Adaptive Selection of an Index in a Texture Cache

Chun-Ho Kim and Lee-Sup Kim
Department of Electrical Engineering and Computer Science,
KAIST (Korea Advanced Institute of Science and Technology)
E-mail: chkim@mvlsi.kaist.ac.kr, lskim@ee.kaist.ac.kr

Abstract

For a specified application, there is an opportunity to improve cache performance by smart choosing of index bits of a cache. A texture cache for texture mapping of 3D computer graphics is an example. Texel(texture pixel)-access characteristics for texture mapping are dependent on the rasterization order of a polygon.

In this paper, we introduce A-index, a method to reduce memory bandwidth required for fetching texture image data by reducing cache miss through adaptive selection of an index according to span direction of a rasterizer. By designing a texture mapping hardware including a texture cache, it is verified that the number of cache misses and the number of total cycles are reduced by 21.6 % and 8.8 % for rendering of textured scenes. In addition, we show that the number of cache misses in a texture cache can be estimated more accurately by calculating intra-span replacement when the proposed A-index is used.

1. Introduction

Computer graphics has become an important technique in many applications such as CAD tools, game, film, virtual reality and etc. Although many techniques are used in 3D Computer Graphics, texture mapping is one of the most successful and popular techniques in high-quality image synthesis. Especially, texture mapping creates the appearance of complexity without the tedium of modeling and rendering every 3D detail of a surface [1]. Moreover, texture mapping is a basis of other mapping techniques such as shadow mapping, environment mapping, bump mapping and etc. However, the greatest weakness of the texture mapping is that it requires high memory bandwidth to fetch the texture image data. The use of a cache is important in improving processing speed of a system. A well-tuned cache hierarchy and organization can induce the increase of system performance and bandwidth saving in a system bus.

So far, many researchers have proposed cache architectures for texture mapping. In 1997, Zihad S. Hakura [2] presented a paper about the design and analysis of a texture cache. He proposed to use a cache for texture mapping and analyzed the cache organization such as cache size, line size and associativity. In 1998, Homan Igehy [3] proposed to use caching in conjunction with pre-fetching for hiding memory latency. Also, Michael Cox [4] proposed to use multi-level texture caching for solving the problem of cache size and bandwidth. Physically, he used a two-level cache: a small L1 cache to directly connect with a graphics accelerator and a large L2 cache to reduce bandwidth requirement through less cache miss. Besides, the capacity of bus bandwidth has been increased as shown in the development of AGP 2/4/8X. Although many researches have been done in the view of cache hierarchy and organization, texture memory bandwidth is still a critical issue in the design of a 3D graphics system. The reason is that the bandwidth requirement is also increasing by more texture usages and more sophisticated filter methods which need more texels per pixel like *footprint assembly* [5]. Meanwhile, there was an attempt to improve cache performance by optimal selection of index bits considering application set [6]. However, the method has a weakness that the index bits are fixed for the whole run time.

In this paper, we propose a new method to reduce texture bandwidth requirement by reducing cache miss-rate through adaptive selection of an index. We call it *A-index*. By using the *A-index*, we can reduce cache miss by changing cache index bits of a texture cache considering data access characteristics. The key characteristics of texture data access are that texture data are a 2D data and access direction is not either horizontal or vertical but randomly directed. Conventionally, the horizontal coordinate of a texel has been used for the cache index. However, we can use the horizontal or the vertical coordinate of a texel as the cache index adaptively by checking the trends of texel-access direction on the texture image. The cache miss reduction obtained by using the *A-index* diminishes pipeline stalls, bus bandwidth required for texel fetch, and time for texture mapping. Also, the saved bus bandwidth can lead more performance increase due to the reduced bus contention. Therefore, we can speed up rendering time in texture mapping by the *A-index*. Besides, we decomposed cache miss into three parts based on the triangle spans. We will show that we can estimate cache performance through the decomposition and linear approximation of the relation of the number of cache misses and the number of intra-span replacements.

