A Low-Cost, Power-Efficient Texture Cache Architecture

Iosif Antochi, Ben Juurlink, and Andrea Cilio
Computer Engineering Laboratory, Electrical Engineering Department, Delft University of Technology
P.O. Box 5031, 2600 GA Delft, The Netherlands
Phone: +31 (0)15 27 83644, Fax: +31 (0)15 27 84898
E-mail: tkg@ce.et.tudelft.nl

Abstract—**Texture mapping is a technique for adding realism to an image by mapping a texture image onto a surface. Realistic texture mapping requires large processing power and high memory bandwidth. Portable graphics devices, however, cannot afford large caches due to system requirements such as gate count limitations and low power. We propose employing a very small (128-512 bytes) texture cache between the graphics accelerator and the texture memory. Simulation results show that a small texture filter cache achieves remarkably high hit rates and significantly reduces the total power consumed by the memory hierarchy. For example, a 256-byte direct-mapped cache with a line size of 16 bytes reduces power dissipation by a factor of 7.0 compared to a conventional 16KB cache with a line size of 32 bytes, while reducing performance by a factor of 1.5. This corresponds to a reduction of the energy-delay product by a factor of 3.1. It is also shown that a widely-used benchmark exhibits anomalous behavior.**

Keywords—**Graphics accelerator, embedded systems, texture mapping, low power.**

I. Introduction

Low power is becoming more and more of a concern in microprocessor design. Especially when designing microprocessors that are to be embedded in portable devices, energy consumption is an important issue, since it must be supplied by batteries. Of the overall power dissipated by modern microprocessors, the power dissipation due to on-chip caches constitutes a significant part. For example, the on-chip D-cache of the StrongARM 110, a low-power RISC microprocessor, consumes 16% of its total power [1]. Furthermore, the rapidly growing portable electronics market is demanding low power devices, making techniques for energy-efficient caches an important research area [1], [2], [3].

In this paper we propose a low-cost, power-efficient cache architecture for portable graphics devices. In particular, we propose a cache architecture for texture mapping, which is one of the most expensive tasks (both in terms of processing power as well as memory bandwidth) that need to be performed by 3D graphics applications. Real-time texture mapping has become possible due to improvements in semiconductor technology, but supplying enough memory bandwidth remains a problem. Consequently, many high-performance texture cache architectures have been proposed (see, e.g., [4] and the references there). A common technique to improve memory bandwidth is to employ large caches. In portable graphics devices, however, large caches are unaffordable due to system requirements such as gate count limitations and low power consumption.

In this paper we, therefore, propose employing a very small (128-512 bytes) texture cache between the graphics accelerator and the texture memory. Such a small cache has been coined filter cache by Kin et al. [1], since it filters references to lower memory levels. Our hypothesis is that a small texture filter cache is able to exploit the access characteristics of texture mapping for the following reason. Texture mapping consists of mapping an image (texture) to an object. Any object can be decomposed in triangles after which each triangle can be rendered separately, having its own coordinates in the texture space. One method to render a triangle is to decompose it in horizontal lines (scanlines) and then to walk along each scanline. In order to find the color component of each point of the scanline, a projection in the texture space is performed and a value corresponding to an interpolation among neighbors is computed. The case of bilinear interpolation is illustrated in Figure 1. As can be seen, point P_1 corresponds to point U_1 in texture space. The value of U_1 is computed by bilinear interpolation among the texels T_1, T_2, T_3, and T_4. When computing the corresponding value for the next point of the scanline, P_2, the texels T_2 and T_3 are reused. In general, for each next point of the scanline two previous texels from the texture space are likely to be needed again. Furthermore, in the case of the more often used trilinear interpolation, it is likely that four of the previous eight texels are needed again. Due to this spatial locality, we assume that even a small texture cache can improve the performance of texture mapping substantially.

This paper is organized as follows. Section II describes the model used to estimate cache power consumption. Section III presents the experimental methods and workloads, and presents the experimental results. Conclusions and directions for future research are given in Section IV.

