Experience with a Difference Engine for Graphics

A.A.M. Kuijk, E.H. Blake*, and E.H. Steffens†
CWI, Department of Interactive Systems
Kruislaan 413,
1098 SJ Amsterdam, The Netherlands.
Email: Fons.Kuijk@cwi.nl

Abstract

The prototype of a novel raster graphics architecture has now become operational. The display hardware can be regarded as a very fast difference engine that works in two-dimensions. The speed is partly achieved by the use of custom VLSI components for the lowest level primitive operations and this permits the video rate reconstruction of images and other signals compressed by encoding them on various polynomial bases. A novel feature of the architecture is that it avoids the use of a frame buffer.

The paper describes our experience with the new hardware in terms of positive and negative performance aspects. We discuss the architecture and its operating parameters. Another part of the paper evaluates our experience of hardware development in an academic setting. We believe there are significant lessons here for graphics researchers who might want to develop their own systems.

1 Introduction

A radical reappraisal of the three-dimensional (3-D) interactive raster graphics pipeline has resulted in an experimental architecture for a graphics workstation which is being evaluated at the CWI. A working prototype has now been built. This paper reviews the design and presents our first experiences.

The principal features of the design for the new raster graphics architecture are:

1. Emphasis on real-time interactive shaded 3-D graphics.
2. Object space methods rather than image space methods are used where possible.
3. Avoid the use of a frame buffer.
4. Use of custom VLSI only at the lowest, most primitive, levels where commercial products are unlikely to suffice in the near term.

A working prototype system that has been built provides further insight into the operation of the heart of the system: the custom VLSI difference engine. Previous papers over the past four years have traced the completion of the design specification [4], the implementation of critical components [3] and detailed simulation of functions [2]. A novel use of parts of the hardware that was not foreseen when the research project was initiated has been its use for decompression of images encoded on a spline wavelet basis [6].

The next section is an overview of the architecture for producing real-time interactive raster graphics. Section 3 details the design and implementation of the prototype system. The next section (4) presents the first results of using the prototype system. Finally a conclusion tries to draw out some of the lessons learned.

2 Overview of the Raster Graphics Architecture

A number of different feedback levels in the image synthesis pipeline can be identified if one takes a new look at the basics of high quality three-dimensional (3-D) raster graphics [7]. From the highest to the lowest level these include:

*Guest researcher at the CWI from the Department of Computer Science, University of Cape Town, Rondebosch, 7700, South Africa

†Electronics design engineer, University of Amsterdam, Faculty of Mathematics and Computer Science, Kruislaan 403, 1098 SJ Amsterdam

- Modelling
- Coordinate Transformations
- Viewing Transformation
- Hidden Surface Removal
- Area Primitive Processing

Visual effects can only be achieved by interaction with the data at each of these levels. A user interacts exclusively with the visible parts of a 3-D model, but not with pixels since they are too primitive a kind of object to be of interest to a user. We provided direct access to graphics objects to support pointing and identification. These actions are considered fundamental because they underlie every change a user makes in a picture.

Changing pictures form the key to the architecture. Actual pixels are not needed for user interaction. If we take this observation seriously and ruthlessly pare away other elements we get a radical prescription for a graphics architecture. One where the visible surfaces of objects are explicitly identified, and without any mandate for a frame buffer. We have thus built a machine that hearkens back to the calligraphic roots of graphics displays and at its heart is a processor which echoes the original Difference Engine of Babbage.

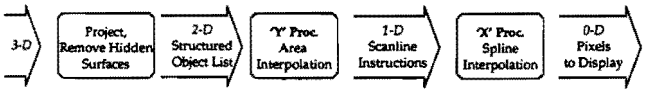

Figure 1: Functional Elements of Display Architecture. The Y and X processors together form the display controller. The arrows show how 3-D data are converted to a stream of pixels. Data structures are indicated by arrows and of these only the 3-D Models and the 2-D Structured Object List are stored. All other data are generated on the fly. The Structured Object List is updated at the animation or interaction rate, 12–24 times per second, and it is read at the video frame rate, 50–60 cycles. The Scanline Instructions are produced at the video line rate and clocked into the X processor at the video pixel generation rate.

The functional elements of the display architecture are shown in Figure 1. The major new components follow from the *bottom up* (right to left in Figure 1):

1. pixel generator (Difference Engine or X processor — Section 2.1).
2. area processor (Y processor — Section 2.2).
3. hidden surface removal, projection and other higher level functions, that are not discussed here at all.

Items 1 and 2 together form the display controller shown in Figure 2. In our architecture the frame buffer is replaced by a structured list of objects which can be *pointed at* (at the video frame rate of 50–60 cycles). At this lowest level of the architecture custom components are needed: these have already been built. The display controller — a powerful VLSI-based hardware block containing a systolic array of processors — produces a full colour pixel stream at video refresh rate. This block is fed with instructions produced by general purpose microprocessors from the structured object list. The processors are capable of producing Phong shaded 3-D objects or 2-D textures at this video frame rate. For full colour images one systolic array is needed for each of the primary colours (red, green and blue).

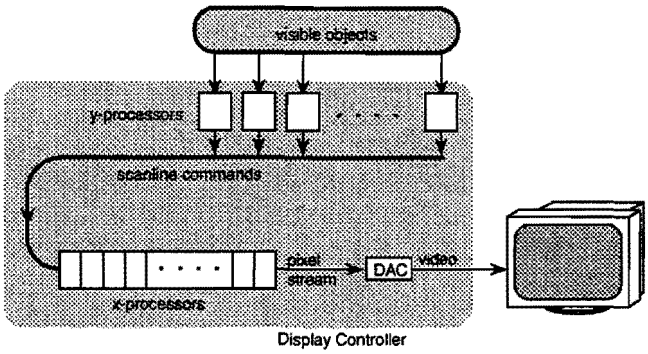

Figure 2: Block Diagram of the Display Controller

The structured object list contains objects that describe small 2-D surface primitives — things like triangle meshes, trapezoids or splines outlining characters in various typefaces. The objects have all the information necessary to render them on the display. The size of the object list does depend on the complexity of the image. The size of the list is about the same as a conventional frame buffer, with the advantage that the object list is resolution independent so that the list will not become larger for higher resolutions. The list is also a much more organized data structure and can therefore support more sophisticated operations than a simple frame buffer. It has to be updated — or produced — at a rate sufficient for animation or interaction, i.e., 12–24 times per second.

