Embedded Function Composition

Turner Whitted Jim Kajiya Erik Ruf Ray Bittner

Microsoft Research*

Abstract

A low-level graphics processor is assembled from a collection of hardwired functions of screen coordinates embedded directly in the display. Configuration of these functions is controlled by a buffer containing parameters delivered to the processor on-the-fly during display scan. The processor is modular and scalable in keeping with the demands of large, high resolution displays.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture—Graphics processors

Keywords: display processor, large display, representation

1 Introduction

For the past few decades the output target for graphics processors has been a frame buffer memory intimately connected to the display processor and less intimately connected to the display device. As others have pointed out [Watson and Luebke 2005] large, dense displays place a strain on this arrangement. Matching copies of the entire conventional apparatus of CPU/Memory/GPU to tiled displays has been tried in numerous forms [Li et al. 2000] but that may not be the most economical solution for future applications.

In this paper we describe a display processor embedded in the display and dedicated to low level functions which can be inserted into the video refresh path. The display is assembled from tiles with one processor module attached to each tile so that the processing scales with the display as tiles are added. As with all display processors, memory access and locality are the foremost considerations affecting performance. In our case each module includes graphics memory along with the low level graphics processor. We propose a partitioning of display functions between the embedded back end and a more conventional front end that delivers high performance with modest additional bandwidth between the front end and the embedded processors.

Before examining the low level implementation of the embedded processor it is essential to establish the high level framework. Two factors, the modularity of a tiled display and the interface to DRAM, play the central role in determining the high level structure. First we recognize the necessity of locality for communications and storage within the processor. Second, we dispense with the notion of random access to DRAM. Because the processor is intimately connected to the refresh operation of the display we treat its embedded main memory as a unidirectional buffer.

Since the processor resides at the back end of the graphics pipeline it is primarily an engine for 2D graphics and imaging functions. While the general purpose programmable elements of 3D GPUs have been successfully employed for 2D graphics and image decoding, we seek a more direct implementation for these common features. The ideas may extend to more complex 3D applications, but we have not yet attempted them. With the processor placed between graphics memory and the display pixel stream, graphics rendering is synchronized with video refresh. In other words, video pixel coordinates are fed to the processor along with function specification and a pixel stream is emitted at the output. Unlike typical GPU methods, the processing is limited to a single pass and graphics representations must be implicit functions of screen coordinates.

*{jtw, kajiya, erikruf, raybit}@microsoft.com

2 Modularity, Scalability, Partitioning

Tiling remains the most economical means of assembling large area displays and tiling of display processors keeps communications between processor and display local. While communications among separate display processors is useful, it must be limited to something as simple as near neighbor links if operation of the embedded processors is to scale. For small scale experiments utilizing a single front end processor we have found it sufficient to use an even simpler common bus broadcasting all function parameters to all tiles.

As shown in figure 1, the embedded processor is placed in the refresh path downstream of graphics memory. A conventional general purpose processor (e.g. CPU/GPU) transmits data and addresses to DRAM main memory; a dispatching processor steers memory reads to specific processor elements; outputs of processor elements are composed into pixels. The simplicity of our partitioning scheme allows us to use FIFOs for all buffering of main memory accesses.

The front end graphics process sorts collections of functions into an array of bins (non-overlapping rectangular subregions of a tile) and forwards the array to the embedded DRAM. Note that each processor tile need only store the bins local to its region of the global display. The embedded processor is a simple pipelined array, programmed via configuration and parameters. As with general purpose dataflow machines, there is no instruction fetch overhead.