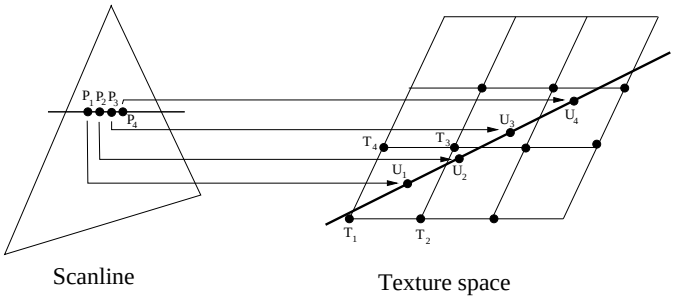


Fig. 1. Bilinear Interpolation.

II. Cache Power Consumption

In this section we describe the model used to estimate cache power consumption, and discuss and motivate the metrics chosen to measure energy efficiency. We did not use the model described by Kin et al. [1] because it can be rather inaccurate. For example, they assume that half the address lines switch during each memory request. However, this assumption is not valid for caches (and, moreover, biases small block sizes) due to temporal and spatial locality of references. In our estimations we, therefore, used precise counts obtained by simulation.

A. Cache Power Model

The cache power model is based on the cache timing and power consumption model used in the CACTI 2.0 tool [5], which in turn is based on the cache model proposed by Wilton and Jouppi [6]. The source of power dissipation in this model is the charging and discharging of capacitative loads caused by signal transitions. The energy dissipated for a voltage transition $0 \rightarrow V$ or $V \rightarrow 0$ is approximated by:

$$E = \frac{1}{2} C V^2, \tag{1}$$

where C is the capacitance driven. An analytical model of the cache power consumption includes the equivalent capacitance of cache components considered and the voltage swing of a transition. The power consumption is estimated by combining Eq.(1) and the transition count at the inputs and outputs of each modeled component, usually obtained by simulation.

To reduce wordline and bitline capacitance, and achieve minimal access time, the cache SRAM array is subdivided into subarrays. The wordlines are subdivided in N_{dwl} partitions, each containing the total number of cache rows and a full row decoder. The bitlines are subdivided in N_{dbl} parts, each containing $\frac{1}{N_{dbl}}$ of the total cache rows. Another parameter, N_{spd}, allows to map more sets in a single wordline and thus change the overall access time without breaking the array into smaller subarrays. The power model used takes into account the value of these organizational parameters. The optimal values for the array organization parameters $N_{dwl}, N_{dbl}, N_{spd}$ depend on the cache size and the block length, and are computed using CACTI.

The other cache parameters used in the formulae presented below are summarized in Table I. The number of columns N_{cols} and rows N_{rows} of the SRAM subarray can be derived from the cache parameters.

parameter	description
A	Associativity.
C	Cache size (in bytes).
B	Block size (in bytes).
D_{bits}	Data Word size.
A_{bits}	Address size.
V_{dd}	Supply voltage.
V_{pre}	Bitline precharge voltage.
T_{acc}	Access time.
N_{acc}	Number of cache accesses.
N_{rd}	Number of cache read accesses.
N_{wr}	Number of cache write accesses (write hits).
N_{rmiss}	Number of cache misses on a read.
N_{wmiss}	Number of cache misses on a write.
$N_{atr,m}$	Total address bit transitions on the outgoing lines.
$N_{atr,c}$	Total address bit transitions on the cache decoders.
W	Average number of bits written per write operation.

TABLE I
Cache parameters and transition counts.

To obtain good estimates of the power dissipated, accurate transition counts are essential. Transition counts can be determined exactly by simulation or, when this is not possible, can be approximated by multiplying the expected transition probability at a node by the cycle count [7]. In our estimations we use precise counts for cache accesses and address bit transitions to and from memory. The average width of a data item written to memory (W) is estimated assuming an equal distribution of bytes, half-words and 32-bit words, as in [1]. We also assume that the transition counts of address and data bits are evenly distributed between accesses that hit and miss the cache.

The following cache components are fully modeled: address decoder, wordline, bitline, sense amplifiers (see Figure 2), and data output drivers (Figure 3). In addition, the address lines going off-chip and the data lines (both going off-chip and going to the CPU, are taken into account. The following sections describe the energy contributions listed above to the overall dissipation.

