Evenly Distributed Depth is the Worst for Distributed Snooping

Michael Cox
Pat Hanrahan

Princeton University*

Abstract

In the purely object-parallel approach to multiprocessor rendering, each processor is assigned responsibility to render a subset of the graphics database. When rendering is complete, pixels from the processors must be merged and globally z-buffered. On an arbitrary multiprocessor interconnection network, the straightforward algorithm for pixel merging requires $\bar{d}A$ total network bandwidth per frame, where $\bar{d}$ is the depth complexity of the scene and A is the area of the screen or window. In another paper we have presented a *distributed snooping algorithm* for pixel merging that requires $\log(\bar{d})A$ expected network bandwidth per frame. In that paper, to simplify the analysis we required the assumption that depth is "evenly" distributed over the screen, and we claimed that this assumption represented a worst-case for expected traffic. In the current paper we prove this claim.

1 Introduction

In [2], we presented a *distributed snooping algorithm* for pixel merging in purely object-parallel graphics architectures. We showed that the expected network traffic of this algorithm is $\log(\bar{d})A$ pixels per frame, assuming that pixel depth is "evenly" distributed over the screen. In the current paper we show that this assumption represents a worst-case for expected traffic, and thus that the analysis in [2] is conservative. Before proceeding, we first reproduce for completeness the terms and concepts, the presentation of the distributed snooping algorithm, and the derivation of expected traffic from [2].

2 Terms and concepts

We will require some terms. Let

$d_{x,y}$ = the *depth* at pixel location (x, y).

$\bar{d}$ = the *depth complexity* of a given scene; that is the average depth over the screen.

p_d = the probability that at arbitrary pixel location (x, y), $d_{x,y} = d$.

A = the resolution of the screen or window, in pixels.

n = the number of processors in the multiprocessor.

Consider one pixel location (x, y). When a graphics scene is rendered, there may be multiple graphics primitives that generate a pixel for (x, y). These multiple pixels are the motivation behind z-buffering. For any given scene, we refer to the number of primitives that render to a given (x, y) as the *depth* $d_{x,y}$ at (x, y). We refer to the average depth $\bar{d}$ over the screen as the *depth complexity* of the scene. If depth is not evenly distributed over (x, y) then we also refer to the probability p_d that at arbitrary (x, y), the depth $d_{x,y} = d$.

*Department of Computer Science, 35 Olden St., Princeton, NJ 08540.

We distinguish *active* from *inactive* pixels at each processor. As each processor renders its subset of the scene, it produces pixels for *some* of the A pixel locations. We say that a pixel location is *active* if at least one pixel has been rendered to that (x, y), and that it is *inactive* otherwise.

3 A distributed snooping merge algorithm

We propose in [2] the following algorithm for pixel merging when network broadcast is available, for example as it is on a shared bus. Pixel merging begins after all processors have completed rendering, and proceeds as follows. Consider the n processors in a multiprocessor ordered from $1 \leq i \leq n$, and suppose there is a global frame buffer to which can be broadcast pixel values. Processor 1 first broadcasts to the global frame buffer each pixel it has rendered (that is, it broadcasts all *active* pixels from its local frame buffer). While processor 1 broadcasts, *each of the other processors listens*, and compares each broadcast pixel with the corresponding pixel in its local memory. If the broadcast pixel "hides" the local pixel, the listening processor deletes the hidden pixel from local memory; if the local pixel hides the broadcast pixel, the snooping processor *does nothing yet.* In turn, then, each processor k broadcasts its active pixels from local memory *that have not already been deleted*, and each processor j, $k < j \leq n$ listens and potentially deletes pixels from its local memory. When all processors have broadcast to the global frame buffer all their non-deleted local pixels, pixel merging is complete, and the global frame buffer contains a correctly z-buffered image.

