***Abstract*—In highly interactive applications, low latency (the time between a user's action, and the response to this action) is critical for a good user experience. Traditional GPU architectures can make very low latencies difficult to achieve. This is because they are designed first and foremost to implement the painter's algorithm - a rendering algorithm that trades-off visual realism for moderate computational speed and high scene dynamism. The dataflow programming paradigm, along with dedicated toolchains such as Maxeler's MaxCompiler, enable the design of application-specific graphics accelerators. Such accelerators, however, have the advantage that their architecture can be completely customised. In this paper we present a custom renderer that composites 2D sprites and maps to emulate a graphical user interface. It was designed to facilitate user interaction tests described in our previous work. Our design is ultra low latency, updating what is being driven to the display within 1 ms of receiving user input. This is far lower than traditional GPUs. We describe the operation of our renderer, and our novel DVI display driver output stage. We measure a latency of under 1 ms for our renderer, with an end-to-end delay of 6 ms for our whole apparatus. We compare this with the end-to-end latency of the same apparatus built with a modern GPU, which we measure at 20 ms.**

Ultra Low Latency Dataflow Renderer

Sebastian Friston and Anthony Steed
University College London, Computer Science Department
Gower Street
London WC1E 6BT, UK
E-mail: sebastian.friston.12@ucl.ac.uk, a.steed@cs.ucl.ac.uk

Simon Tilbury and Georgi Gaydadjiev
Maxeler Technologies Ltd.
1 Down Place
London W6 9JH, UK
E-mail: s.tilbury@maxeler.com, georgi@maxeler.com

I. Introduction

Computer graphics algorithms create images by simulating the transport of light through an environment to the viewer. The algorithms that perform this simulation trade-off fidelity, computation time and scene dynamism, optimising for one or another depending on the application [1]. For real-time rendering, arguably the most popular algorithm is the painter's algorithm. In this algorithm primitives (triangles) that make up the geometry of a scene are projected onto a viewing plane. For each fragment (nominally a texel on the plane) the colour of the surface of the primitive under that location is computed. The colour is based on an approximation of the transport of light from illumination sources in the scene to the view plane [2]. This is computationally efficient. However until all primitives have been processed, the frame cannot be displayed as any portion of it could change in the future. There are applications for which these characteristics are suboptimal. Examples are highly interactive applications such as virtual reality or simulators.

Modern Graphics Processing Units (GPUs) are highly parallel stream processors; the ancestors of these cards implemented the painter's algorithm algorithm in fixed logic, and their architecture is optimised for this algorithm still [3], [2]. The implementation details of the algorithm are up to the application designer, with some fixed functionality and other reprogrammable parts known as shaders. The responsibilities of each type of shader however are similar across applications and GPUs - each type corresponding to a different stage of the painter's algorithm (e.g. the vertex shader projects triangles onto the viewing plane, the fragment shader estimates the colour). The shader programs are executed by sets of stream processors - typically Single Instruction Multiple Data (SIMD) cores, with the output of each stage feeding into the next. The final stage writes to a frame buffer. GPUs can also have dedicated hardware for operations such as depth culling or texture sampling [2]. The format of the data transported between stages is highly homogenous, and each stage will execute the same small set of operations on a large number of input tokens. This makes SIMD cores a highly efficient way to accelerate the algorithm. However, the requirement to build a frame from geometric primitives places a lower bound on latency tied to frame rate and geometry processing rate.

This presents a problem for those who require a rendering algorithm with a different set of optimisations. In interactive systems, low latency is important for a good user experience and optimal performance, as demonstrated in a number of studies (e.g. [4], [5], [6]). The typical way to conduct such studies, is to measure participants' performance at a primitive interaction task, while controlling the latency. Participants complete pointing and steering tasks using a specially designed interface, while metrics about their performance (e.g. task completion time) are measured under different conditions. This requires an apparatus with very low tunable latency. Studies continue to find the effects of latency at lower and lower levels, and these are beginning to reach the limits of the hardware typically used to conduct such tests.