2. Backgrounds

2.1. Texture Mapping & Triangle Span

In 3D computer graphics, surfaces of a 3D object are modeled into sum of triangles. Mapping of a 2D image onto the surface is texture mapping. The image mapped onto the surface is called a texture map and its individual element, texture pixel, is often called a texel. The texture mapping consists of two steps: the first is a transform from the 2D texture space to the 3D object space and the second is a transform from the 3D object space to the 2D screen space [7]. The composition of two transforms is denoted as a rational linear projective transform as shown in (1). The x_s, y_s and u, v are coordinate values of a pixel in the screen space and a corresponding texel in the texture space. And, $a \sim i$ are constants.

$$x_s = \frac{au+bv+c}{gu+hv+i} \qquad y_s = \frac{du+ev+f}{gu+hv+i} \tag{1}$$

All the triangles composing a 3D object are decomposed into spans in a span rasterizer for screen viewing. Span means a set of consecutive horizontal pixels resided in a triangle. By mapping, a span is mapped into a random-directed straight line on a texture image as shown in Figure 1. This line conservation property is well explained in the following. When an arbitrary line in the screen space, $y_s = Ax_s + B$, is mapped into texture image space, a corresponding line, $v = A'u + B'$, is obtained by substituting x_s and y_s in (1) and rearrangement.

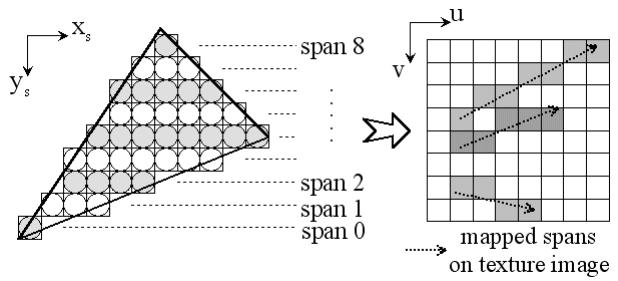


Figure 1. Mapping of triangle spans

2.2. Texture Cache & Triangle Span

The effectiveness of cache memory depends on locality of reference in data accesses. Both spatial and temporal localities are present in texture mapping [2]. Mip-map filtering [8] increases spatial locality in texture access since the level of the map is selected to closely match the level-of-detail that is being drawn on the screen. That is, due to the Mip-map filtering, one pixel movement in screen space is nearly mapped to one texel movement in texture space. One texture image can be mapped to several polygons of single frame or consecutive frames. Therefore, temporal locality in texel access is also present. In the cases of bilinear or trilinear filtering, multiple texels are needed for single pixel. It also contributes to temporal locality because some of the multiple texels for a pixel are apt to overlap with some texels for neighboring pixels. Due to the locality of texture, we can use a texture cache to improve system performance and to save the required bandwidth in system bus.

A rendered scene can be decomposed into a set of spans. Cache replacement can be divided based on the triangle span. If cache replacement occurs between texels residing in the same span, we call it *intra-span replacement*. If replacement occurs between texels of two different spans, we call it *inter-span replacement*. In cache operation, there is one cache replacement when one cache miss occurs excluding cold miss. So, cache miss can be calculated by counting cache replacement occurring in the cache. That is, total cache miss in a texture cache for rendering a textured scene can be denoted as the following :

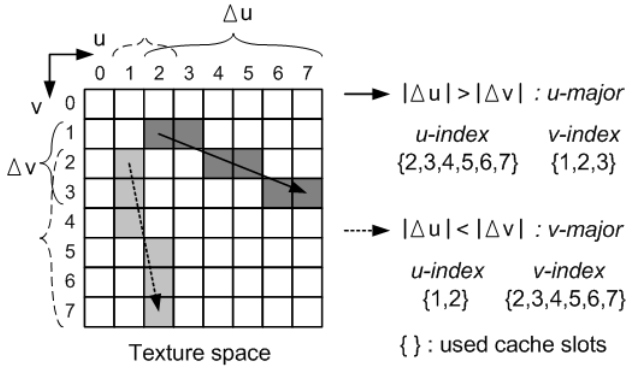
$$Total\ cache\ miss = cold\ miss + R_{intra} + R_{inter} \tag{2}$$
$$R_{intra} : intra-span\ replacement$$
$$R_{inter} : inter-span\ replacement$$