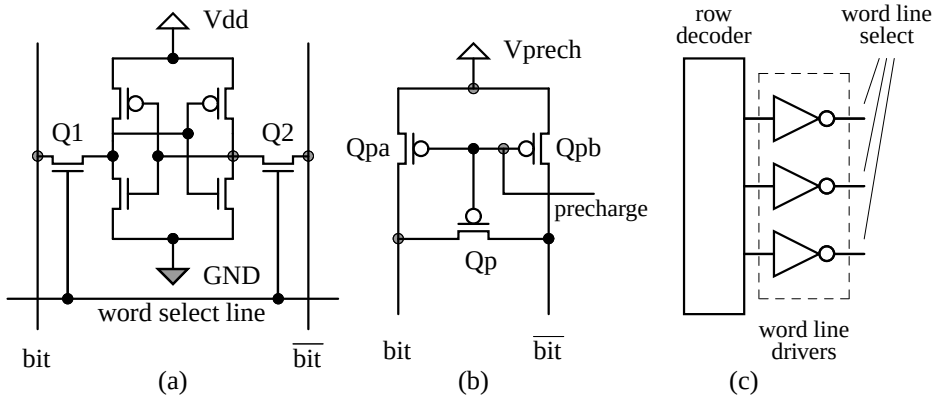


Fig. 2. Components of a static RAM considered in the cache power model: (a) 6-transistor bit cell; (b) bit line precharging logic; (c) word line drive logic.

B. Bitlines

A major fraction of the power consumption is needed to drive the bitlines of the data SRAM arrays. The cache bitlines are driven during precharging (Eq.(2)), read (Eq.(3)) and write (Eq.(4)).

$$E_{bit,pre} = \frac{1}{2} V_{dd} V_{pre} \cdot N_{acc} \cdot N_{cols} \cdot C_{bitline}, \quad (2)$$

$$E_{bit,rd} = \frac{1}{2} V_{dd}^2 \cdot N_{rd} \cdot D_{bits} A \cdot C_{bitline,out} + \frac{1}{2} V_{dd} V_{pre} \cdot N_{rd} \cdot N_{cols} \cdot C_{bitline}, \quad (3)$$

$$E_{bit,wr} = \frac{1}{2} V_{dd} V_{pre} \cdot (N_{wr} \cdot W + N_{rmiss} \cdot 8B) \cdot C_{bitline}. \quad (4)$$

The capacitance of the bitline, $C_{bitline}$, is given by:

$$C_{bitline} = N_{rows} \cdot \left(\frac{1}{2} C_{d,Q1} + C_{bwire} \right) + 2C_{d,Qpa} + C_{d,Qp} + C_{d,mux}$$

The value of drain and gate capacitances depend on the size of the transistor and are computed using the equations presented by Wilton and Jouppi. The factor $\frac{1}{2}$ accounts for the fact that the drain of the pass transistor of a bit cell is shared with the adjacent cell, therefore the capacitance $C_{d,Q1}$ is reduced by half. C_{bwire} is the capacitance of a bitline segment one bit cell high. $C_{d,Qpa}, C_{d,Qp}$ are the drain capacitances of precharge and equilibration transistors, respectively. The drain capacitance $C_{d,mux}$ is due to the bitline multiplexor at the input of the bit line sense amplifiers, which is present only if $8B \cdot N_{dbl} N_{spd} > D_{bits}$.

The capacitance $C_{bitline,out}$ is due to the input transistors of the sense amplifiers at the end of the bitlines and, if present, the drains of the bitline multiplexors.

$$C_{bitline,out} = 2C_{g,senseamp} + \left(\frac{8B \cdot N_{dbl} N_{spd}}{D_{bits}} \right) \cdot C_{d,mux}$$

Notice that the caches modeled are write-through, therefore a write operation causes a cache update only if the word is already in the cache (write hit).