For a recent experiment described in [7], we found ourselves in need of a renderer with very low latency. The architecture of traditional GPUs did not allow us to achieve the required latency. Instead, we designed a new renderer based on dataflow computing, and implemented it in reconfigurable hardware. FPGAs have been used for real-time rendering before, but with limited scope and tight vertical integration (e.g. [8], [9]). The dataflow computing paradigm and tools such as Maxeler's MaxCompiler are making it possible to construct image generators with entirely novel architectures, with similar complexity of programming as traditional GPUs [10].

To minimise latency, the time between the computation of any pixel and the latest input from the user must be minimised. With full control of the rendering architecture enabled by the use of reconfigurable hardware, we can ensure that the latency is deterministic. This allows us to 'race the beam'. Racing-the-beam involves computing pixel values at the rate of the pixel clock, with a latency much less than one frame period. The result is pixel values are computed and then transmitted directly to the display with minimal buffering, minimising the time between user input and the corresponding change in the display. The lower limit on latency is now due to the characteristics of the display, including the response time of the display elements.

In this paper we describe our dataflow renderer in detail. In Section II we review how reconfigurable hardware has contributed to image synthesis. Section III describes the design and operation of our renderer in detail. In Section IV we discuss the performance of our apparatus and compare it to the performance of the same apparatus, built with a GPU. Finally, we present our conclusions in Section V. We hope this to be the first step in building a high fidelity real-time renderer using dataflow computing.

II. Related Works

The abilities of dedicated hardware make it popular for a number of applications in computer graphics - including the problem of reducing apparent latency. For example, a

warper-board was designed to deform images, making them appear from a slightly different perspective than the one they were rendered from [6]. The warper board could mitigate the rendering delay, by updating the images with the latest tracking data before they were presented to the user in a head mounted display. This approach creates the illusion of low latency, but risks introducing artefacts if there is not enough data to re-compute the appearance of the image from the new viewpoint.

In addition to image processing, FPGAs have contributed to novel image synthesis, by performing simple but high volume computations. For example, Woop et al presented the design for a hardware accelerated ray-tracer prototyped on an FPGA [11]. In 2013, Caustics Professional released a ray-tracing accelerator card [12]. Yongsheng et al also describe a hardware ray-tracer, but built with dataflow computing. Like Woop et al, theirs performs both intersection tests and shading in hardware. They note that dataflow computers are suitable for ray-tracing due to the highly parallel nature of the algorithm [13].

Ten Hagen et al presented one of the first practical examples of an image generator utilising dataflow computing in their Dataflow Graphics Workstation [14]. The dataflow co-processor in their system did not drive a display directly, but performed pre-processing on 3D data, a function not dissimilar to modern day vertex or geometry shaders. Voitsechov & Etsion present an alternative architecture for GPGPUs based on dataflow computing [15]. In their architecture instructions from CUDA kernels are mapped to a dataflow graph. Their architecture supports multiple concurrent threads, using input token buffers at each node, which dynamically select which tokens to execute at any time based on available input parameters from the various tokens. This allows out-of-order execution at each node, maximising usage of the entire graph when some nodes (e.g. memory access nodes) have non-deterministic latencies.

FPGAs have also been applied to the specific problem of low latency image synthesis. Regan et al constructed a very low latency 3D light field renderer [8]. The architecture of their renderer is similar to ours, and their goal was also to design a renderer for investigating latency in human computer interaction. They achieved a latency of only 200μs, though the memory limitations of their platform permitted parallax in only one dimension. The closest example to ours is that of Ng et al [9], who built a low latency direct touch interaction interface. They also designed theirs for latency interaction experiments. In their apparatus, the touchscreen and projector were connected directly to the FPGA, where the experiment logic as well as the rendering algorithm were implemented. The PC running the experiment was connected via USB and was used to control the flow of the experiment. The apparatus achieved an end-to-end latency of 1 ms.