3. Adaptive Selection of an Index

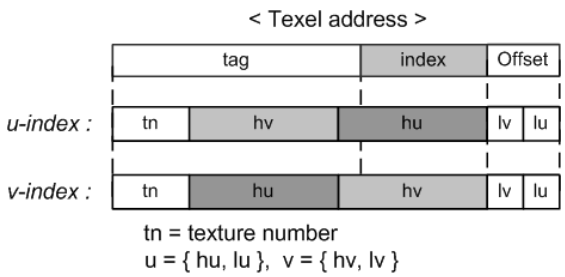
3.1. Basic Idea

In a cache, address for cached data is divided into three parts (tag, index, and offset) as shown in Figure 2. And, LSBs of an address have been used for an index traditionally. Meanwhile, 2D data for a texture image are loaded in a horizontal line-scan order into memory and the address of a texel consists of concatenation of a vertical(v) coordinate (MSBs) and a horizontal(u) coordinate (LSBs). Therefore, only the u-coordinate of a texel has been used for a cache index. However, texel access pattern may be any direction. Therefore, if we use u- or v-coordinate value adaptively for a cache index according to the direction of texel accesses, then we will obtain a more efficient cache index than using only the u-coordinate value for a cache index. In the texture mapping, texel-access direction is dependent on the rasterization order and many rasterization algorithms are based on the horizontal span rasterizer. Therefore, a polygon is decomposed into a set of horizontal line by a horizontal scan rasterizer. Besides, the horizontal line is also mapped to a line on a texture image. In this mapped line, if $|\Delta u|$ is greater than $|\Delta v|$, we call it *u-major*. Otherwise, we call it *v-major* (shown in Figure 2(a)). If we get the access direction information adaptively, we can use the information to

select adaptively the index bits from *u*-coordinates or *v*-coordinates. When the mapped direction on a texture image is *u-major*, we can obtain a more effective cache index if we use *u-index*, using u-coordinate as a cache index (shown in Figure 2(b)). That is, we can use wider range of cache slots in this case. But, if we use *v-index*, using a v-coordinate as a cache index, texels fetched from memory will reside in fewer cache slots. Therefore, the cache index will be ineffective and conflict miss will be increased in this case. Oppositely, if we use *v-index* for the case of *v-major*, we can obtain more effective cache indexing and cache miss reduction. That is, wider cache slots will be used for storing the texels resided on the dotted line.



(a) Texel access direction and *u/v-major*



(b) Texel address and *u/v-index*

Figure 2. Adaptive selection of an index

To do the above operation, it is need to select *u-major* or *v-major* adaptively. To decide whether texel-access direction is *u-major* or *v-major*, we use history of texture sampling points. That is, u/v-coordinates of the previous texture sampling point are stored. For the decision, coordinates of current and previous texture sampling points are compared. That is, the decision is obtained from moving direction of a texture sampling point. The decision is made in the time of address generation and it is used to select an index in the texture cache.

3.2. Hardware Design

To prove our idea, we designed a texture mapping hardware including a texture cache by using Verilog-HDL. From simulations, we compared cache miss count and total cycles of the proposed indexing method to those of the conventional method. We employed the values of the papers [2, 3] for important design parameters such as line size, way number, and cache size. Cache size is selected to 16 KB. Line size is selected to 64 bytes (4×4 texels) according to the paper by Hakura [2]. In the paper, he presented that square regions of texture are most effective in exploiting spatial locality and a 4×4 block has the lowest miss rate. The greater block size has the large miss penalty and causes the working set size to become unnecessarily large, leading to many capacity misses. Contrarily, the smaller block size has less spatial locality and causes high miss rate. Also, he presented that conflicts between blocks at adjacent levels of the Mip-map can be successfully eliminated with two-way set associative cache. Therefore, we use a two-way set associativity in texture cache design. As shown in Figure 3, texture unit consists of five pipeline stages: Fetch, AddrGen, TexelRead, Filter, and Blend.

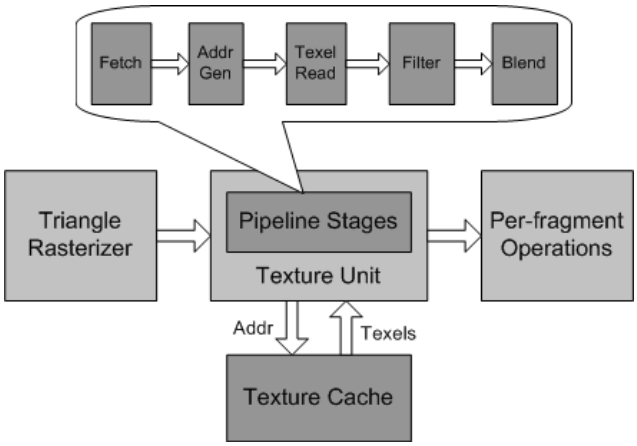


Figure 3. Texture mapping H/W

The texture cache ought to be modified to support our idea. It should be marked whether a cached data uses *u-index* or *v-index*. Therefore, it is necessary to add a 1-bit register per cache line for distinguishing *u-index*/*v-index* in the cache write time. In addition, hit check operation should be modified in order not to commit hit/miss decision errors. In a conventional cache, hit check operation is done by tag comparison of the cache line only in a matched index. For our method, however, tag comparison is needed twice as often as in the conventional cache because hit may occur in one of two cache lines of *u-index* and *v-index*. In the cache read time, only when the index of the hit cache line coincides with the index of the distinguishing register, final hit occurs. The modified part in a cache for the *A-index* is depicted in hard lines in Figure 4.