At higher levels of the architecture the objects become more complex (but also fewer — less fragmented) to maintain information about light sources, textures and viewing (hidden surface removal or overlay priorities etc.). Here representations for incremental changes typical for real-time interaction are favoured. The hardware requirements for the operations on this level appear to be satisfiable in the short term by powerful but "off-the-shelf" parallel hardware.

2.1 The Difference Engine

The processor referred to above as the pixel processor or X processor (because it deals with the scanlines of an image) is actually a difference engine. That is, it can do forward difference calculations at high speed. Any order of forward difference can be done, the limiting factor is the accumulation of errors during addition and (in real-time applications) the extra cycle time taken by each higher order. The architecture is tailored to second order forward differences.

The processor is a systolic array with a dedicated processor for each pixel in a scanline. Additions (and input) are done to a precision of 36 bits and the top 12 bits are output. So we basically have data values of 12 bits, say 10 bits magnitude and 1 sign bit and 1 bit to detect common overflow situations, and 24 precision bits. For quadratic interpolation (second order forward differences) spans of 2^{12} bits — i.e., 4096 pixels wide — can be interpolated before the error becomes noticeable. This is also the maximum addressable pixel and span width allowed for input. If we use just 9 data bits of the output (i.e., 8 bit values + sign bit) then cubic interpolation for spans of 512 pixels can be done accurately. Naturally longer spans can always be done by splitting them into shorter ones which can be done correctly.

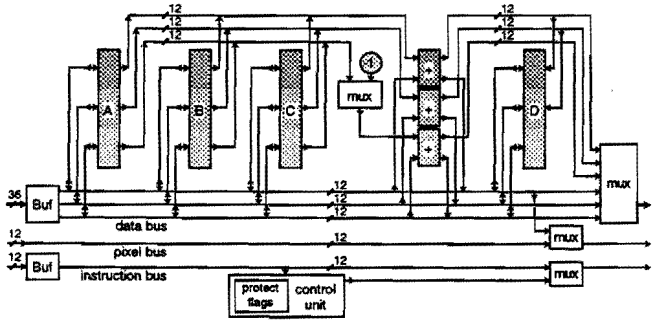

Figure 3: Register level architecture of the X processor. The arrows indicate the directions for read/write access. To allow for pipelined block carry addition, the data bus and the 36 bit registers are subdivided in three 12 bit sections, in the figure indicated by different shades of grey.

The bias towards quadratic interpolation is also seen in the fact that there are 3 internal registers with which the value I ("intensity"), first difference δI and second difference $\delta\delta I$ can be set at every pixel location (see Figure 3). The operation of the difference engine is illustrated in Figure 4. The interpolated intensities are calculated in register A and the contents of A is added to an output accumulator (D) which can contain the results of a number of previous interpolation spans.

The 12 most significant bits of the content of the accumulator are read out (for display) on receipt of a "refresh" instruction. Other instructions are listed in Table 1.

Higher order differences are done by reusing the registers. In fact, since the result of each step of the forward differencing is communicated to the neighbouring processor, the registers are not needed to perform forward differences but only so that differences may be set ahead of time by means of the 'set' instructions (see Table 1). It is a way of changing just certain (usually the highest) differences within an interpolation that is continuous in the lower orders.

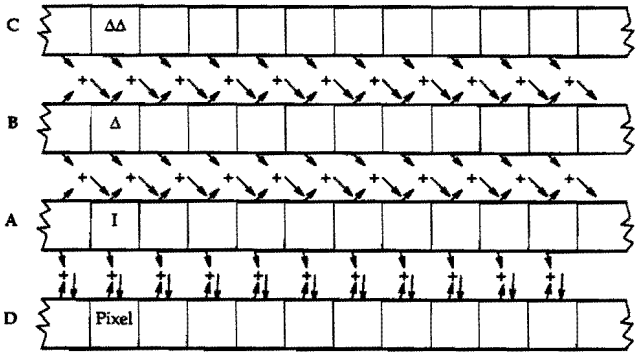

Figure 4: Interpolation on the Difference Engine. The difference engine is implemented as a systolic array of processors: one for each output pixel. For quadratic interpolation the second differences remain constant and are propagated to all processors within an active span of pixels (Register C). The first differences (Register B) are changed by the second differences at each step and the results added to the intensity values (Register A). The results of the interpolation step are added to an accumulator (Register D) so that multiple interpolation spans may overlap to produce the final pixel value. For higher order differences the same registers are reused.

To conclude: this X processor, which was developed as a very specialized pixel generator is really very general; it is a *Difference Engine* with:

- Any order forward differences.
- 36-bit numerical accuracy and an 11ns cycle time.
- Any spline interpolation, with constant cost independent of span length.

2.2 'Y' Processor

The information to drive the area or Y processor is contained in the Structured Object List (see Figure 1). It contains the visible areas of the displayed objects and their colouring information. This processor has the basic task of producing the instructions for the Difference Engine.

operation	description	cycles
acc_mode	Accumulate mode: if enabled negative intensities are not added to accumulator	1
dis(x,dx)	disable accumulation of intensities from pixel 'x' for 'dx' pixels (cleared after next 'eval*' command)	1
eval0(x,dx,i)	Set pixel (i.e., accumulator) from 'x' for span of 'dx' directly, disable further additions until next refresh.	1
eval1(x,dx,i)	add i to accumulator for span from 'x' for 'dx' pixels	3[a]
eval2(x,dx,i,di)	First order forward difference, starting at pixel 'x' with value 'i' and increment 'di', for 'dx' pixels	4[a]
eval3(x,dx,i,di,ddi)	Second order forward difference — like 'eval2' except now 'di' is also changed by 'ddi' at each step	5[a]
eval*n*(x,dx,i,di,...)	Higher, $n-1$, order forward differences, like 'eval3' *mutatis mutandis*	$n+2$[a]
nop	No operation	1
refresh	Output accumulator value and clear everything	2
setddi(x,dx)	Set (i.e., override) second difference[b] at points 'x', for a span of 'dx' pixels in the middle of the next 'eval' command	2
setdi(x,dx,i)	Like 'setddi' only it affects the lower forward difference	2
seti(x,dx,i)	Like 'setdi' except that this creates a span of intensities	2
setpddi(x,dx)	Set (i.e., override) second difference[b] at points 'x', 'x+dx', 'x+2dx', ... in the middle of the next 'eval' command	2
setpdi(x,dx,i)	Like 'setddi' only it affects the lower forward difference	2
setpi(x,dx,i)	Like 'setdi' except that this creates a pattern of intensities	2

Table 1: *X* Processor Instructions and Their Costs in Cycles. Or identically, the number of "atomic" instructions that make up the above "macro" instructions, see Section 3.1.1. **Note:** The costs mentioned above are incurred whether a span is 1 pixel long or covers the whole width of the scanline.