III. Design of a Low Latency Renderer

We designed an experiment, documented in [7], in which we asked participants to perform pointing and steering tasks. To conduct this experiment we required a renderer that could draw 2D images with a very low latency. Examples of these are in Figure 1.

A. Architecture

We constructed such a renderer using a Maxeler Dataflow Engine (DFE). This is a computing platform for executing

Fig. 1. Images of the stimuli the participants were exposed to.

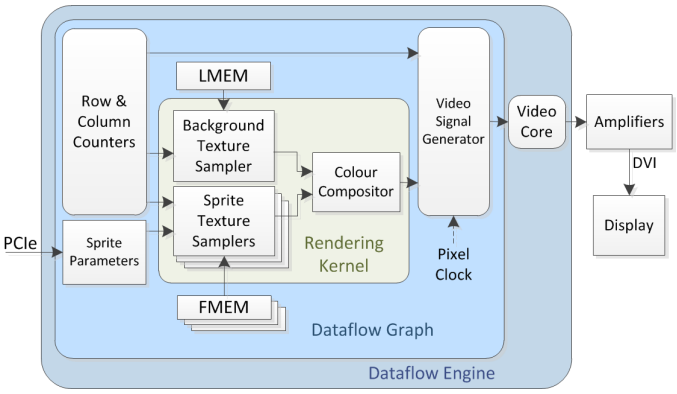

Fig. 2. High level architecture of our dataflow renderer.

dataflow graphs described in Maxeler's high level language MaxJ. The architecture of our renderer is shown in Figure 2. The renderer computes pixel values by combining a set of sprites (2D images) of different sizes and locations. The DFE has two types of memory, 48GB of DRAM (off-chip) and ~5MB of SRAM (on-chip). The smaller sprites are stored in SRAM and the larger background maps in DRAM. The pixel colour values are computed by the Rendering Kernel. The current location on the display is tracked by row and column counters outside the kernel. For each pixel the Rendering Kernel samples colour values from the memories and combines them using a set of functional blocks operating in parallel. The final colour values are transferred to the Video Signal Generator, which generates timing signals such as HSync, VSync and DE. The combined colour and timing data is transferred to the video core, which performs 8b/10b encoding, and then line level encoding and output from the DFE using high speed transceivers. The output is logically compliant DVI. A simple adapter board implements the physical interconnect. By synthesising the DVI signals & driving the display directly from the FPGA, we can ensure no additional latency is introduced and minimise the complexity of our apparatus.

Most functionality is implemented within the dataflow graph, in kernels running at ~200 MHz. The video signal generator runs at 152 MHz, the pixel clock rate of the 144 Hz display. MaxCompiler, Maxeler's toolchain, handles the transfer of data between kernels in different clock domains, as well as buffering and backpressure signalling between them. By controlling the size of the buffer between the Rendering Kernel and the Video Signal Generator Kernel, we can control how many pixels are rendered in advance. The buffer only has to be large enough to account for pauses in the data stream due to non-deterministic operations such as DRAM memory accesses. Pixel values are computed continuously. The only communication from the CPU is to update the algorithm parameters asynchronously.

The parameters include which background map to use, and the location and content of the sprites.

B. Image Composition

The background map is sampled using burst reads into multiple DRAM modules simultaneously, with the reads being concatenated into a single 3072 bit word. Each word contains pixels within a segment of a line. The aspect ratio of these words are changed to form a stream of pixels. At the start of each new line in the frame, commands to read the next line in full are issued to memory. At the same time individual pixels from the previous reads are read from the input buffer. The commands to sample the first line of a frame are sent during the synchronization period, when nothing is drawn. For line widths that are not multiples of the memory width, the remaining data is discarded during the synchronization period.

