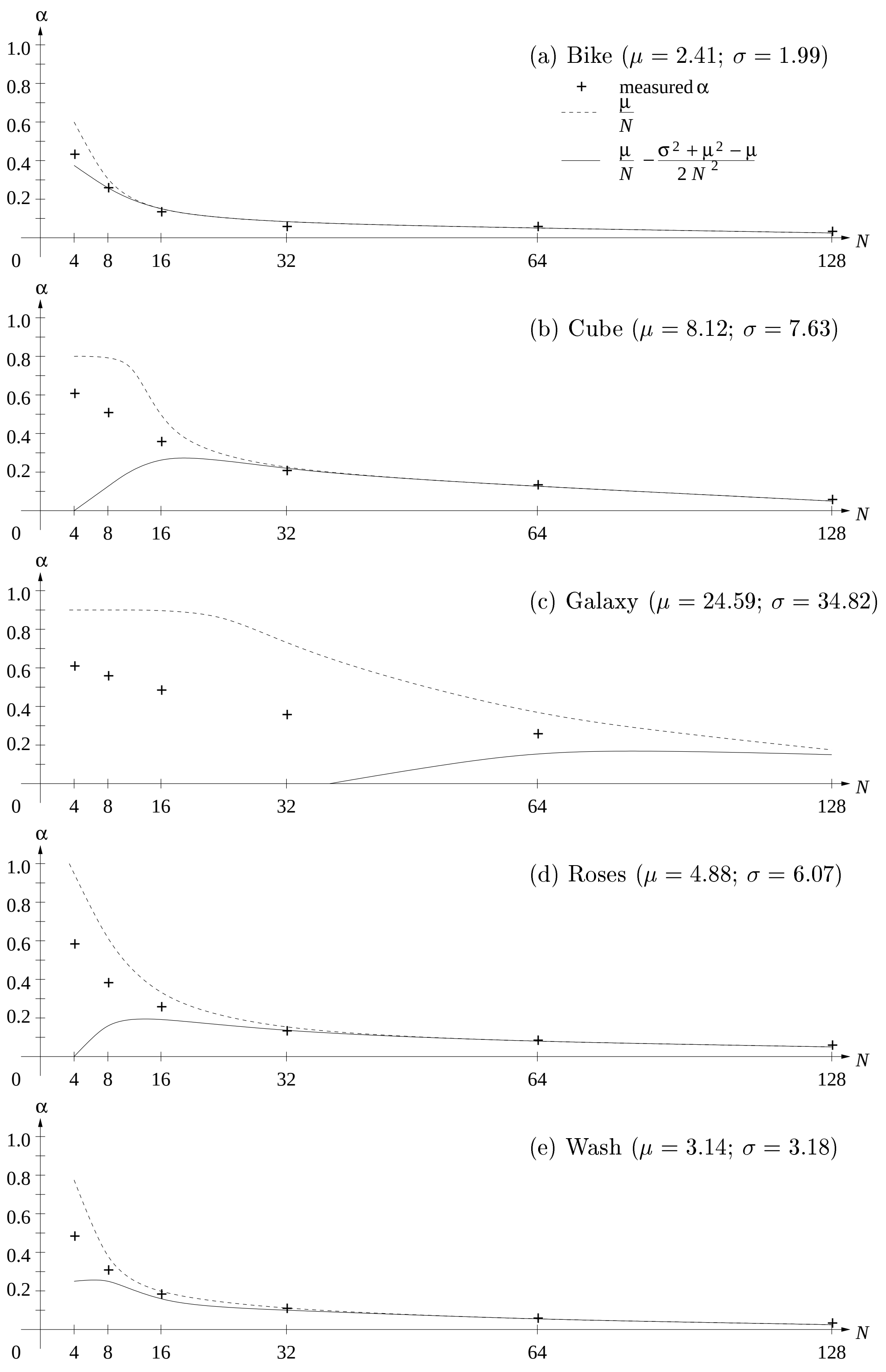


Figure 1 Comparison of closed-form approximations for α to Cox's measurements. α is the expected fraction of active pixels in a renderer's buffer.