[a] The cost of this operation can be reduced by 1 cycle in future versions

[b] If there are higher order differences then this sets the highest order difference

The *Y* processors operate at the frame refresh rate and go through a complete cycle once every video frame. Their input data are produced at the interaction or animation rate (between 12 and 24 cycles per second). The output goes to the Difference Engine operating at the line refresh rate.

The task of these *Y* processors can also be described as having to change 2-D display information into 1-D scanline information. The third dimension has been dealt with at an earlier stage by projection and hidden surface removal. The geometry can be specified solely in terms of (2-D) display coordinates but (3-D) world coordinates are still needed for the vectors which underlie the shading calculations.

The edges of the surface primitives (triangles or trapezoids) are simple enough to find. Shading (in particular Phong shading) and anti-aliasing are more of a challenge. We have to recast the Phong shading model, but without lapsing into expensive Gouraud shaders or being unable to deal with all practical situations. Phong shading itself is nothing more (or less) than a very good practical approximation — it is not an end in itself. In [5] we introduced a method for quadratic Phong shading via angular interpolation. It has the lowest per pixel cost in time and storage of any method we are aware of.

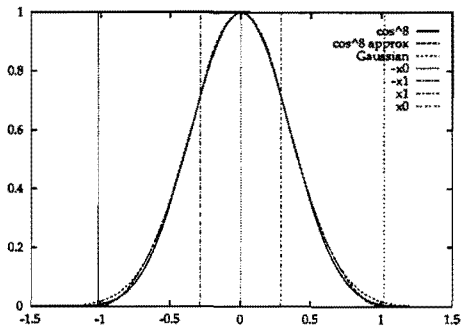

Figure 5: Quadratic Approximation of a Cosine to a Power Used in for Phong Shading
The diagram shows the curve of $\cos^n$ and the quadratic spline approximation of the same function, here $n = 8$ but the approximation is valid for wide range of n. The knot points of the splines are at $-x_0$, $-x_1$, x_1 and x_0. A Gaussian curve (i.e., a function of the form e^{-x^2}) is shown which has the same value at the points $-x_1$, 0 and $+x_1$ as the other curves.

Our approach to quadratic Phong shading depends on two major results:

1. A parameterized piecewise quadratic expression for $\cos^n \theta$ — the cosine of an angle $\theta, -\pi/2 < \theta < \pi/2$, raised to a power $n, 1 \leq n \leq 125$. The shape of the curve is very similar to a Gaussian (Figure 5).

2. A linear expression in terms of the pixel position, x, on a scanline for the angle θ between the interpolated normal vector of the surface and the light or highlight vector.

Anti-aliasing in the form of exact area integration is applied. Object space hidden-surface removal preserves the necessary information on pixel coverage. Where pixel coverage gradually increases along a scanline this

is treated as a linear modulation of the quadratic shading function. This results in a cubic expression which is passed on to the pixel generator.

The instructions generated by the area processor are thus expressions that describe various splines that are to be interpolated. An area primitive has been coded in terms of a spline basis. It is possible to encode the shading functions of whole trapezoids as 2-D quadratic functions.

3 The Prototype Difference Engine System

To demonstrate the feasibility of this new approach we have built a prototype system. We implemented the Difference Engine in full custom VLSI. This is the technologically most challenging part of the architecture. The other components were implemented with standard components, necessitating some compromises with overall system performance. In the prototype the instructions for the Difference Engine are stored in a buffer, rather than being generated in real-time by Y processors. As illustrated in Figure 6, the prototype system comprises the following three sub-systems:

- a SUN 4/100 Workstation,
- an INMOS B408/B409 Graphics system,
- the Difference Engine.

The SUN workstation serves as a host computer on which program development is done, data is stored and via which the graphics display system communicates with the rest of the world.

The INMOS Graphics system consists of two transputer based modules (TRAMS), the frame buffer IMS B408 and the display card IMS B409 (see Figures 7 and 8). In the frame buffer of the IMS B408 the instructions to be executed by the Difference Engine are stored in an encoded format. This instruction store is accessible via a T800 transputer which can either receive instructions from the SUN workstation or generate them locally. Data from the host is transferred via a transputer link. The instructions in the instruction store ("frame buffer") are decoded and send to the first processor of the Difference Engine from where it is processed by the processor array. At the end of the array, the pixels generated on the basis of the stored instructions are presented to the IMS B409 display board and displayed on a monitor.