C. Wordline

The energy dissipated by the wordlines is given by:

$$E_{word} = V_{dd}^2 \cdot N_{acc} \cdot (N_{cols} \cdot (2C_{g,Q1} + C_{wwire}) + C_{d,wdrv} + C_{g,wdrv} + C_{d,winv}). \quad (5)$$

The factor $\frac{1}{2} V_{dd}^2$ is multiplied by 2 because the word select signal is assumed to be pulsed, thus each read or write operation causes two transitions on the selected word line. The capacitance $C_{d,wdrv}$ is due to the drain of the P- and N-channel transistors of the wordline driver. The overall load of the wordline inverter stage at the input of the wordline driver is modeled as the combined capacitance of the driver gates $C_{g,wdrv}$ and the inverter drains, $C_{d,winv}$.

D. Sense Amplifiers

Each sense amplifier detects the voltage variation of a bitline resulting from a read operation and amplifies such variation to the full voltage swing V_{dd}. According to the cache model used, the energy dissipated in the sense amplifiers is a large fraction of the overall dissipation, and cannot be ignored:

$$E_{senseamp} = P_{sense} \cdot T_{acc} \cdot N_{acc} \cdot AD_{bits}. \quad (6)$$

P_{sense} is the average power consumption of a sense amplifier when a bitline transition occurs. Notice that the number of sense amplifiers is reduced by multiplexing the bitlines, hence the factor AD_{bits}.

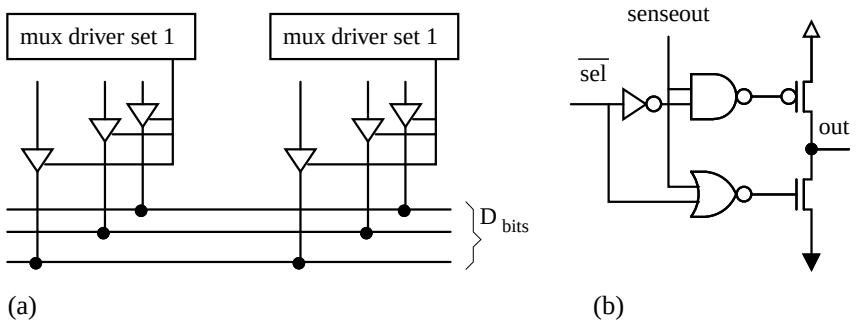


Fig. 3. Output data bus: (a) overview; (b) output driver (tri-state).

E. Data and Address Output

The data output driver drives the cache output data lines with the signals from the sense amplifiers. In a set-associative cache there are AD_{bits} sense amplifier outputs and only D_{bits} outgoing data lines, therefore each output driver is actually a tri-state that charges or discharges the output line only if the corresponding set is selected (see Figure 3).

The energy dissipated in the data and address output lines is given by:

$$E_{dout} = \frac{1}{2}V_{dd}^2 \cdot N_{rd} \cdot D_{bits} \cdot (C_{d,outnor} + C_{g,outdrv} + A \cdot C_{d,outdrv} + C_{wwire} \cdot 8BA \cdot N_{spd} \cdot vstack + C_{doutc}) + \frac{1}{2}V_{dd}^2 \cdot \frac{1}{2}W \cdot N_{wr} \cdot C_{doutm}, \quad (7)$$

$$E_{aout} = \frac{1}{2}V_{dd}^2 \cdot N_{atr,m} \cdot C_{aoutm}. \quad (8)$$

where $C_{d,outnor}$ is the total capacitance of the drains of the NOR gate P- and N-channel transistors, $C_{g,outdrv}$ is the gate capacitance of the output driver and $C_{d,outdrv}$ is the capacitance of the output line due to the drivers attached to it. The remaining output line capacitance is due to the wire (whose estimated length is $8BA \cdot N_{spd} \cdot vstack$, where $vstack$ is the number of subarrays arranged horizontally) and C_{doutc}, the overall capacitance of the path between cache output line and the CPU. Similarly, C_{doutm}, C_{aoutm} are, respectively, the capacitance of the data and address paths going off-chip.