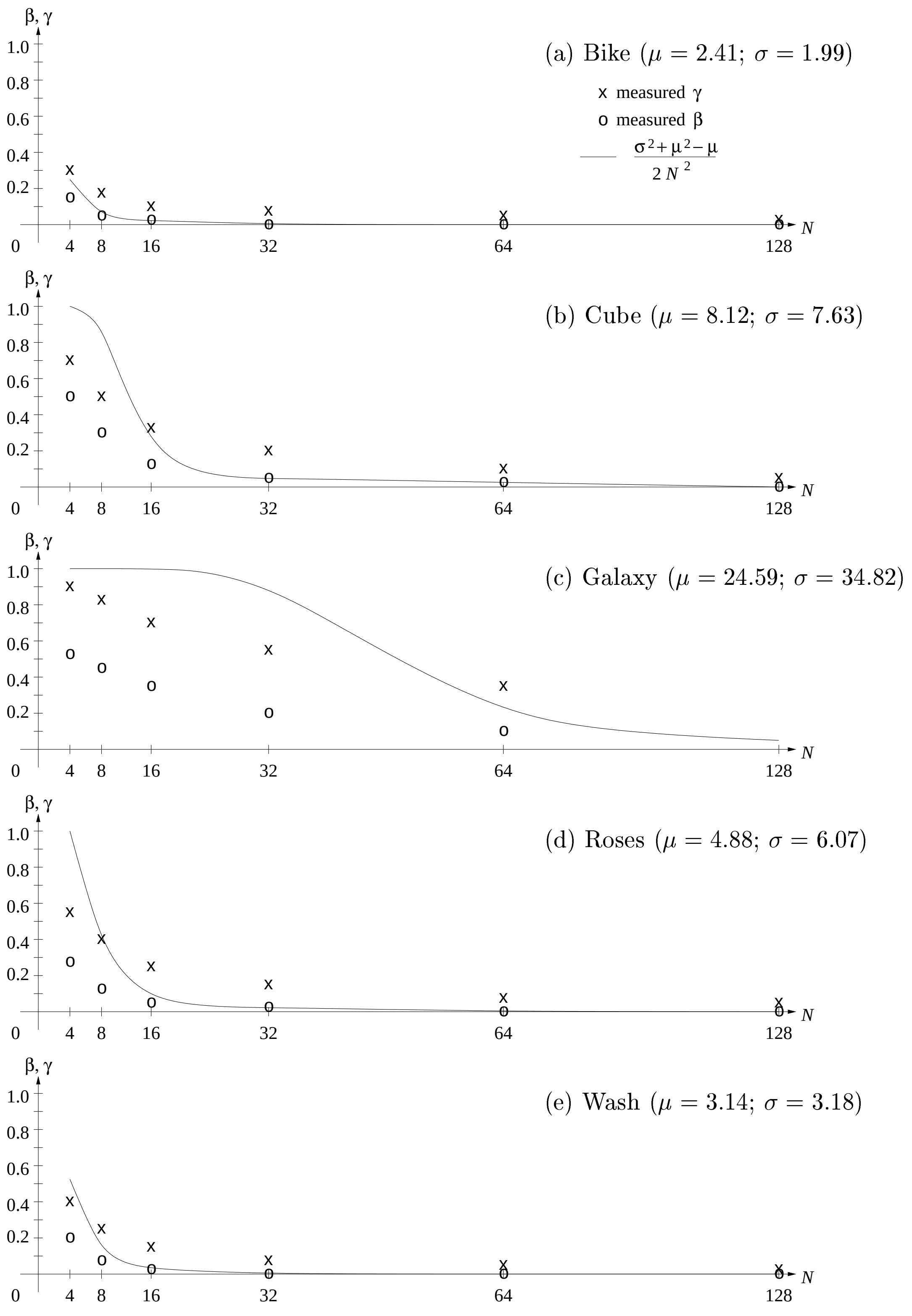


Figure 2 Comparison of closed-form approximations for β and γ to Cox's measurements. β is the expected fraction of pixels with pixel depth exceeding 1; γ is the expected fraction of pixels that need not be sent for compositing if renderers do z-buffering.