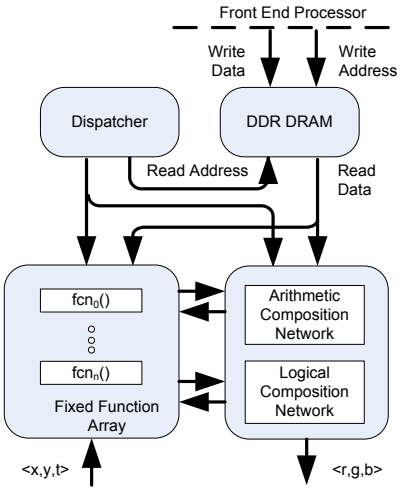

Figure 1: *Embedded function array and datapaths.*

3 Function Composition

Function composition is central to graphics and imaging in forms as complex as realistic shaders or geometry definition or as simple as linear interpolation for pixel compositing. Designing the functions themselves is a subject of continuing research, but the graphics literature is full of references to rich, reusable functions such as linear expression evaluators, quadratic expression evaluators, pixel composition functions, etc. The final stage of decoding compressed images via sum of products is a simple example. Our current function repertoire includes all of these plus flexible arithmetic and logical interfaces for composition. In all these cases function composition lends itself easily to parallel implementations in dataflow processors such as the one described here.

As an illustration, the function shown in figure 2 computes a signed distance function for the second order Bezier curves utilized in TrueType fonts as described in [Loop and Blinn 2005]. While the first two transformation stages can be easily replaced by a single stage, there are situations in which the first transformation is constantly updated while the second is fixed. Keeping the transformations separate facilitates this. The third stage produces a signed distance function, but it is not normalized to screen coordinates. The normalization stage requires operations which would be slow and expensive if implemented directly. We utilize a bipartite approximation [Ercegovac and Lang 2004] containing small lookup tables and an adder. At the output of each stage intermediate results are stored in a pipeline register. Discounting the initial transformation, latency in the example shown is three clock stages (pixels) while the latency of simpler linear functions is only one cycle. Latency matching is achieved by offsetting the x input. Many other examples are simpler, but this one illustrates most of the elements of an individual graphics function.

As screen size increases the range of x and y coordinate values increases without bound. To restrict the growth of datapath widths, function coordinates must be translated to local tile coordinates. Since this places an additional load on the front end, an alternative for small arrays is to use 12 or 16 bit global coordinates and translate local coordinates to the global space.

Screen coordinates of each pixel are updated at the video rate, but other parameters may change much more slowly. For example, three linear expression functions can provide a gradient screen background with function coefficient updated once per frame. As we shall see with text examples, binning of parameters dramatically reduces the size of a function tree. Time is also an input to function units, but we have not exploited it except to apply simple transformations to functions.

3.1 Configuring Composition

As seen in the example of figure 2 composition may be as simple as cascading stages of an integrated function. Composing a set of such functions to render a font, for example, can be as simple as ANDing the sign bits of signed distance functions as in figure 3. Other specific examples include finding a minimum of several signed distance functions. This is most easily done in a tree of pairwise adder and multiplexer stages. One possibility is to embed a fixed tree with fixed connections to individual function units. Specific subsets of minima can be obtained by tapping the tree at the appropriate node. Collections of functions are defined simply by steering coefficients to neighboring function units. Since most shapes require a mix of functions, the array is overprovisioned with a mix of function types. This is a straightforward tradeoff between area devoted to function evaluation and area devoted to configuration switching.

In the absence of limits on circuit costs, arbitrarily complex composition can be accomplished with exhaustive (crossbar) connection. One goal of this project is to see how much flexibility and utility can be obtained from more constrained connectivity. A traditional reconfigurable array processor would embed all functions into an array overlaid with one or more flexible interconnection paths. What we have described so far is an array of specialized graphics functions connected to a second array of composition functions. Note that elements such as pairwise minimum adder/multiplexers are functions themselves. In fact, given that functions compose to generate a single output we have adopted these piecewise tree structures as building blocks of the datapaths.