F. Address Decoder

The cache model used in CACTI assumes that each subarray has its own decoder, which is made of three stages. The first stage contains predecoder blocks. These take 3 address bits as input and produce a 1-of-8 output signal by means of 8 3-input NAND ports. The 1-of-8 codes from different blocks are combined using N_{3to8}-port NOR gates, where

$$N_{3to8} = \left\lceil \frac{1}{3}\log_2(N_{rows}) \right\rceil.$$

Each NOR gate controls the selection signal of one word-line.

The energy dissipated in the decoder is subdivided in three components corresponding to the three stages:

$$E_{adec,1} = \frac{1}{2}V_{dd}^2 \cdot 2N_{atr,c} \cdot (C_{d,ddrv} + N_{dbl}N_{dwl} \cdot 4C_{g,3to8} + 2BA \cdot N_{dbl}N_{dwl} \cdot C_{wwire}), \quad (9)$$

$$E_{adec,2} = \frac{1}{2}V_{dd}^2 \cdot N_{acc} \cdot \left(C_{d,3to8} + C_{g,decnor} \cdot \frac{N_{rows}}{8} + C_{bwire} \cdot N_{rows} \right), \quad (10)$$

$$E_{adec,3} = \frac{1}{2}V_{dd}^2 \cdot N_{acc} \cdot (C_{d,decnor} + C_{g,winv}). \quad (11)$$

The energy dissipated in the first and third stage is based on the precise transition counts obtained via simulation: $N_{atr,c}$ and N_{acc}, respectively. The capacitance load of the drivers at the input of the first stage, $C_{d,drv}$, due to the drains of the transistors, is shared across all the subarray decoders. Each decoder contributes to the overall capacitance of the first stage with the wire that connects the driver output with the NAND gates of the 3-to-8 predecoder blocks and with the transistor gates of the blocks. The factor 2 of $N_{atr,c}$ accounts for the fact that for each address bit there also the inverted bit must be driven. For each address bit undergoing a transition, all 8 gates of the 3-to-8 block undergo a transition (4 are charged and 4 are discharged). The factor 2 before the term due to the wire capacitance accounts for the fact that the SRAM arrays are assumed to be organized so that the predecoders are at the center of the array and the wires that connect them to the input driver are a quarter of the total array length. The last stage spends power to drive the capacitance of the NOR drains of the selected word line and the gates of the inverter at the input of the wordline driver. Note that the

power consumed by the wordline inverter is accounted for in Eq.(5).

The power consumed in the tag path to select and retrieve the cache block tag is also taken into account. The expressions for the tag path are similar to those for the data path. In this case, the number of columns is the number of tag bits. The tag array is organized in subarrays independent from the data subarrays, i.e. the tag path is characterized by three organizational parameters. The model presented above does not include the power dissipated by comparators that verify a tag / address match, nor data steering logic and cache control logic. These components give a minimal ($< 2\%$) contribution to the overall power dissipation.

G. Energy-Related Metrics

In order to evaluate the efficiency of the texture cache architecture, we also measure the energy-delay (E-D) product. This metric was proposed by Gonzales and Horowitz [8], who argue it is superior to the commonly used power or energy metrics because it combines energy dissipation and performance.

To compute the delay D we assumed a clock frequency compatible with the access times estimated by CACTI. The E-D product is given by:

$$ED = E \cdot D = P \cdot D^2 = P \cdot (N_{cycles} \cdot T_{clock})^2. \quad (12)$$

Although the E-D metric presents important advantages, like the reduced dependence from technology, clock speed and implementations, we also present our measurements for energy consumption. As argued in [7], the energy consumption is an important metric for battery-operated processors in portable devices, because it determines their battery duration.

III. Experimental Evaluation

A. Tools and Benchmarks

One of the most often used benchmarks for 3D Graphics Accelerators is the QuakeIII Arena application. This benchmark requires an OpenGL compliant library, for which we used Mesa. Since the source of the Mesa library is freely available, we could instrument the code to generate address traces. We used our own trace-driven cache simulator (called BOCS) to gather information such as the number of cache accesses and transition counts. This information was subsequently fed to CACTI, which produces power, energy and energy-delay estimates.

