Re-visit Blocking Texture Cache Design for Modern GPU

Jhe-Yu Liou   Chung-Ho Chen

Department of Electrical Engineering, National Cheng Kung University
No.1, University Road
Tainan, Taiwan

elvis@casmail.ee.ncku.edu.tw   chchen@mail.ncku.edu.tw

Abstract

Texture cache plays a significant position in GPU design especially in a limited memory bandwidth environment such as mobile SoC system. In this paper, we evaluate the 6D blocking texture cache design through a sophisticated GPU simulator using DRAM memory model. Our experiment reveals that using a larger block can take advantage of the spatial locality of texel accesses, however, fetching a larger block which requires several burst runs in DRAM access, results in poor memory access efficiency. As a result, the block size used has to match with the DRAM burst length for the best memory access efficiency.

Keywords-component; DRAM model, Texture cache, GPU architecture.

I. Introduction

Mobile graphic processor unit (GPU) is the most bandwidth demander in a hand-held device, especially for those using shared memory approach. The major memory access demand in GPU comes from the texture operations, and as a result, all GPU architectures have used some form of texture caches.

Due to the texture image characteristics, Igehy has proposed to use 6D blocking texture data to dramatically increase cache efficiency[1]. The main idea, shown in Fig.1, is to re-address the texture data, which in the past is linearly placed in the memory like Fig.2(b), into a 3-level and 2-dimension organization and convert the texture coordinate to perfectly fit the general cache hierarchy as tag, entry (index), and block. This method not only exploits texture spatial locality, but also takes the advantage of DRAM burst access because of the ordered sequence of texels in the blocks as depicted in Fig.2(c).

In this paper, we evaluate a 6D blocking texture cache design for modern shader based GPU with detailed DRAM model.

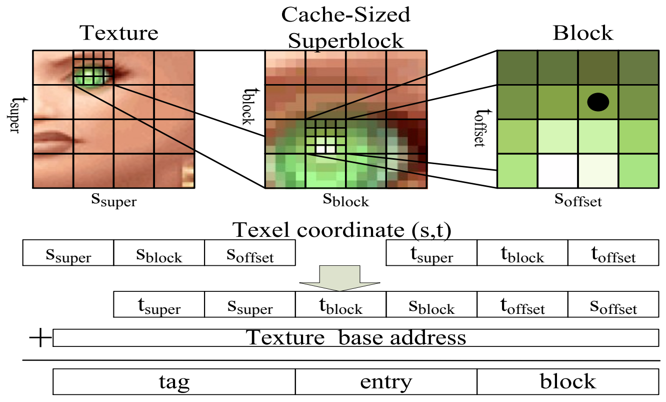


Fig.1 Texture data in 6D blocking organization and how texel address is represented in general cache from the texel coordinate (s,t).

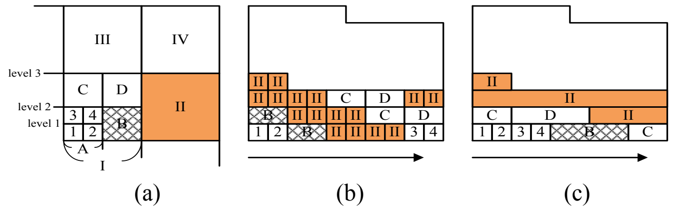


Fig. 2 (a) texture data sequence (b) traditional linear fashion and (c) 6D blocked fashion in memory.

II. Simulation environment

In Hakura's research[2], Igehy chose 16 texels (4x4 texels) per block which worked quite well and used lesser bandwidth than the other configurations. However, the result in Igehy's work has not been based on contemporary DRAM models. Also they use the fixed-function OpenGL test cases unlike modern graphic programming model that bases on the OpenGL Shading language. To render the issues we address above, we develop our own simulation environment which includes a full OpenGL ES 2.0 API implementation and a soft-pipe GPU simulator. To cope with the concept of 6D blocking data organization, we leverage the triangle traversal strategy in Intel larrabee[3] that will test a triangle in 16x16-pixel supertiles. Once a supertile is covered by the triangle, the traversal scheme will subdivide the supertile into four quarter tiles recursively until the 2x2 tile size is reached, and then put these four pixels on the GPU's shader. The GPU simulator has a unified shader which allows 4 threads to be synchronously executed under single instruction multiple thread (SIMT) scenario. The unified shader is also accompanied by a texture unit with the 6D texture cache which can be flexibly configured.

A.Memory model

To precisely model DRAM timing, we create a DDR DRAM simulator based on the following core DRAM parameters, dram clock(CLK), column latency (CL), row to column delay (tRCD), and row precharge time (tRP). The timing scheme is described as below:

```
1. Initial access: time = tRP + tRCD + CL
2. All following accesses:
   if this access's desired row differs from previous.
      time += tRP + tRCD + CL
   else
      if this access is a continuous burst access
        time += CLK/2 (Because of double data rate, DDR)
      else
        time += CL
```

To model a GPU in a mobile SoC chip in which the GPU has no dedicated local memory, we pick low power DDR(LPDDR2) DRAM from Micron instead of graphic DDR(GDDR) DRAM as our target simulated DRAM even though GDDR has larger

TABLE I
Parameters of Micron LPDDR2 800Mhz DRAM. This table hides write latency because all texture operations are read only.