The INMOS modules were selected to drive the Difference Engine because they met our needs with respect to functionality. Unfortunately they did not meet our requirements with respect to performance. The maximum throughput rate of the 32-bit pixel-bus interconnecting

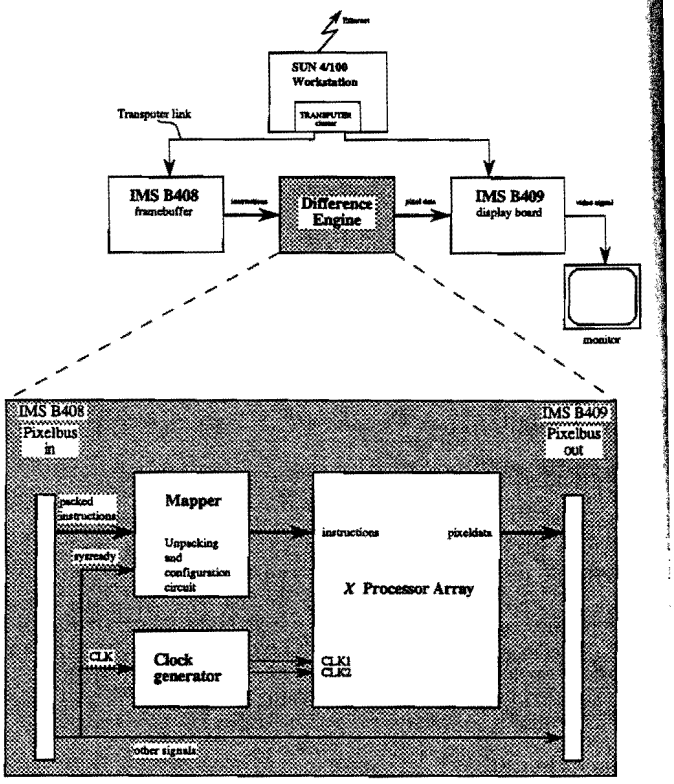

Figure 6: Block diagram of the prototype Difference Engine System. A frame store module is used to store instructions for the Difference Engine. These (encoded) instructions are put on the interrupted pixel bus, decoded by the Mapper and send to the X processor array of the Difference Engine, which produces the pixels that are inserted on the other side of the interrupted pixel bus, ready to be displayed.

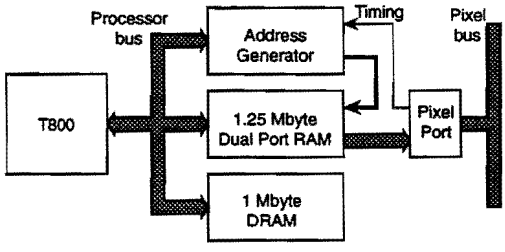

Figure 7: Block diagram of the IMS B408 Frame Store Module.

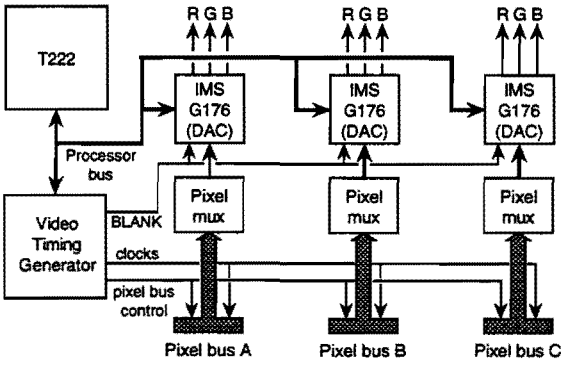

Figure 8: Block diagram of the IMS B409 Display Module.

the B408 and B409 is 25 Mhz. This is below the potential 48-bit 86 Mhz input capacity of the Difference Engine. The Display Module only has 6 bit DAC's, whereas the output pixels are 12-bits deep (per colour).

These compromises did not affect the usefulness of the prototype for demonstrating the feasibility of the architecture. They enabled low cost hardware testing without the delay of building special modules.

3.1 Board Level Implementation of the Difference Engine

The pixel bus between the INMOS modules is 32 bit wide, while the Difference Engine needs a 9 bits instruction and a 36 bits data word. This requires a mapping scheme between the pixel bus and the first X processor to expand the width. The X processor has 25 "atomic" instructions that make up the 14 instructions indicated in Table 1 (including the NOP instruction), so we can include a look up table between the pixelbus and the X processor and use a 5-bit index to refer to the appropriate 12 bit atomic instruction. This leaves us with 27 bits available for the data word. This 27 bit word has to be mapped onto the 36 bits of the two different data formats of the Difference Engine. One of the two types of data formats is a 36 bit intensity value, the other type is a 12 bit pixel address, a 12 bit pixel-span value and a 12 bit intensity value (used by the Eval0 instruction only).

In this prototype system, both types of instructions can do with 7+10+10 bits for the operand, exactly what is available if the 12 bits for the instructions are produced by means of a look up table.

In Figure 9 it is illustrated how the data from the instruction store is unpacked before it is send to the Difference Engine. Extension of the 27 bits stored in the frame buffer for intensity values to the 36 bit intensity value required by the Difference Engine is achieved by padding the four lower bits to zero, and extending the sign bit to the upper 5 bits. Extension of the 10 bit pixel address and 10 bit pixel-span value stored in the frame buffer to a 12 bit pixel address and a 12 bit pixel-span value is achieved by padding the two upper bits of both parts with zero. Of the remaining 7 bits, the sign bit is extended to the upper 5 bits to result in the 12 bit intensity value needed for the Eval0 instruction.

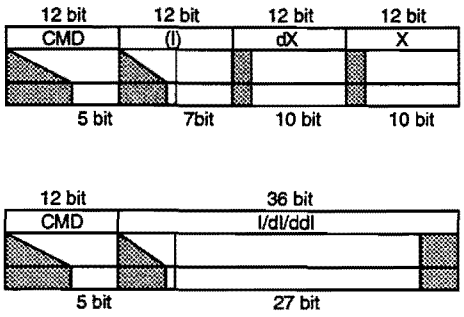

Figure 9: Expansion for instructions that specify X and dX (top) and instructions that specify an intensity (bottom).

In the following subsections we will discuss the circuitry we needed to add to the B408/B409 system in order to be able to drive the Difference Engine.

3.1.1 Input Circuit

The elements of the input circuit are shown in a block diagram (Figure 10). The input circuit consists of an instruction look up table, a data mapper, a delay circuit and a configuration circuit.

The instruction look-up table is build with one pal, the GAL P26CV12. The look-up table has one extra output signal, Type, indicating the type of the instruction. This signal is used in the mapper circuit to be able to handle the two different types of data formats appropriately.

The data mapper is build with three Lattice GAL 26CV12 pal's. The input to the mapper circuit are Type and 27 bits from the pixel bus which are mapped to 36 bits data word as described above. For each of the three 12 bit parts of the data port one pal is used.