This algorithm is a derivative of the well-known "snooping" cache coherency protocols upon which a number of shared-memory multiprocessors have been based (cf. [3]) In these architectures, each node maintains the consistency of its cache with other nodes' memory accesses by snooping on a shared bus over which memory updates must be written. In these protocols for cache coherency, snooping is employed in order to maintain consistent globally shared virtual memory. In our scheme, snooping is employed in order to remove pixels, wherever possible, from consideration by pixel merging, and thus to reduce the bandwidth required on a shared broadcast medium such as a bus.

Consider the network bandwidth required by this algorithm. In the worst-case, a sequence of planes ordered back-to-front is assigned to processors such that the first processor renders the backmost plane, the last processor renders the frontmost frame. This results in network traffic of nA pixels per frame. But this scene is uninteresting from the perspective of animation or visualization. We turn now to an analysis of the expected-case performance of the algorithm.

3.1 Expected-case network traffic of distributed snooping

We first assume that after rendering, there is no pixel location in any *local frame buffer* at which $d_{x,y} > 1$. In essence, we assume that no two primitives that render to (x, y) are assigned to the same processor. This means that no pixels are deleted by local z-buffering, and represents a worst-case assumption for the expected traffic of the snooping algorithm [2]. Previous work has suggested that for machines larger than about 8 processors, in fact few pixels are deleted in object-parallel architectures by local z-buffering [1]. Second, we assume that over all A pixel locations, the z-values of the $d_{x,y}$ pixels that render to (x, y) are randomly and independently distributed. Third, we will assume that the pixel z-values to be broadcast are independent of the order of the processors that must broadcast them. From these assumptions it follows that the probability that the k^{th} broadcast pixel (at (x, y)) is frontmost is $1/k$.

Now, consider a single pixel location (x, y). The first processor with an active pixel at (x, y) must broadcast its pixel, and must do so with probability 1. The second processor that has (x, y) active must on average broadcast its pixel with probability $1/2$, since it has the frontmost of two pixels with probability $1/2$. The third processor that has (x, y) active broadcasts a pixel with probability $1/3$, and so on. Since no pixels have been previously deleted at (x, y) by local z-buffering, the pixel traffic expected from location (x, y) is

$$ET_{x,y} = \sum_{1 \leq d \leq d_{x,y}} 1/d \quad (1)$$

$$= H_{d_{x,y}} \quad (2)$$

where $H_{d_{x,y}}$ is the harmonic number for $d_{x,y}$.

If we know the distribution of depth over A, the expected traffic from all pixel locations is

$$ET = A \sum_{1 \leq d} p_d H_d \quad (3)$$

Some work characterizing distributions of depth in graphics scenes has been done (cf. [1]). However, we can safely bound expected traffic. We will say that depth is "evenly" distributed over A if for all (x, y), $\lfloor \bar{d} \rfloor \leq d_{x,y} \leq \lceil \bar{d} \rceil$. We

show in the following section that this represents a worst-case assumption for the expected traffic of the snooping algorithm; that is, when graphics scene depth is evenly distributed, the expected-case traffic of the algorithm is worse than it is for any other distribution of depth. Thus, given $\alpha = (\bar{d} - \lfloor \bar{d} \rfloor)$, we bound ET by

$$ET \leq A\left[(1-\alpha)H_{\lfloor \bar{d} \rfloor} + \alpha H_{\lceil \bar{d} \rceil}\right] \tag{4}$$

where $H_{\lfloor \bar{d} \rfloor}$ and $H_{\lceil \bar{d} \rceil}$ are the harmonic numbers for $\lfloor \bar{d} \rfloor$ and $\lceil \bar{d} \rceil$. Now, harmonic numbers grow very slowly, about as $\log(\bar{d})$. Thus, as the depth at (x, y) increases, expected traffic grows only about as the log() of the average depth.

4 Proof that evenly distributed depth is worst

We must show that equation 3 is bounded by equation 4. We will do so by identifying equivalence classes of graphics scenes, and by showing that the equivalence class of evenly distributed scenes leads to the worst expected-case network traffic.