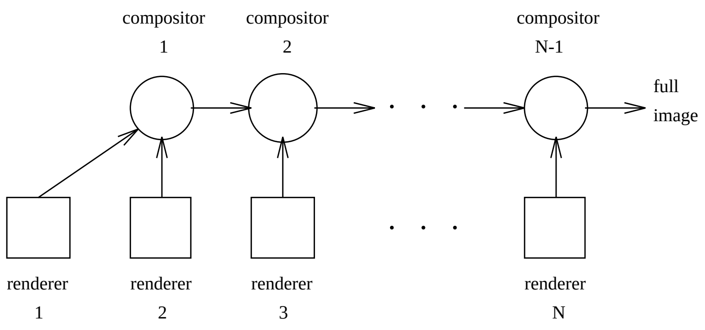


Figure 3 Composition in a pipeline.

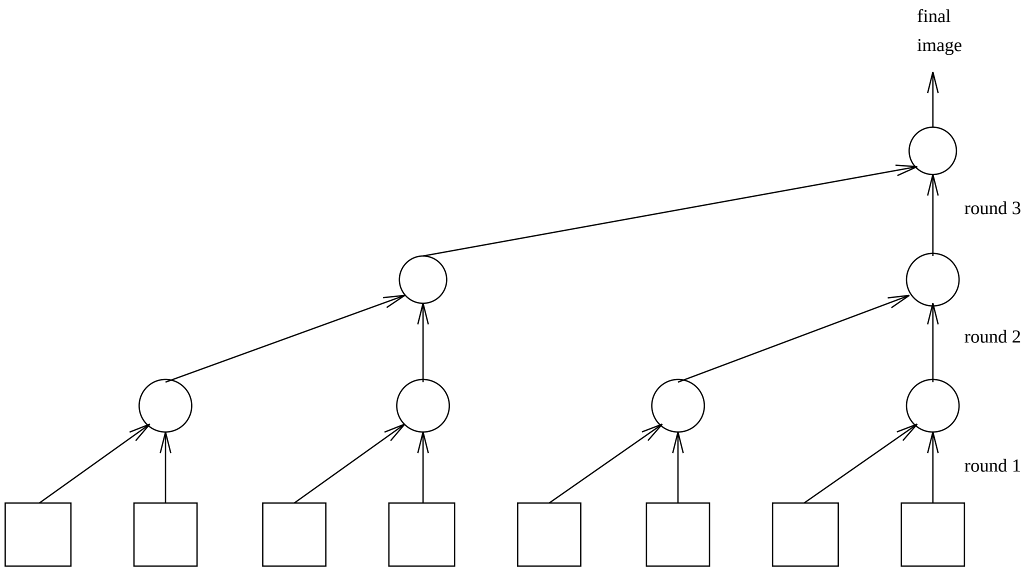


Figure 4 Composition in a balanced tree.

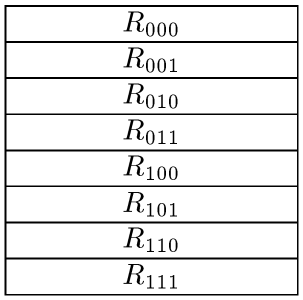


Figure 5 A frame buffer divided by scanlines into 8 regions for binary-swap.