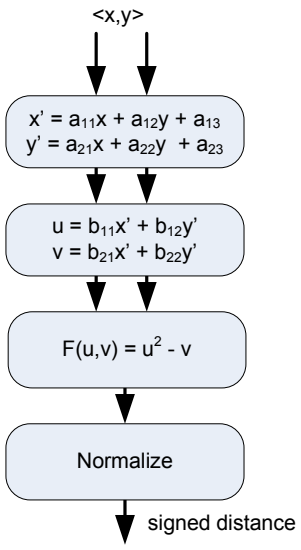

Figure 2: *Four stage signed distance function for second order Bezier curve.*

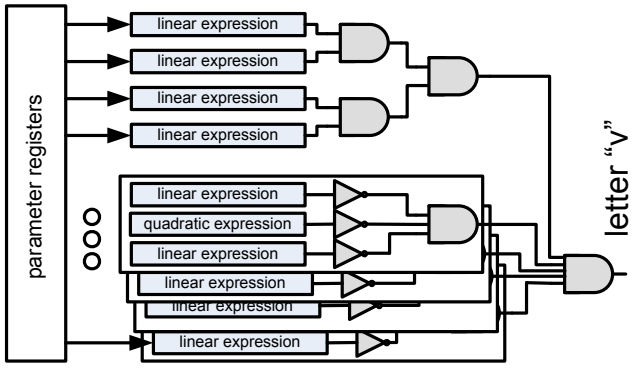

Figure 3: *Glyph constructed by logical composition of signed distance functions.*

A third class of functions includes simple logical operators. For example, determining whether a pixel is inside or outside any closed shape is determined by the AND of sign bits of the signed distance function. Since single bit logical signals comprise the data to be composed in this case, a rich logical composition network is relatively inexpensive.

What we have implemented for graphics computation, then, is a standalone array of parameterized fixed functions feeding a modestly flexible network of composition functions and a third richly reconfigurable array of logical functions. This heterogeneously configurable arrangement is dictated by both cost and performance

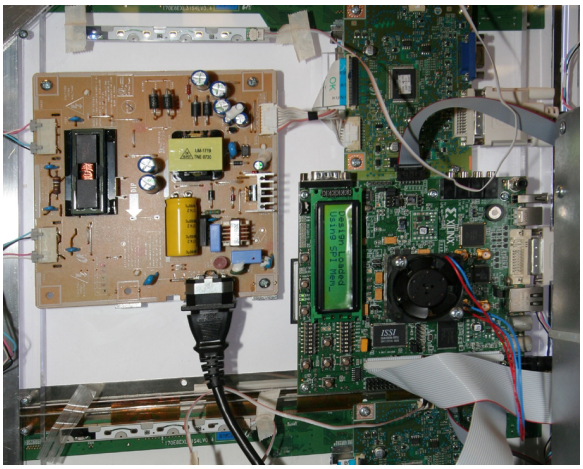

Figure 4: *Rear view of physical apparatus including one of the six LCD panels with power supply (left) and FPGA development kit (lower right) attached.*

considerations. Note that a conventional graphics processor employs generic functions whose inputs and outputs are addressable memory locations. Our scheme employs addressable functions connected via (mostly) fixed data paths and the experiments examine the utility and flexibility of this alternative arrangement.

4 Experimental Apparatus

The experimental apparatus is a desktop array of six tiled LCD panels, each driven by a Xilinx ML501 prototyping board containing a Virtex-5 field programmable gate array and 256MB of attached DDR2 DRAM. Even this modest number of modules forces us to address scalability.

For the sake of simplicity the function composition network produces a single real time pixel stream. At SXGA resolution of 1280x1024 the pixel clock runs at 120 MHz. The function units within the network are pipelined and utilize both rising and falling edges of the pixel clock. Implementing multiple parallel networks with lower clock rates and less pipelining is straightforward, but without interfacing to the LCD panel at the row (gate) and column (data) level, it is not possible to explore pixel level parallelism other than along a scan line.

FPGA programming is specified in Verilog using the Xilinx ISE tools which are provided with the development kits. Since the parallelism of the hardware is expressed directly in the hardware description language it is a natural choice for experimentation.

5 Performance

A limited collection of examples is shown in figure 5 including a bitmapped background, a PowerPoint overlay, and miscellaneous shapes. The background bitmapped images are sprites routed through the same function array as other objects, although arbitrary transformations of sprites (texture mapping) is not yet implemented.