Following the data mapper we find an input delay circuit. This delay circuit serves to delay certain bits of each of the three 12 bit sections by one or two clock cycles. By skewing of the 12 bit data words, the 12 bit adders can be pipelined which allows them to operate at the required maximum speed of 11 ns [3]. The delay circuit is build with GAL P26CV12 pal's.

A fully configured Difference Engine prototype consists of 80 IC's housing 9 processors each. With this

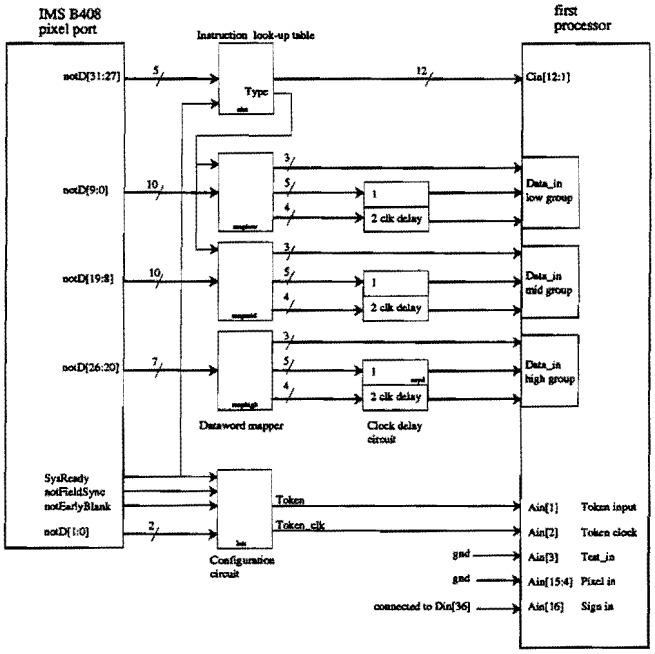

Figure 10: Block diagram of the input circuit.

number of processors it is not unlikely that some of the chips may have processors that are defective but are still capable of passing on the data. By switching off defective processors, we can still make use of these "somewhat" defective chips.

To switch off defective processors we can activate a configuration circuit that is then driven by the data put on the pixel bus.

3.1.2 Clock Circuit

The Difference Engine is clocked by a four-phase clock (two phase pulses and their inverse). In Figure 11 the timing of these clock pulses is shown. At the time of design of the test system, little was known about the exact parameters of the timing. A mere indication of the required pulse widths and intervals were given as result of some tests and analysis of the chip design. These tests already made clear that timing was critical and that the values given were too uncertain to design a fixed phase clock circuit. The appropriate clock timing could therefore only be established by tuning the clock parameters in a populated (or even partly populated) prototype. As a consequence of this we needed a clock circuit of which the width and phases are fully programmable.

Such a circuit has been build using Analog Devices AD9500 programmable delay generators (PDG). As is shown in Figure 12, two such programmable delay generators drive one D-flip-flop (a 10135 JK). One PDG will set the output of the flip-flop after a programmed delay of t_1, the second PDG will reset the flip-flop after a programmed delay of t_2. This creates a programmed

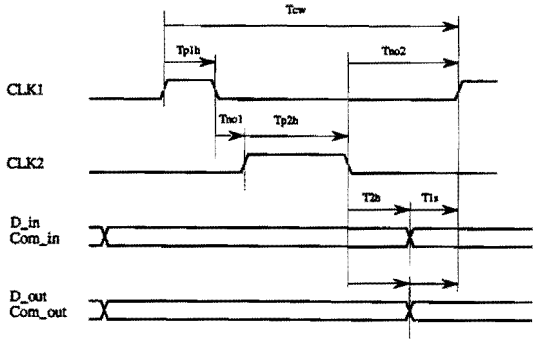

Figure 11: Timing diagram of the two phase processor clock.

pulse width of $t_2 - t_1$. These two PDG's are triggered by means of a PDG which in turn is triggered by the clock signal of the B408/B409 pixelbus. In this way we can shift the clock pulse thus obtained relative to the clock of the pixelbus. By using two of these circuits, we produce the fully programmable, two phase clock. Inverse phases are obtained by using the inverse output of the two D-flip-flops.

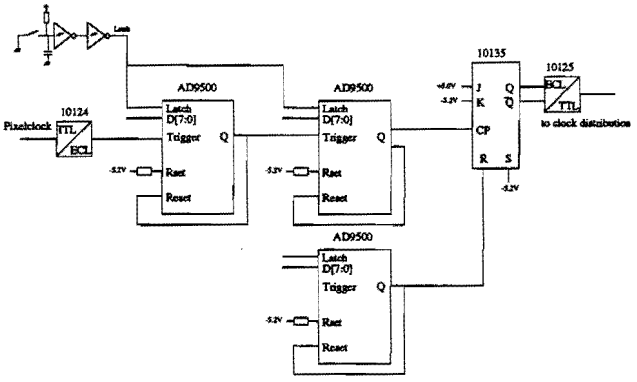

Figure 12: Block diagram of one of the two identical programmable pulse width clock generators.

The delay of the AD9500 is set by an 8-bit value, which is provided by two hexadecimal rotary switches, and latched manually by pressing a switch. The resolution of the delay can be set between 10 ps and 10 μs by means of an external resistor and capacitor. With Rset being 150 Ω and Cset not connected we obtain a maxim delay of 76.8 ns with a resolution of 0.3 ns.

The TRIGGER and RESET of the PDG's expect differential ECL signal levels, so that TTL/ECL conversion and visa versa is necessary. The delay setting however is TTL compatible.

The prototype system is implemented on multi-layer printed circuit boards. Two of the layers, serve for distribution of the power (a +5V and a ground plane). The clock circuit required a separated ground plane and

+5V, -5V plane. The AD9500 programmable delay generator is an analogue device that turned out to be rather sensitive to even small variations in the power as caused by switching of other digital components. Therefore, we had to make use of a separate power supply to avoid unstable operation of the clock generator.

3.1.3 Output Circuit

In 3.1.1 we indicated that a delay circuit was needed to skew certain bits of the input data with one and two clock cycles. As a result we also need an output circuit to unskew the pixel values that leave the last processor of the processor array (see Figure 13). The delay circuit that does this is build using 74AS574 registers. The output is buffered by 74AS642 buffers and terminated conform the pixel bus specifications.