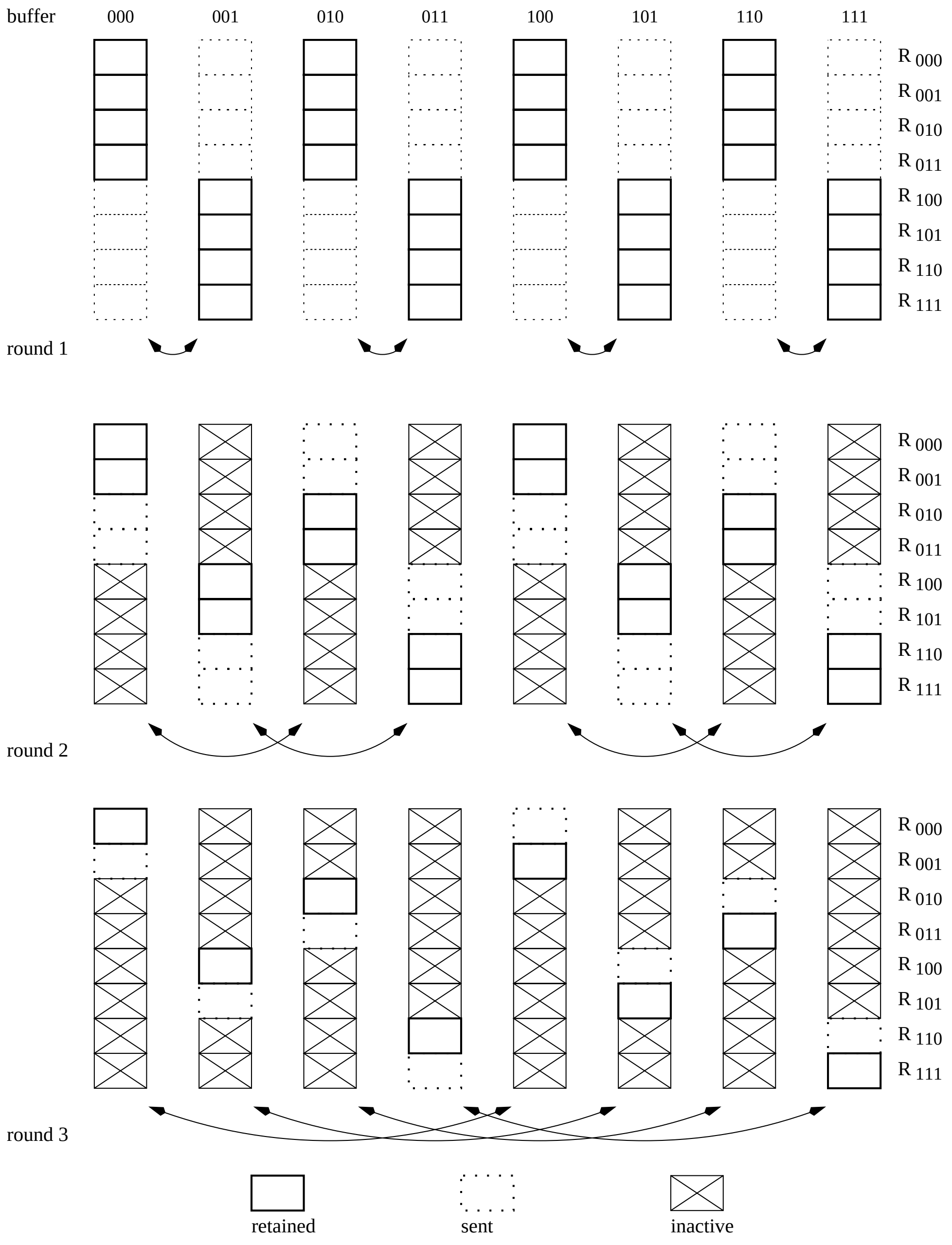
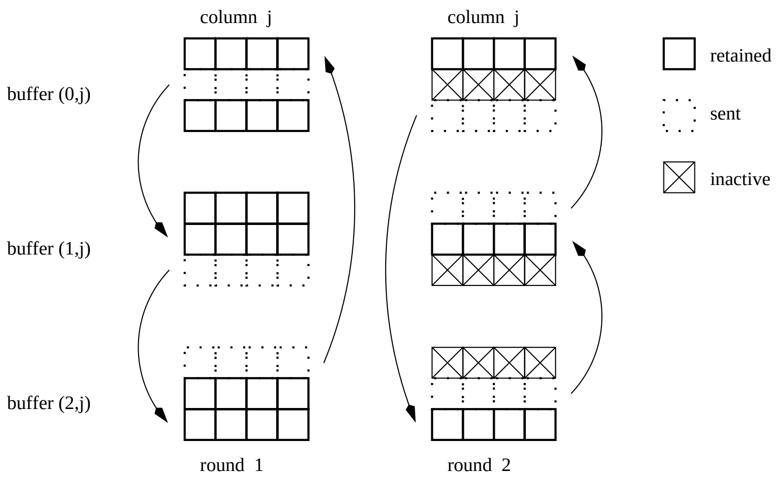