Clocked at 140MHz the DRAM interface provides a peak throughput of 2.24 GB/s. In the worst case we assume that the images may be written from the front end and immediately read by the display at the frame rate, consuming 960 MB/s with raw pixel transfer. The remaining 1.28 GB/s is available for writing and reading function parameters and configurations as well as locally stored image data.

Since the embedded array runs at constant video rate, performance is measured purely in terms of image complexity versus memory bandwidth and circuit area. As shown in table 1, dividing the image into bins reduces the required number of function units at the added cost of higher bandwidth for memory writes from the front end processor. Since typical memory modules have become large and inexpensive, memory capacity is a secondary issue.

Content	Bin Size (pixels)	Maximum Number of Functions	Memory Input Bandwidth (MB/sec)
large text	unbinned	2088	3.4
	64x64	19	3.6
	32x32	11	5.3
	16x16	9	9.7
small text	unbinned	3449	5.6
	64x64	59	6.6
	32x32	26	7.9
	16x16	18	10.5
teapot line drawing	unbinned	2016	1.9
	64x64	16	4.6
	32x32	16	6.7
	16x16	16	11.9

Table 1: *Area and bandwidth costs versus bin size.*

While the examples above as well as those included in figure 5 are sparse, they demonstrate that moderate binning clearly offers a dramatic reduction in circuit area for an acceptable increase in memory bandwidth. Not shown in the table is the additional cost of front end sorting for smaller bins.

6 Related Work

Accelerating image computation with reconfigurable dataflow architectures is an old idea. Acosta et al [Acosta et al. 1995] describe a standalone reconfigurable image processor. Their approach is to identify specialized components and flexibly compose them through a switch. More recently GPUs have been programmed to perform image decoding operations [Han and Zhou 2006] .

Alternatively, Pixel Planes was devised as a special purpose rasterizer which was later discovered to have hidden flexibility [Fuchs et al. 1985]. Pixel Planes' core element was a parallel linear expression evaluator distributed throughout frame memory. In multiple passes the function evaluator was re-parameterized and composed sequentially with previous passes. The range of applications extended well beyond polygon rasterization. A thorough explanation of the use of signed distance functions for rasterization and other graphics uses can be found in [Frisken et al. 2000].

Our approach is a mix of these ideas in which a reconfigurable platform is used to implement a variety of specialized functions to better understand which ones support the broadest range of media applications. However, our implementation is markedly different in that the image generation circuitry produces its pixel stream directly from parameters stored in memory rather than accumulating pixels in a frame buffer. For image decoding operations the embedded processor employs a more direct implementation than those proposed for programmable shaders in GPUs.

In most instances, rendering for large displays continues to be limited to multiple projection systems coupled to multiple graphics computers [Li et al. 2000], a configuration which is acknowledged to be expensive [Kurtenbach and Fitzmaurice 2005]. Finally we