Sprites representing buttons and cursors were so small they could be stored in SRAM, which supports fast random access. The address to sample is computed based on the offset into the sprite of the current pixel being computed. This is a function of the sprites location. The location and content of the sprites are updated via PCIe streams. If the offset is outside the bounds of the sprite, the sample is set to transparent. In most modules colours are represented as 32 bit RGBA allowing alpha blending. The sprites are composited in a fixed order obscuring or blending with the background map or those below them based on their alpha component. The alpha channel is discarded when the final colour samples are transmitted to the Video Signal Generator.

C. Logical Display Interface

The colour and timing data is encoded into logically compliant DVI in the Video Core. DVI is a fully digital protocol which transmits 24 bit RGB data over three serial channels, with a fourth transmitting the pixel clock. The 8 bit colour words are converted to 10 bit words with a specialised version of 8b/10b encoding, ensuring each channel remains DC balanced [16]. During the blanking periods of the frame, the RGB data is substituted for control data, which includes the HSync and VSync signals.

The Data Enable (DE) signal from the Video Signal Generator determines whether the colour or control data (Hsync & Vsync - also from the Video Signal Generator) should be transmitted on a given clock cycle. The colour or control words are routed to an 8b/10b encoder and then into the serialiser of the high-speed transceivers on the FPGA of the DFE. The words pass through a shallow FIFO buffer used to transfer the words between the clock domains of the Video Signal Generator and the transceiver Physical Media Access (PMA), which run at the same rate but may be out of phase.

Maxeler's toolchain does not currently support direct access to the transceivers, however with Maxeler's assistance a small modification was made to the toolchain to give access to our dataflow graph. The Physical Coding Sublayers (PCS) within the transceiver blocks were configured to provide direct PMA access to the transceivers. The serialiser is present within the PMA of the transceivers, and was configured with a serialisation factor of 10. The transceivers were bonded together and driven by a single external serial clock, produced by a fractional-PLL placed within the transceiver bank. A fourth transceiver was used to transmit the pixel clock. To do this a constant pattern of 0x1Fh was written to the parallel data port. The driver output stage is shown in Figure 3.

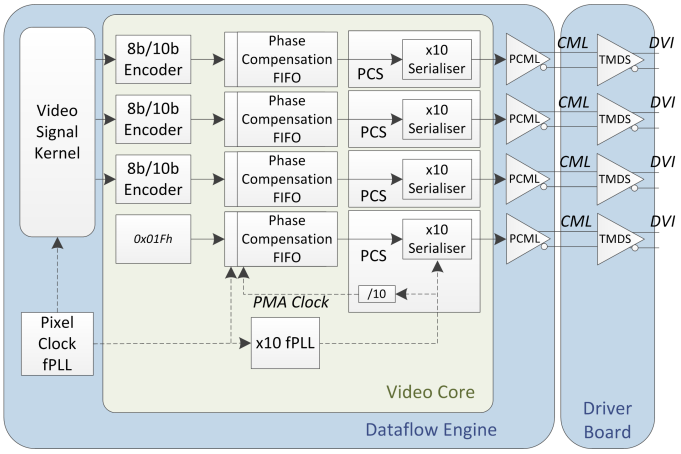

Fig. 3. Diagram of the data transfer and clocking of our DVI driver output stage, and at what level each component exists.

D. Physical Display Interface

The transceivers on the DFE use PCML (Pseudo Current Mode Logic) and are AC coupled. DVI specifies DC coupled CML, with the common-mode voltage set by the receiver. To make the output DVI compliant, a board was constructed which routed the serial data through a TI DS34RT5110. This is an HDMI re-timer IC which implements TMDS outputs. The board attaches to proprietary gold finger connectors on the DFE, and presents a standard DVI female connector. It's design and layout was based on the reference provided by Texas Instruments [17].