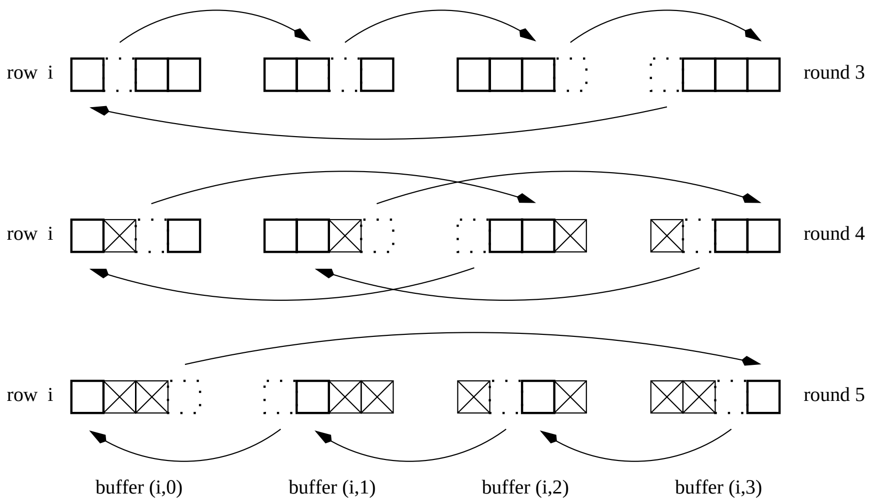
Figure 6 Binary swap for 3-dimensional hypercube.

$R_{(0,0)}$	$R_{(0,1)}$	$R_{(0,2)}$	$R_{(0,3)}$
$R_{(1,0)}$	$R_{(1,1)}$	$R_{(1,2)}$	$R_{(1,3)}$
$R_{(2,0)}$	$R_{(2,1)}$	$R_{(2,2)}$	$R_{(2,3)}$

Figure 7 A frame buffer divided into 12 regions for direct pixel forwarding.

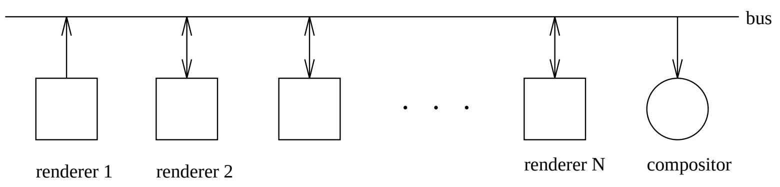


Phase 1 Each column of buffers executes above rounds.
Result: each buffer has one row of active regions left.



Phase 2 Active regions in each row of buffers execute above rounds.
Result: each buffer has one active region.

Figure 8 The two phases in direct pixel forwarding for a 3×4 mesh.



Arrows indicate the directions of pixel traffic during composition.
Figure 9 Composition by snooping on a bus.

References

[1] B. Apgar, B. Bersack and A. Mammen, *A display system for the Stellar^{TM} graphics supercomputer model GS1000^{TM}*, Proc. SIGGRAPH '88, Atlanta, GA(Aug. 1988), 255–262.

[2] P.R. Atherton, *A method of interactive visualization of CAD surface models on a color video display*, Proc. SIGGRAPH '81, Dallas, TX(Aug. 1981), 279–287.

[3] B. Bäumle, Kohler and Gunzinger, *Interactive parallel rendering on a multiprocessor system with intelligent communication controllers*, Proc. Parallel Rendering Symposium, Atlanta, GA (Oct. 1995), 89–95.