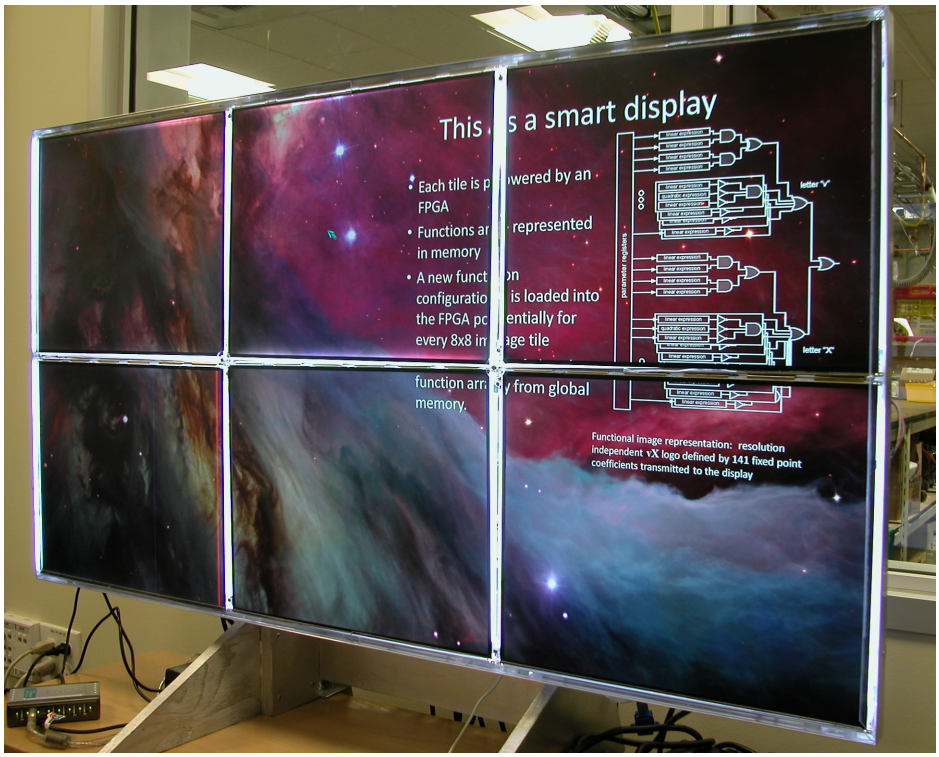

Figure 5: *Multipanel desktop display. (Worldwide Telescope image courtesy of Curtis Wong, Microsoft Research.)*

note that the notion of embedding a processor in the display is not novel [Myer and Sutherland 1968], but there has been little work specifically targeting circuitry for tiled flat panel displays.

7 Summary

A large, tiled display has been enhanced with the addition of simple function composition trees and a simple memory interface. The use of reconfigurable logic gives us great latitude to explore architectural tradeoffs, but the constraints of scalability and operation as an embedded graphics processor have led us to a simple design. Nevertheless, we have found the function composition array processor to be remarkably flexible as well as effective in moving a significant amount of common graphics and image processing to the back end of the display pipeline.

References

ACOSTA, F. K., BOVE, JR., V. M., WATLINGTON, J. A., AND YU, R. A. 1995. Reconfigurable processor for a data-flow video processing system. In *Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing, Proc. SPIE 2607*, SPIE – The International Society for Optical Engineering, Bellingham, WA, J. Schewel, Ed., 83–91.

ERCEGOVAC, M. D., AND LANG, T. 2004. *Digital Arithmetic*. Morgan Kaufmann.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Siggraph 2000, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, K. Akeley, Ed., 249–254.

FUCHS, H., GOLDFEATHER, J., HULTQUIST, J. P., SPACH, S., AUSTIN, J. D., FREDERICK P. BROOKS, J., EYLES, J. G., AND POULTON, J. 1985. Fast spheres, shadows, textures, transparencies, and imgage enhancements in pixel-planes. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 111–120.

HAN, B., AND ZHOU, B. 2006. Efficient video decoding on gpus by point based rendering. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, ACM, New York, NY, USA, 79–86.

KURTENBACH, G., AND FITZMAURICE, G. W. 2005. Guest editors' introduction: Applications of large displays. *IEEE Computer Graphics and Applications 25*, 4, 22–23.

LI, K., CHEN, H., CHEN, Y., CLARK, D. W., COOK, P. R., DAMIANAKIS, S. N., ESSL, G., FINKELSTEIN, A., FUNKHOUSER, T. A., HOUSEL, T. C., KLEIN, A., LIU, Z., PRAUN, E., SAMANTA, R., SHEDD, B., SINGH, J. P., TZANETAKIS, G., AND ZHENG, J. 2000. Building and using a scalable display wall system. *IEEE Computer Graphics and Applications 20*, 4, 29–37.

LOOP, C., AND BLINN, J. 2005. Resolution independent curve rendering using programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 1000–1009.

MYER, T. H., AND SUTHERLAND, I. E. 1968. On the design of display processors. *Commun. ACM 11*, 6, 410–414.

WATSON, B., AND LUEBKE, D. 2005. The ultimate display: Where will all the pixels come from? *Computer 38*, 8, 54–61.