IV. Experimental Results

We measured the end-to-end latency of our apparatus at ~6 ms, using the cross-correlation variant of Steed's Method [18]. This was predominantly the scan out time of the display. In addition, we connected LEDs to a specially designed output of the DFE, and the parallel port of the host PC. On receipt of a specific input the CPU application would signal the parallel port and the DFE to illuminate these. High speed video (1000 fps) monitored the input devices and the LEDs. High speed video was used rather than an oscilloscope because it could monitor the input device and scan-out in addition to the LEDs, without additional instrumentation. The delay between each stage was determined by counting the number of frames between the occurrence of each event. The high speed video showed the first LED activating within ~1 ms of the input device being triggered. The delay between the first and second LED was so low it could not be discerned from the video. This is summarised in Figure 4.

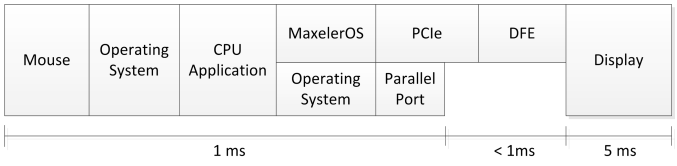

Fig. 4. Total observed delay of each stage of our apparatus.

A. Comparison with GPU

We rebuilt our apparatus with a GPU (an NVidia Quadro NVS 290) in place of the dataflow renderer in order to measure the latency a traditional GPU would provide. The system remained otherwise unchanged. A small program was written which drew three textured squares, one controlled by the mouse. The program used the GLUT toolkit to draw quad primitives specified directly in normalised device coordinates. The latency was measured under two conditions using the cross-correlation variant of Steed's Method. The results are shown in Table I. We used a swap chain of length 2. With no swap chain visual artefacts were so severe measurements could not be taken. We judged that the visual artefacts present with no VSync would not be too disruptive to run the experiment, therefore the best case for the GPU had a latency 3x higher than our renderer.

TABLE I. AVERAGE AND STANDARD DEVIATION OF THE LATENCY OF OUR STIMULI WHEN RENDERED WITH A GPU

Condition	Average (ms)	Std. Dev. (ms)	Sample Size
VSync On	26.17	2.79	6
VSync Off	19.86	4.16	7

V. CONCLUSION

Reconfigurable hardware has been used before for low latency image synthesis. These are typically low level implementations with tight vertical integration. For example the apparatus of both Regan et al and Ng et al had the tracker driven by the same device performing the rendering. Reconfigurable hardware combined with the dataflow programming model can make application specific rendering hardware cost effective. Our sprite renderer has comparable scope to both prior examples, but our dataflow graph can be adapted to other use cases with an effort comparable to GPU shader programming.

Dataflow computers are an ideal platform to create new renderers without the limits of the painter's algorithm. Our renderer avoided the buffering inherent in that algorithm, and ran asynchronously of the CPU. It's architecture allowed us to race the beam, minimising the delay between user input and what is being drawn to the screen. For the renderer itself, this was less than 1 ms.

Our implementation currently requires a modified toolchain to drive the display. An alternative would be to use a standard platform interface such as Ethernet to route display data to another device. So long as the chosen protocol included back-pressure functionality the design would remain conceptually the same, but with the Video Core and display driver implemented externally. The design would be more complex however, with far more buffering and therefore higher latency.

We are currently extending our design to render more complex 3D environments by building a ray-caster environment map renderer. In this renderer, a cube surrounds the user's viewpoint. For each pixel on the user's viewplane, a ray is cast and a texture map sampled based on the intersection point. Like our 2D renderer, it will be able to update the viewpoint while drawing the frame. An environment map will offer a convincing view of a 3D environment. The renderer will allow further user studies to be performed in immersive virtual reality.

REFERENCES

[1] M. Zwicker, M. Gross, and H. Pfister, "A Survey and Classification of Real Time Rendering Methods," Mitsubishi Research Laboratories, Tech. Rep., 2000.