First, without loss of generality, we will consider all graphics scenes as rendering to the same screen area A. Now, let

$\#_S(d)$ denote the number of (x, y) in graphics scene S for which $d_{x,y} = d$.

Note that all scenes S' for which $\#_S(d) = \#_{S'}(d)$, $1 \leq d \leq n$ form an equivalence class with respect to distribution of scene depth complexity, and also with respect to expected bus traffic ET. We will choose some S as a representative of this class and refer to the class itself as S. Furthermore, we will call an equivalence class R *adjacent* if for some $d_a \neq d_b$ $\#_R(d_a) = (\#_S(d_a) - 1)$ and $\#_R(d_b) = (\#_S(d_b) + 1)$, but for all $d_c \neq d_a$ and $d_c \neq d_b$, $\#_R(d_c) = \#_S(d_c)$.[1] Note that in general for any S there are two adjacent equivalence classes: (1) the class in which $d_a > d_b$, which we will say is the *more evenly distributed adjacent equivalence class*, and (2) the class in which $d_b > d_a$, which we will say is the *less evenly distributed adjacent equivalence class*.

Now, consider expected traffic in each of two adjacent equivalence classes, S and R, where R is more evenly distributed (that is, $d_a > d_b$); let ET_S = the expected traffic for scenes in class S, and note

$$\begin{aligned} ET_S = \quad & \textstyle\sum_{d=1}^{d_a-1} \#_S(d)H_d + \quad \sum_{d=d_b+1}^{d_a-1} \#_S(d)H_d + \quad \sum_{d=d_a+1}^{n} \#_S(d)H_d + \\ & \#_S(d_a)H_{d_a} + \qquad \#_S(d_b)H_{d_b} \end{aligned}$$

Therefore, ET_R, the expected traffic for scenes in class R, is

$$\begin{aligned} ET_R = \quad & ET_S + \quad H_{d_b} \; - \; H_{d_a} \\ & ET_S + \quad \textstyle\sum_{k=1}^{d_b} 1/k - \quad \sum_{k=1}^{d_a} 1/k \\ & ET_S - \quad \textstyle\sum_{d_a+1}^{d_b} 1/k \end{aligned}$$

Thus, $ET_R > ET_S$, and from an arbitrary scene, if there is an adjacent equivalence class that is more evenly distributed, that adjacent class results in more expected bus traffic. By induction on the number of adjacent equivalence classes, it can be verified that the class that is "most" evenly distributed leads to worst-case expected bus traffic. Finally, it can be verified by the definition of a more evenly distributed adjacent graphics scene equivalence class (above), that our definition of the evenly distributed graphics scene of section 3.1 in fact has no more evenly distributed adjacent equivalence class. Thus, expected bus traffic is bounded by the theorem's equation.

5 Conclusions

In [2] we derived the expected traffic from a distributed snooping algorithm for pixel merging. We made the assumption in that derivation that pixel depth is evenly distributed over the screen, and we claimed that this represented a worst-case assumption. In the current paper we have proved this claim.

[1] This can be viewed as "moving" a single pixel from one pixel location to another such that the distribution of scene depth complexity changes; however, note that we do not actually "move" a pixel – rather, we identify scenes with such depth complexity distributions.

6 Acknowledgements

We would like to thank Frank Crow and Apple Computer, Kai Li and Matsushita Information Technology Laboratory, and DARPA (grant DABT63-92-C-0053) for funding of this work. We would also like to thank David Dobkin for useful discussion.

References

[1] Cox, Michael, and Pat Hanrahan, "Depth Complexity in Object-Parallel Graphics Architectures," *Proceedings of the Seventh Workshop on Graphics Hardware, Eurographics '92*, Cambridge, England, September 1992.

[2] Cox, Michael, and Pat Hanrahan, "Pixel Merging for Object-Parallel Rendering: a Distributed Snooping Algorithm" *Proceedings of the Parallel Rendering Symposium, Visualization '93*, San Jose, CA, October 1993.

[3] Hennessy, John, and David Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Mateo CA, 1990.