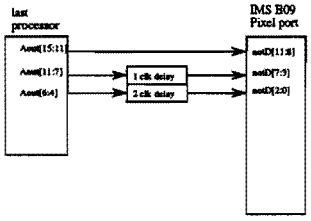

Figure 13: Block diagram of the output delay circuit.

3.2 Physical Design

The logical design of the system looks deceptively simple. The main difficulties with the hardware implementation arise simply from the size of the overall system: a system which houses a pipeline of 720 processors. The size of the system is determined primarily by the chip carrier in which the X processors are housed. This chip carrier is a 144 pin, 15 × 15, pin grid array (PGA). Building the pipeline with PGA's mounted flat on a printed circuit board, would imply that the pipeline would have to be spread out over several boards. In a normal bus based system such as the VME bus, this does not have to be a problem. However in this system, each X processor is synchronously clocked. Building the pipeline as a bussed system would make the clock lines unworkably long. On the other hand, providing each board with its own clock circuit would make synchronization between the boards too complex. Hence, it is important to reduce the distance covered by the processor pipeline and thereby to shorten the length of the clock lines as much as possible.

3.2.1 Board Layout

A workable solution was found by placing two X processor IC's on a small multi-layered printed circuit board. A motherboard contains 2 × 20 slots to house maximally 40 of these two-chip processor boards. This motherboard also houses the clock generator, the data mapper, the configuration circuit, and the in- and output delay circuits. The layout of this multi-layered printed circuit board is shown in Figure 14. Two connectors that connect to both ends of the interrupted pixel bus are situated on the front edge of the board, while a power connector is placed at the rear edge.

The length of the pipeline is minimized by using connectors with four rows of pins. Two adjacent rows are used for the input section, while the other two rows are used for the output section. By choosing each input pin opposite to the position of the related output pin, the pins of neighbouring connectors can be connected via a straight line so that no board space is used up by detours or crossing of lines. The connectors for the processor boards are placed as close as possible to each other. The limiting factor is the height of the PGA socket used on the processor boards. The space needed for routing of the interconnections of the X processors has been kept entirely on the two-chip processor boards.

A further reduction of the extent of the processor pipeline has been obtained by breaking up the pipeline in two sections as is shown in Figure 14. The clock lines can be found in the space between these two sections. This layout reduces the length of the clock lines — and thereby the propagation delay — by half. The length of the four clock lines is 273 mm each, which, given the clock rate, can be considered as being rather long. Driving wires of this length at 25 Mhz does not have to be a problem as long as the usual termination techniques are applied.

The setup of the system makes it possible to bypass empty slots and run the system with even a partly populated system. Any of the processor boards can be replaced by an adapter card, via which the data stream through the system can be monitored by means of a logic analizer. This is also supported by the flexible configurability of the system.

3.2.2 Clock distribution

At the time of design of the system, the electrical specifications of the individual processor chips were not fully known, so that calculation of the fan-in/fan-out of the clock lines was not possible. To avoid fan-out problems by the driving buffer and avoid driving too much capacitance (which would lead to degrading rise and fall times), we decided to buffer the clock lines for each separate two-chip processor board. Note that there is a potential problem involved in this approach. A buffer like

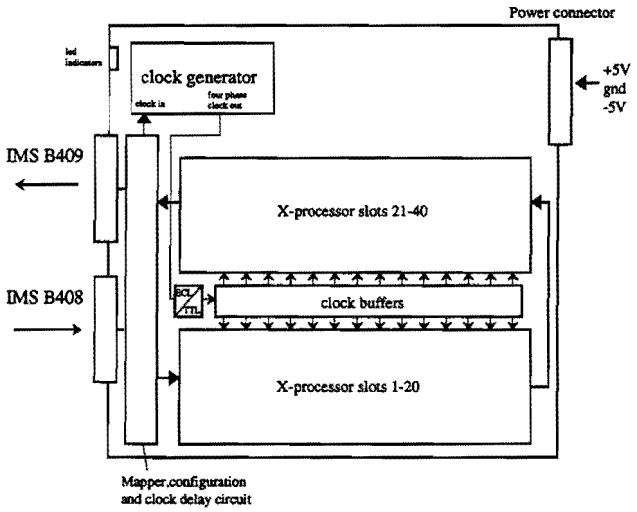

Figure 14: Layout of the mother board of the Difference Engine. This board houses the auxiliary circuitry needed for the processor array. The processors themselves can be found on daughter boards inserted in 40 slots.

the 74AS244 can have a skew — i.e., a difference of propagation delay between any two outputs of the device — of 5 ns. Such a skew is unacceptable regarding the minimum time interval between the two clock phases (min. 4 ns) and the set up times between two X processors in the pipeline. To overcome this, we would be forced to reduce the system's clock frequency.

Therefore we make use of 49FCT805 clock buffers. The length of the wires between the buffer and the clock pins on the processor board ranges from 6.9 to 8.9 inch. All clock lines are terminated by parallel termination resistors.

Each clock driver consists of two banks of drivers. Each bank drives five outputs from one TTL compatible CMOS input, divided over the eight sections of the processor pipeline. The skew between each section due to propagation delay on the clock lines is depended on the length between the points where the signal is connected to the main clock line. For this board the typical delay on a wire is 0.15 ns per inch. The maximum distance between the connections is 1.1 inch which relates to 0.165 ns delay between each section. Such a skew delay is insignificant.

As we can see in Figure 14 the clock generator is relatively far away from the processor array. This also implies that we have to deal with relatively long clock lines. For this reason the ECL/TTL converter is positioned at the beginning of the "clock distribution highway" between the two sections of the processor array rather than near the clock generator circuit itself. Naturally the ECL wires are terminated as usual for ECL.

4 Results

This section discusses what we have achieved with our prototype system. In summary: we have been able to demonstrate the concept but the realities of undertaking such an ambitious task in an academic setting has caused several problems.

The development of simulators was done in parallel with hardware specification and not prior to such specification. This reduced the delay in getting feedback on the feasibility implementing ideas in hardware from our VLSI designers but meant that proper simulation of the system was occasionally neglected. The experience with simulators is discussed in Section 4.1. One useful purpose of the simulator was that it served as a communication tool between the architecture designers/specifiers at the CWI and the VLSI design/implementation team.