[2] J. F. Hughes, A. van Dam, M. McGuire, D. F. Sklar, J. D. Foley, S. K. Feiner, and K. Akeley, *Computer Graphics Principles and Practice*, 3rd ed. Addison-Wesley, 2013.

[3] J. D. Owens, "Computer Graphics on a Stream Architecture," Ph.D. dissertation, Stanford University, 2002.

[4] I. S. MacKenzie and C. Ware, "Lag as a determinant of human performance in interactive systems," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York: ACM Press, 1993, pp. 488–493. [Online]. Available: http://portal.acm.org/citation.cfm?doid=169059.169431

[5] M. Slater, B. Lotto, M. M. Arnold, and M. V. Sanchez-Vives, "How we experience immersive virtual environments: the concept of presence and its measurement," *Anuario de Psicología*, vol. 40, no. 2, pp. 193–210, 2009.

[6] D. a. Vincenzi, J. E. Deaton, T. J. Buker, E. L. Blickensderfer, R. Pray, and B. Williams, "Mitigation of System Latency in Next Generation Helmet Mounted Display Systems," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 55, no. 1, Sep. 2011, pp. 2163–2167. [Online]. Available: http://pro.sagepub.com/lookup/doi/10.1177/1071181311551451

[7] S. Friston, P. Karlström, and A. Steed, "The Effects of Low Latency on Pointing and Steering Tasks," *Submitted 2014*, 2014.

[8] M. J. P. Regan, G. S. P. Miller, S. M. Rubin, and C. Kogelnik, "A real-time low-latency hardware light-field renderer," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, 1999, pp. 287–290. [Online]. Available: http://portal.acm.org/citation.cfm?doid=311535.311569

[9] A. Ng, J. Lepinski, D. Wigdor, S. Sanders, and P. Dietz, "Designing for low-latency direct-touch input," *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*, p. 453, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2380116.2380174

[10] O. Pell and V. Averbukh, "Maximum Performance Computing with Dataflow Engines," *Computing in Science & Engineering*, vol. 14, no. 4, pp. 98–103, Jul. 2012. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-4614-1791-0_25

[11] S. Woop, J. Schmittler, and P. Slusallek, "RPU: a programmable ray processing unit for realtime ray tracing," *ACM Transactions on Graphics*, vol. 1, no. 212, pp. 434–444, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1073211

[12] J. Hruska, "The future of ray tracing, reviewed: Caustic's r2500 accelerator finally moves us towards real-time ray tracing," http://www.extremetech.com/extreme/161074-the-future-of-ray-tracing-reviewed-caustics-r2500-accelerator-finally-moves-us-towards-real-time-ray-tracing, 2013.

[13] L. Yongsheng, S. L. Chen, Z. Wenhao, L. Xiaojun, X. Tao, Z. Limin, Y. Shiming, and C. Jingfeng, "The design methodology of a High-Performance dataflow supercomputer on a reconfigurable chipset for use in 3D graphics applications," in *2014 12th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2014, pp. 1–3. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7021400

[14] P. Ten Hagen, I. Herman, and J. R. G. De Vries, "A dataflow graphics workstation," *Computers & Graphics*, vol. 14, no. 1, pp. 83–93, 1990.

[15] D. Voitsechov and Y. Etsion, "Single-graph multiple flows: Energy efficient design alternative for GPGPUs," *Proceedings - International Symposium on Computer Architecture*, pp. 205–216, 2014.

[16] Digital Display Working Group, "Digital Visual Interface Specification," no. April, 1999.

[17] Texas Instruments, *DS34RT5110-EVKH HDMI Extender Demo Kit for HDMI Cables User's Guide*, 2013, no. SNLU033A.

[18] S. Friston and A. Steed, "Measuring Latency in Virtual Environments," *IEEE Transactions on Visualization and Computer Graphics (Proceedings Virtual Reality 2014)*, vol. 20, no. 4, 2014.