[4] M. Barnett, R. Littlefield, D.G. Payne and R. van de Geijn, *Global combine on mesh architectures with wormhole routing*, Proc. Int. Parallel Processing Symposium, Newport Beach, CA (Apr. 1993), 156–162.

[5] L. Carpenter, *The A-buffer, an antialiased hidden surface method*, Proc. SIGGRAPH '84, Minneapolis, MN(July 1984), 103–108.

[6] U. Claussen, *Parallel subpixel scanconversion*, Advances in Graphics Hardware II, Springer-Verlag (1987), 155–166.

[7] M. Cox, *Algorithms for Parallel Rendering*, PhD dissertation, Princeton University (1995).

[8] M. Cox and P. Hanrahan, *Depth complexity in object-parallel graphics architectures*, Eurographics Technical Report Series (1992), 204–222.

[9] M. Cox and P. Hanrahan, *A distributed snooping algorithm*, IEEE Parallel and Distributed Technology 2, 2(1994), 30–36.

[10] M. Deering, *Data complexity for virtual reality: where do all the triangles go?*, Proc. IEEE Virtual Reality Annual Int. Symp. '93, Seattle, WA(Sept. 1993), 357–363.

[11] 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*, Proc. SIGGRAPH '88, Atlanta, GA(Aug. 1988), 21–30.

[12] T. Duff, *Compositing 3-D rendered images*, Proc. SIGGRAPH '85, San Francisco, CA(July 1985), 41–44.

[13] D. Ellsworth, H. Good and B. Tebbs, *Distributing display lists on a multicomputer*, Computer Graphics 24, 2(Mar. 1990), 147–154.

[14] T.T. Elvins, *Volume rendering on a distributed memory parallel computer*, Proc. Workshop on Volume Visualization, Boston, MA (Oct. 1992), 93–98.

[15] Evans and Sutherland Computer Corporation, *Freedom Series Technical Report*, Salt Lake City, UT (Oct. 1992).

[16] Fujitsu Limited, *AG Series Graphics Technical Overview*, Fujitsu Open Systems Solutions, San Jose, CA (1993).

[17] D. Fussel and B.D. Rathi, *A VLSI-oriented architecture for real-time raster display of shaded polygons*, Graphics Interface '82, 373–380.

[18] N. Greene, M. Kass and G. Miller, *Hierarchical z-buffer visibility*, Proc. SIGGRAPH '93, Annaheim, CA (Aug. 1993), 231–238.

[19] W.M. Hsu, *Segmented ray casting for data parallel volume rendering*, Proc. Parallel Rendering Symposium, San Jose, CA (Oct. 1993), 7–14.

[20] R.J. Karia, *Load balancing of parallel volume rendering with scattered decomposition*, Proc. Scalable High Performance Computing Conference, Knoxville, TN (May 1994), 252–258.

[21] A. Kaufman and R. Bakalash, *Memory and processing architecture for 3D voxel-based imagery*, IEEE Computer Graphics and Applications 8, 6(Nov. 1988), 10–23.

[22] Kubota Pacific Computer, *Denali Technical Overview*, version 1.0, Santa Clara, CA (Mar. 1993).

[23] L. Lamport and N. Lynch, *Distributed computing: models and methods.* In *Handbook of Theoretical Computer Science, Volume B* (J. van Leeuwen, ed.), North-Holland, Amsterdam (1990), 1157–1200.

[24] T.-Y. Lee, C.S. Raghavendra and J.B. Nicholas, *Image composition schemes for sort-last polygon rendering on 2-D mesh multicomputers*, IEEE Trans. Visualization and Computer Graphics 2, 3(Sept. 1996), 202–217.

[25] W. Lefer, *An efficient parallel ray tracing scheme for distributed memory parallel computers*, Proc. Parallel Rendering Symposium, San Jose, CA (Oct. 1993), 77–80.