4.1 Simulation Results

Apart from the detailed simulation of the VLSI design by the custom VLSI implementors a couple of other simulators of the Difference Engine were built:

1. The structural level simulator, and,
2. the functional level simulator.

The different simulators are characterized by different accuracy/performance trade-offs. When we refer to simulators we mean software systems that not only implement our algorithms but reflect some features of the hardware and its limitations. At the very least the communication with the simulator is in terms of the same instruction set as the hardware system. In this sense simulators were build for the difference engine. Other components of the architecture were simulated at an algorithmic level.

Structural Level Simulator

In a layered set of graphics hardware simulators, a *structural level simulator* was build to simulate the operation of the Difference Engine before is was actually available in VLSI[2]. This simulator bridges the gap between hardware fidelity on the one side, and sufficient performance to visualize graphics algorithms on the other. We needed such a simulator to validate the hardware design, and to visualize the result of the software that should run on it. On the one hand, the applications programmer must be able to verify that the output from a shading algorithm will be accurate on the target hardware. On the other hand, the hardware specialist must be able to trace the data and commands as they pass through the simulated hardware, and be able to interrogate the state of any part of the system at any stage. It is this combination of interests that makes this level

of simulation a useful adjunct in dealing with one of the problems which crop up in hardware design: the communication between applications programmers and hardware specialists.

Algorithmically, the simulator was used to test routines implementing angular interpolated shading, as well as those for anti aliasing via exact area integration. The simulator is the only medium, short or the hardware itself, which offers sufficient numerical accuracy to compute the pixel intensities. With it, we verified that the hardware could execute the graphics algorithms correctly and that the limitations on numerical accuracy and range were graphically acceptable.

The simulator revealed several errors in the hardware documentation. The simulator allowed us to explicitly describe our observations, ask the appropriate questions, and interpret the answers to resolve the errors. All were attributed to errors in the documentation.

By testing sequences of instructions rather than testing the functionality of individual hardware instructions, we can observe the relationships (and problems) which arise only among multiple commands. The simulator has served as a liaison between software and hardware engineers. It has been used to formulate and specify questions about the hardware before asked of the hardware designers; and to evaluate their answers. A number of non-trivial ambiguities have been resolved this way.

Functional Level Simulator

The *functional level simulator* was optimized for speed. The simulator accepted the same instruction set as the Difference Engine but employed instruction codings and word sizes appropriate for fast execution on workstation platforms.

Such simulators were useful for exploring high level ideas and testing collections of instruction datasets to compare their output to the prototype system.

The investigation of wavelet image decoding was done exclusively on the functional level simulator (see Section 4.5). Alternative shading algorithms were also initially verified on this type of simulator.

4.2 Documentation

This project has been subject to a number of delays. This is in part due to the imperfect standards of documentation of the subsystems. In many cases the documentation was so unreliable (or lacking) that only an experimental investigation of the hardware could produce firm answers.

Apart from urging better documentation we would also strongly suggest the use of software simulators as a documentation and communication tool. We found that different team members (engineers, computer scientists, research managers) had very different communication styles but the concrete results of an understandable simulation system can form a powerful persuasive documentation tool.

4.3 Design Lessons

In a multi-discipline project like this, the different "worlds" of application programmers and technicians was foreseen. It was somewhat naively expected that VLSI designers would — by nature — anticipate on the problems board-level system designers have to solve. This turned out to be not the case. For instance, the extreme demands imposed on the board-level design by the complex two phase bipolar clock signals required by the processor array caused many headaches. Given that the final prototype could not run at full speed anyway (see Section 4.4) it might have been a better decision to create a prototype VLSI system which had the ultimate potential to run at 11 ns speed but which had a simpler clock signal in the first stage of implementation.

The chip carrier is a PGA. The pin assignment of the PGA should have been arranged differently to accomodate routing of interconnections (in fact, for chips that have to be interconnected in a pipeline fashion a DIP would have been an even much better choice).

The speed requirements of the system are met by means of pipelining. For this, the data flowing through the array had to be skewed (see Section 3.1.1). At first sight it seems to make sense to skew data that enters the pipeline and unskew data that comes out. However in doing so, monitoring data that travels through the pipeline becomes more complicated[1]. By performing these skew and unskew operations on chip we would introduce an insignificant delay of 80×40 ns. This would have helped the board level designer and would also simplify maintenance of the system.

These imperfections did not preclude a successful implementation of the prototpe Difference Engine. However, they did cause annoyance and a significant delay in the project. It is worth it to invest some time to avoid these type of mismatches. This would not have prevented the following errors which are of a more serious nature.

One of these serious errors is an error in carry propagation. As it seems, the carry of the addition of the middle 12 bit bytes is propagated to the second least significant bit of the highest 12 bit byte. Fortunately a compensation for this error could be found, but how this could possibly escape detection in the lowest level simulation is not clear yet.

[1] **Note:** The system operates on 36 bit "words" subdivided in 3× 12 bit "bytes'. Each of these 12 bit "bytes" comes in one 3, one 5 and one 4 bit "nibble".

The lowest level simulation demonstrated that the individual processors could run at the speed required for high resolution systems (i.e., at a throughput rate of $\approx$ 11 ns). By being focussed on the timing characteristics of the individual processors, it was overlooked that the characteristics of the bond-pads — of which the design was taken out of a library — did not meet the requirements.

4.4 Prototype Performance

We consider the implementation of the prototype Difference Engine as being a success: the prototype can produce pictures on a CRT display directly from instructions and without buffering images in a frame buffer. Due to the characteristics of the bond-pads we were forced to reduce the pixel clock to 20 MHz. This is sufficient to drive a 640×480 display.

The prototype was build to demonstrate the feasibility of the architecture by displaying static images. Given the characteristics of the B408 that drives the Difference Engine, it should be clear that we cannot expect an outstanding dynamics behaviour of this prototype implementation. Nevertheless, we have some applications in which frame rates of about 50 Hz is obtained. This demonstrates the main objective of the architecture described in Section 2: due to a structured object list it can at times be sufficient to change just a few bytes in the instruction buffer.