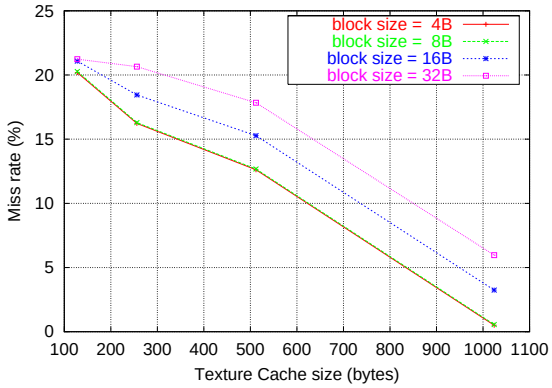
A.1 Details on Tools Settings

The QuakeIII Arena application was chosen due to its flexibility and a rich set of features that allowed us to customize the benchmark to our needs. The graphic options were set to "High Image Quality" (trilinear filtering) and the screen resolution was chosen as "640x480". One of the features of QuakeIII is to be able to run a prerecorded demo as a reference. The first available record (demo001) was used and we gathered statistics for all frames starting from frame 40 (the first 40 frames are not relevant since they are introductory frames). The Mesa library was instrumented with new code so that each reference to a 2D texture element was logged to a file. Write operations were not logged since we are interested in read operations mostly and assume that the number of writes is insignificant compared to the number of reads. Furthermore, write operations are encountered at the initialization phase only. The address trace file obtained from Mesa was simulated using BOCS for several cache and block sizes. We only consider direct-mapped caches, because associativity increases the amount of data and control information read out on each cache access, and therefore associative caches consume more power [1]. The assumptions we made for the cache simulator are that the word size is 32 bits and that the miss penalty is $12+w$ cycles, where w is the block size in words.

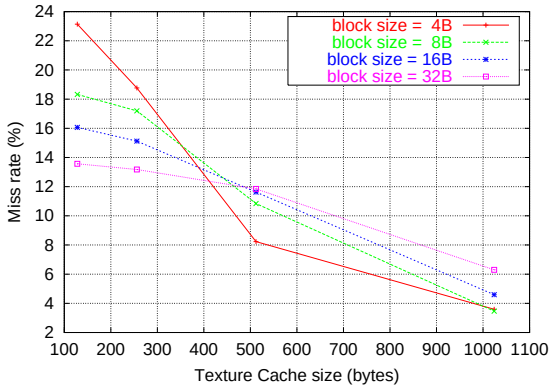
B. Experimental Results

At first, we used the Mipmap benchmark from the Mediabench suite [9] instead of the QuakeIII Arena Demo. However, this benchmark showed anomalous behavior, as is explained below.

Figure 4(a) depicts the miss rates for the texture mapping phase of the Mipmap benchmark as a function of the cache size, for four different block sizes. Somewhat surprisingly, the miss rate increases with the block size. This effect became even more significant when we changed the sampling method from point sampling to the more often used trilinear interpolation (Figure 5(a)). Although it can be expected that for small caches small block sizes are better due to the small number of entries and the risk of cache conflicts, it is really surprising that, for example, a 256-byte cache with a block size of 8 bytes exhibits a higher miss rate than a 128-byte cache with a block size of 4 bytes, even though they have the same number of entries. Furthermore, as explained in the introduction, trilinear filtering should be able to take advantage of spatial locality. We, therefore, studied the code of the Mipmap benchmark, and discovered that there was not only a high degree of spatial locality, but also a high degree of temporal locality