[26] P.P. Li, W.H. Duquette and D.W. Curkendall, *Remote interactive visualization and analysis (RIVA) using parallel supercomputers*, Proc. Parallel Rendering Symposium, Atlanta, GA (Oct. 1995), 71–78.

[27] J. Li and S. Miguet, *Z-buffer on a transputer-based machine*, Proc. Distributed Memory Computing Conf., Portland, OR (April 1991), 315–322.

[28] K. Ma, J.S. Painter, C.D. Hansen, M.F. Krogh, *Parallel volume rendering using binary-swap compositing* IEEE Computer Graphics and Applications 14, 4(July 1994), 59–68.

[29] P. Mackerras, *A fast parallel marching cubes implementation on the Fujitsu AP1000*, Technical Report TR-CS-92-10, Dept. of Computer Science, Australian National University, 1992.

[30] S. Molnar, *Combining z-buffer engines for higher-speed rendering*, Advances in Graphics Hardware III, Springer-Verlag (1988).

[31] S. Molnar, *Image-composition architectures for real-time image generation*, PhD dissertation, University of North Carolina at Chapel Hill (Oct. 1991).

[32] S. Molnar, M. Cox, D. Ellsworth and H. Fuchs, *A sorting classification of parallel rendering*, IEEE Computer Graphics and Applications 14, 4(July 1994), 23–32.

[33] S. Molnar, J. Eyles and J. Poulton, *PixelFlow: High-speed rendering using image composition*, Computer Graphics 26, 2(July 1992), Proc. SIGGRAPH '92, Chicago, IL (July 1992), 231–240.

[34] U. Neumann, *Communication costs for parallel volume-rendering algorithms*, IEEE Computer Graphics and Applications 14, 4(July 1994), 49–58.

[35] C. Papadimitriou and M. Sipser, *Communication complexity*, Proc. ACM Symposium on Theory of Computing (1982), 196–200.

[36] F. Parke, *Simulation and expected performance analysis of multiple z-buffer systems*, Proc. SIGGRAPH '80 (July 1980), 48–56.

[37] C.R. Ramakrishnan and C.T. Silva, *Optimal processor allocation for sort-last compositing under BSP-tree ordering*, TR 9728, Dept. Applied Mathematics and Statistics, SUNY Stony Brook (Oct. 1997).

[38] G.C. Roman and T. Kimura, *A VLSI architecture for real-time color display of three-dimensional objects*, Proc. IEEE Micro-Delcon (1979), 113–118.

[39] J.R. Rossignac and A.A.G. Requicha, *Depth-buffering display techniques for constructive solid geometry*, IEEE Computer Graphics and Applications 6, 9(Sept. 1986), 29–39.

[40] K. Sano, H. Kitajima, H. Kobayashi and T. Nakamura, *Parallel processing of the shear-warp factorization with the binary-swap method on a distributed-memory multiprocessor system*, Proc. Parallel Rendering Symposium, Phoenix, AZ (Oct. 1997), 87–94.

[41] A. Schilling and W. Straßer, *EXACT: Algorithm and hardware architecture for an improved A-buffer*, Proc. SIGGRAPH '93, Annaheim, CA (Aug. 1993), 85–91.

[42] B.-O. Schneider and U. Claussen, *PROOF: An architecture for rendering in object space*, Advances in Graphics Hardware III, Springer-Verlag (1988), 121–140.

[43] R. Scopigno, A. Paoluzzi, S. Guerrini and G. Rumolo, *Parallel depth-merge: A paradigm for hidden surface removal*, Computers and Graphics 17, 5(1993), 583–592.

[44] C.D. Shaw, M. Green and J. Schaeffer, *A VLSI architecture for image composition*, Advances in Graphics Hardware III, Springer-Verlag (1988), 183–199.

[45] T.A. Theoharis, *Algorithms for parallel polygon rendering*, Springer-Verlag, New York, NY (1989).

[46] R. Weinberg, *Parallel processing image synthesis and antialiasing*, Proc. SIGGRAPH '81, Dallas, TX(Aug. 1981), 55–62.