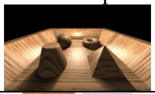
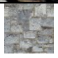
CLK	CL(read)	tRCD	tRP	width	burst length
2.5ns	15 ns	18 ns	18 ns	32 bits	1, 2, 4, 16

memory width and longer burst length. The Micron LPDDR2 DRAM parameters can be found in Table I.

III. Benchmarks

In this paper, three benchmarks are used. They are teapot, StoneFloor, and FourShapes. Each benchmark's characteristic can be found in Table II. The teapot is a simple object but is popular in 3D graphic tutorial with single texture mapping. The StoneFloor achieves the normal mapping effect by utilizing two texture maps as the color map and the normal vector map respectively. The FourShapes demonstrates the parallax occlusion mapping[4] effect by using only two triangles to accomplish lots of image details just like constructed by hundreds or thousands triangles.

TABLE II
Benchmark scenes and their characteristic

Benchmark	Teapot	StoneFloor	FourShapes
			
Texture image			
Triangles	6400	2	2
Texels / pixel	6.48	10.72	288.15

To analyze the result, all textures are stored in 32 bits per texel even though the last 8 bits are redundant in some textures. Besides, All screen resolution in benchmarks is set in 1024x768 with 2X anisotropic filter enabled.

IV. Performance Analysis

In order to evaluate the relationship between cache organization and memory utilization, we configure the different combination of the cache block-size in 2x2, 4x4, and 8x8, and cache size in 4, 16, and 64KB. Because at most two texture images are used in our benchmarks, the cache is 4-way set association. The maximum burst length in our DRAM model is 16 words. This is obviously harmful to fetching of an 8x8 cache block size because once a cache miss has occurred, the cache needs four burst read accesses to retrieve the data. In contrast, both 2x2 and 4x4 block-size cache can mostly get data in a single burst read access.

Fig. 3 shows the simulation result under various cache block sizes, total cache size, and different benchmarks. Both teapot and StoneFloor show very similar trend. This is because the way they use texture mapping is completely the same except the number of texture images used.

For these two benchmarks, 4x4 block size works quite well in

TABLE III
Cache miss penalty. Note: 8x8i means 8x8 cache block size but with ideal 64 burst length supported in the DRAM model.

block-size	min (ns)	max (ns)	avg (ns)
2x2	18.75	104.5	61.625
4x4	33.75	119.5	76.625
8x8	135	207	171
8x8i	93.75	179.5	136.625

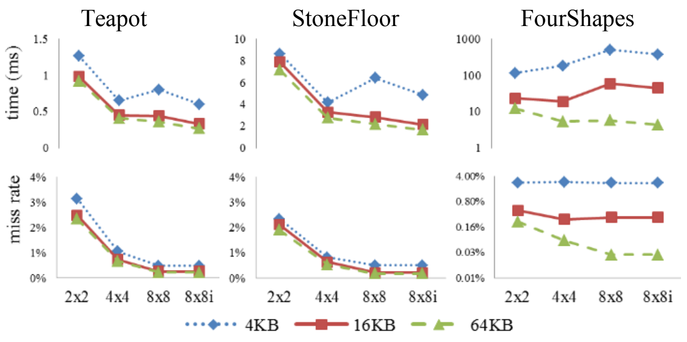


Fig.3 The total memory access time and cache miss rate in different cache parameter and benchmarks. Note: the Y-axis in both FourShape diagrams are drawn in log scale.

all cache sizes. The 8x8 block size has even better performance in 16KB and 64KB. The range of miss penalty can be simply calculated, which is listed in Table III. Considering the miss penalty in Table III, an 8x8 block size needs to decrease the cache miss by about 223% in average compared to 4x4 case to have the same total memory latency. This is the reason why the 8x8 block size in 4KB cache size has higher total memory access time due to the insufficient cache miss drop which, however, can be improved by using the DRAM to have longer burst length for the 8x8 configuration in Table III and Fig.3. Let's look at the FourShapes. The miss rate is about 2% for any block-size under 4KB cache size. Having an average of 288 texel fetches per pixel means that each pixel encounters at least one conflict miss before the next neighbored pixel goes. This destroys the cache spatial locality and leads very poor cache efficiency. Consequently, a large texture cache is required.

V. Conclusion

In this paper, we have developed an OpenGL ES 2.0 GPU simulator which includes a realistic DRAM model and we evaluated the design of 6D blocking texture cache. Our experiment shows that using a larger block can take the advantage of spatial locality, however, fetching a larger block which requires several burst runs in DRAM access, results in poor memory access efficiency. We have observed that the block size used has to match with the DRAM burst length for the best memory access efficiency.

Acknowledgement

This research is supported by the National Science Council of Taiwan under grand NSC 102-3113-P-006 -018.

References

[1] H. Igehy, M. Eldridge, and K. Proudfoot. “Prefetching in a texture cache architecture,” In Proc. ACM SIGGRAPH/ EUROGRAPHICS conference on Graphics Hardware, pp. 133-142, 1998
[2] Z. S. Hakura and A. Gupta. “The design and analysis of a cache architecture for texture mapping,” In Proc. 24th annual international symposium on Computer architecture (ISCA '97), pp108-120, 1997.
[3] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, et al. “Larrabee: a many-core x86 architecture for visual computing,” ACM Trans. Graphics (Proc. SIGGRAPH ‘08), vol.27, no. 3, Aug. 2008.
[4] N. TATARCHUK. Dynamic parallax occlusion mapping with approximate soft shadows. In: Proceedings of the 2006 symposium on Interactive 3D graphics and games. ACM, 2006. p. 63-69.