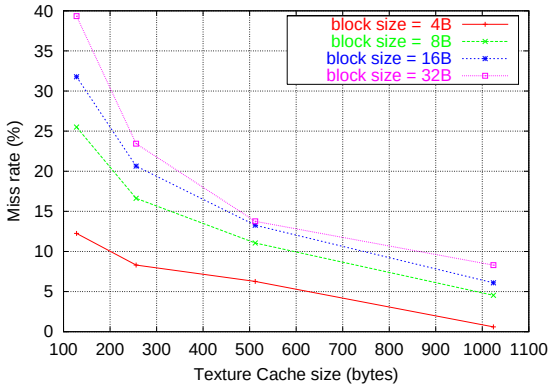


(a) Behavior using original benchmark

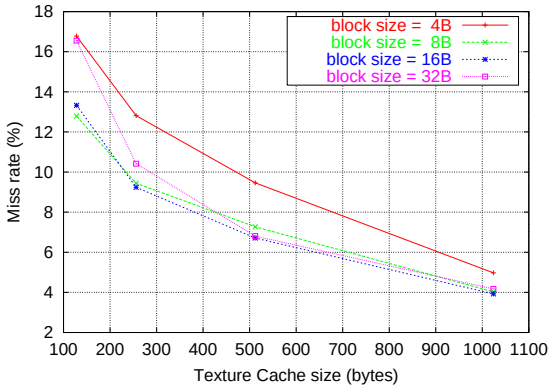


(b) Realistic behavior

Fig. 4. Mipmap miss rates using point sampling.



(a) Behavior using original benchmark



(b) Realistic behavior

Fig. 5. Mipmap miss rates using trilinear interpolation.

because most of the texels from the texture were covering more than one pixel. So, the same texel was reused for the coverage of more than one pixel. Such a behavior is not realistic since it can be encountered only when, for instance in a 3D game, we are very close to a wall and we have a low resolution texture. Figures 4(b) and 5(b) show the miss rates after modifying the number of tiles applied on the respective surface. They show that this indeed changes the cache behavior, and that the Mipmap benchmark is not representative. We therefore used the QuakeIII Arena Demo instead.

The results obtained were very similar across frames. We, therefore, present only the results for one representative frame (Frame 520).

Table II depicts the power consumed by the texture cache when frame 520 is processed as a function of the cache size and block size. As can be expected, the smaller the cache and the block size, the lower the power consumption. One interesting observation about this table is that doubling the block size does not imply a similar increase in power consumption, although there are twice as many lines. This can be explained by the improved hit rate that compensates for the increased power per reference, and by the spatial locality of references (larger blocks imply fewer address line transitions).

Table III shows the energy (power times delay) consumption as a function of the cache and block size. In this case the best results are not necessarily obtained for the smallest cache and block sizes, since they incur a large delay. It appears that the best results are achieved when the cache size is 128 or 256 bytes, and that the block size is less relevant. Although employing larger blocks improves the hit rate and reduces the number of address line transitions, this appears to be almost neutralized by the increase in the number of bitline and wordline transitions.

The miss rate as a function of the cache and block size is depicted in Table IV. It shows that the 128-byte filter cache is too small to be useful as a texture cache, since it incurs miss rates in the order of 20-50%. We also observe that although the miss rate decreases when we increase the

block size, the improvements quickly become smaller.

However, the most important metric for our study is the energy-delay product, which is shown in Table V. As can be seen, the 256-byte cache with a 16-byte block size yields the best result. This was the case for most frames, but for some frames characterized by intense geometry requirements, higher polygon counts and higher overdraw, the 512-byte cache with a block size of 16 bytes yielded the best result. To confirm this, Table VI depicts the energy-delay product for one of these frames.

For comparison reasons, we have also included the results for a conventional cache size (16KB) in Table VII. It can be verified that a 16KB cache consumes much more power and energy than a small texture filter cache, but of course, it exhibits a lower miss rate. Overall, however, the filter cache yields a much smaller energy-delay product. For example, when processing frame 520, the 256-byte filter cache with a block size of 16 bytes improves the energy-delay product by a factor of 2.1 compared to a 16KB cache with the same block size.

Cache Size (B)	Block Size (B)			
	4	8	16	32
128	11.16	13.47	13.67	13.45
256	39.02	45.87	46.39	48.90
512	49.72	59.23	61.94	59.15
1024	64.75	77.14	83.80	85.03

TABLE II
POWER FOR FRAME 520 (MW).

Cache Size (B)	Block Size (B)			
	4	8	16	32
128	7.34	7.11	6.74	7.39
256	7.08	7.11	6.85	7.54
512	7.71	8.02	8.03	7.96
1024	9.03	9.51	9.87	10.20

TABLE III
ENERGY FOR FRAME 520 (MJ).