4.5 Reconstruction of Encoded Signals on a Difference Engine

The X processor can interpolate any spline (polynomial) curve. Thus any signal that is expressed in terms of a *spline basis* can be reconstructed. Not only that, the architecture with its accumulator allows one to sum over incrementally generated output so that the splines can be summed over different scales to produce the final image to any required accuracy.

The reconstruction time depends *not* on the spacing of the knots in the splines (the lengths of the interpolation spans) but only on the number of knots. An image can be decompressed even at video rates provided that the number of knots are less than the number of pixels to be generated (by some fixed overhead per scanline).

This has opened the way for using this hardware to reconstruct images that have been coded with a *wavelet transform*. The wavelet transform is a multiresolution description of the image that can be decoded to yield more and more accurate reconstructions of an image. The transform also precisely locates high-frequency features in space and low-frequency signals in the frequency domain. In fact it is argued [1] that wavelet transforms perform better than the discrete cosine transform advocated by the JPEG standard, it fits in better with human perceptual aptitudes and is a more compact coding.

For a detailed discussion of the use of this system for wavelet reconstruction see [6].

5 Conclusion

The design decisions made in the project lead to a number of interesting consequences that have made parts of the architecture eminently suited to a far wider range of problems in computer graphics and image processing[2]. The Difference Engine described in this paper is designed based on the notion that rasterisation involves relatively simple operations repeated numerous times. For tasks like this, an implementation in VLSI of highly parallel relatively simple elements can compete with even the last generation of general purpose processors.

In spite of several "mismatches" described in Section 4.3 we have demonstrated that a Difference Engine for Graphics is technically feasible. It is now up to us to demonstrate that we know how to make use of an architecture like this for dynamic graphics applications. In order to do so we will have to build the next layer of the architecture, i.e., the Y processor level.

5.1 Lessons in Making things work in an Academic Environment

In retrospect, we consider ourselves lucky that the prototype Difference Engine could be made to work at all. It is now clear that we have to reduce the "luck"-factor and pay more attention to simulation on different abstraction levels. Also it turned out to be essential to force people to pay attention on documentation. Especially in a multi-disciplinary project these issues are too important to be overlooked.

In an academic environment we oftern have to work with people that can work on a project for a limited amount of time. Usually the budget is limited so that projects take longer than would be desirable. One should be aware that PhD students that may have a position for four years or so can be used about half of that time: they have a thesis to write as well. This again illustrates the need for good documentation. It should be possible for successors to pick up where others have left.

All in all one should realize that it is difficult to start up an ambitious project in an academic envirionment.

[2]Without going into a contemplation of the meta-levels of the design process it is interesting to observe that this generality of application resulted from bottom-up design. The initial top-down design produced an architecture for raster graphics (only). The bottom-up design that followed concentrated on extracting the lowest common denominator of primitive operations for synthesizing pixels — a language for manipulating related pixels. This vocabulary can now be used for expressing other facts about images.

5.2 Future Work

As mentioned in the above we will now have to focus on the higher level of the architecture. The basic requirements at these levels can be fulfilled by "of-the-shelf" hardware. We may have to design a bus arbitration unit to connect a set of Y processors to the Difference Engine.

By now we have gained experience in using the Difference Engine in its current form. By working with it, we came up with several ideas to improve the performance of the system. Also the study on reconstruction of encoded signals mentioned in Section 4.5 gave ideas of how to make the system more general by adapting the instruction set.

In its current form the Difference Engine can generate a limited type of textures only. We certainly will pay attention to this in forthcoming versions of the system.

Acknowledgement

We thank prof. L.O. Herzberger (University of Amsterdam) for providing the facilities and manpower to help build the prototype. Also we would like to thank STW for their support for this project, especially with respect to the budget. We also acknowledge Patric Marais (guest researcher from the University of Capetown) for his work with respect to wavelet-based image reconstruction.

References

[1] DESARTE, P., MACQ, B., AND SLOCK, D. T. M. Signal-adapted multiresolution transform for image coding. *IEEE Trans. Information Theory 38*, 2 (March 1992), 897–904. Special Issue on Wavelet Transforms and Multiresolution Signal Analysis.

[2] GURAVAGE, M. A., BLAKE, E. H., AND KUIJK, A. A. M. Xinposse: Structural simulation for graphics hardware. In *Rendering, Visualization and Rasterization Hardware*, A. Kaufman, Ed. Springer-Verlag, Berlin, 1994, pp. 9–19. Record of the Sixth Eurographics Workshop on Graphics Hardware, 1-2 September 1991, Vienna, Austria.

[3] JAYASINGHE, J. A. K. S., KARAGIANNIS, G., MOELAERT EL-HADIDY, F., HERRMANN, O. E., AND SMIT, J. Two-level pipelined systolic array graphics engine. *IEEE Journal of Solid-State Circuits 26*, 3 (1991), 229–236. Revised version of paper by the same title in Proceedings IEEE 1990 Custom Integrated Circuits Conference, Boston, Massachusetts. pp. 17.2.1-17.2.4.

[4] JAYASINGHE, J. A. K. S., KUIJK, A. A. M., AND SPAANENBURG, L. A display controller for an object-level frame store system. In *Advances in Computer Graphics Hardware, III*, A. A. M. Kuijk, Ed. Springer-Verlag, Berlin, 1990, pp. 141–170.

[5] KUIJK, A. A. M., AND BLAKE, E. H. Faster phong shading via angular interpolation. *Computer Graphics Forum 8*, 4 (1989). Please note that on p. 321 the definitions of a and b should be swapped.

[6] MARAIS, P. C., BLAKE, E. H., AND KUIJK, A. A. M. Adaptive spline-wavelet image encoding and real-time synthesis. In *Proceedings of the IEEE International Conference on Image Processing (ICIP94)* (1994). To appear.

[7] TEN HAGEN, P. J. W., KUIJK, A. A. M., AND TRIENEKENS, C. G. Display architecture for vlsi-based graphics workstations. In *Advances in Computer Graphics Hardware, I*, W. Strasser, Ed. Springer-Verlag, Berlin, 1987.