Cache Size (B)	Block Size (B)			
	4	8	16	32
128	46.67	33.39	26.89	24.51
256	7.31	4.67	3.67	3.28
512	5.14	3.25	2.46	2.22
1024	3.84	2.33	1.66	1.49

TABLE IV
MISS RATE FOR FRAME 520 (%).

Cache Size (B)	Block Size (B)			
	4	8	16	32
128	4.82	3.75	**3.32**	4.05
256	1.28	1.10	**1.01**	1.16
512	1.20	1.09	**1.04**	1.07
1024	1.26	1.17	**1.16**	1.23

TABLE V
ENERGY*DELAY FOR FRAME 520 (MJ*S). FOR EACH CACHE SIZE, THE BLOCK SIZE THAT ACHIEVES THE BEST RESULT IS SHOWN IN BOLD.

Cache Size (B)	Block Size (B)			
	4	8	16	32
128	11.40	8.75	**7.77**	9.53
256	5.12	4.25	**3.93**	4.74
512	2.57	2.34	**2.25**	2.33
1024	2.50	**2.39**	2.44	2.66

TABLE VI
ENERGY*DELAY FOR FRAME 120. FOR EACH CACHE SIZE, THE BLOCK SIZE THAT ACHIEVES THE BEST RESULT IS SHOWN IN BOLD.

	Block Size (B)			
	4	8	16	32
E*d	2.39	2.33	2.15	3.14
E	21.70	22.70	21.60	32.00
P	197.93	220.95	217.50	326.54
Miss Rate	1.39	0.75	0.43	0.27

TABLE VII
RESULTS FOR A 16KB CACHE USING FRAME 520.

IV. CONCLUSIONS

In this paper we have proposed and evaluated using a small filter cache between the graphics accelerator and the texture memory in portable graphics devices. Such devices cannot afford large caches due to restrictions such as gate count limitations and the need for low power.

For most frames in the used record a cache of 256 bytes and with a block size of 16 bytes yields the best results w.r.t. the Energy-Delay metric, but for some frames better results are obtained when a 512-byte cache with a block size of 16 bytes is used. The performance penalty compared to a conventional cache size of 16KB is approximately a factor of 1.5, and it remains to be evaluated if this is acceptable. We have also shown that a widely-used benchmark exhibits anomalous behavior.

In the future we intend to evaluate the robustness of the texture filter cache architecture by conducting experiments using other scenes with different characteristics. We also intend to complete the design of a low-power graphics device and see if the low-power cache design translates to a low-power graphics system.

REFERENCES

[1] Johnson Kin, Munish Gupta, and William H. Mangione-Smith, "Filtering Memory References to Increase Energy Efficiency," *IEEE Trans. on Computers*, vol. 49, no. 1, Jan. 2000.

[2] M.B. Kamble and K. Ghose, "Energy-Efficiency of VLSI Caches: a Comparative Study," in *Proc. Tenth Int. Conf. on VLSI Design*. IEEE, Jan. 1997, pp. 261–7.

[3] Hsien-Hsien S. Lee and Gary S. Tyson, "Region-based Caching: An Efficient Memory Architecture for Embedded Processors," in *Proc. CASES*, San Jose, CA, Nov. 2000.

[4] H. Igehy, M. Eldridge, and K. Proudfoot, "Prefetching in a Texture Cache Architecture," in *Proc. EUROGRAPHICS/SIGGRAPH Workshop on Graphics Hardware*, 1998, pp. 133–142.

[5] G. Reinman and N.P. Jouppi, "An Integrated Cache Timing and Power Model," Tech. Rep., COMPAQ Western Research Lab, Palo Alto, California, 1999.

[6] S.J.E. Wilton and N.P. Jouppi, "An Enhanced Access and Cycle Time Model," Tech. Rep. 5, Digital Western Research Laboratory, Palo Alto, California, July 1994.

[7] David Brooks, Vivek Tiwari, and Margaret Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proc. 27th Annual Int. Symp. on Computer Architecture*, Vancouver, British Columbia, June 12–14, 2000, pp. 83–94.

[8] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1258–1266, 1996.

[9] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," in *Proc. 30th Annual Int. Symp. on Microarchitecture*, 1997, pp. 330–335.