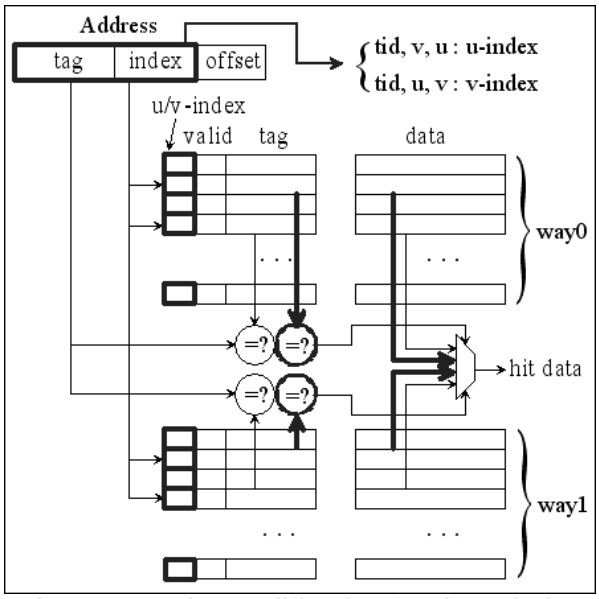
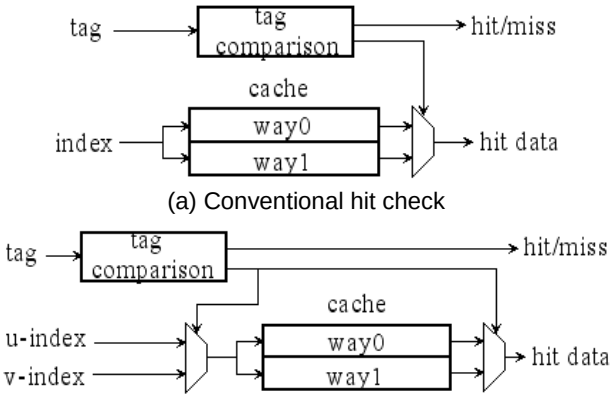


Figure 4. Cache modification for the *A-index*



(a) Conventional hit check

(b) Modified hit check for the *A-index*

Figure 5. Hit check data-path

In Figure 4, hit data is selected from four slots. Does it mean that each way has two read ports? The answer is no. The reason is that we can fix a cache index before a read request by pre-comparing. As shown in Figure 5, tag comparison and cache memory read operations are processed in parallel in the conventional hit check. However, the two operations are processed sequentially for the pre-comparing during a single cycle in the case of the modified hit check. That is, tag comparison is done to select cache index first, then the cache line of the selected index is read. This will extend logic depth for hit check. However, it is not a problem since the logic is not in the critical-path. The critical-path is present in the Filter or Blend stage of the texture unit pipeline since a GPU is running in much lower clock frequency than a CPU. Logic depth of data-path in a GPU becomes much longer than in a CPU. Meanwhile, the direction decision is processed by simple comparison logic in the AddrGen stage shown in Figure 3. Hardware overhead for the *A-index* is composed of two parts: one is data-path for direction decision (1,500 gates) and the other is additional logic for the cache modification (18,500 gates). Additional logics for the two parts are about 20,000 gates. It is under 10 % of the whole texture mapping unit excluding the 16 KB texture cache which is about 220,000 gates when it is approximately converted into gate count.

4. Simulations

We prove that the *A-index* has benefits in terms of performance and bus bandwidth. From HDL simulations using the hardware model explained in the above section with some test scenes, we compared *u-index*, *v-index*, and *A-index* in terms of cache miss counts and total cycles. Rasterized fragments of test scenes, inputs for Verilog simulations, were obtained from GATE [9], the graphics architecture simulator programmed in C-language which models overall graphics hardware architecture through a modular approach and supports OpenGL. In addition, we show that performance estimation of a cache is possible by the relation between *intra-span replacement* and cache miss. And, we compared estimation errors in the case of using the *u-index* and the *A-index*.

4.1. Performance Improvement

We simulated the HDL model and measured cache miss counts and total clock cycles with three test scenes shown in Figure 6. Size of the scenes is 640×480 pixels. Table 1 and 2 show the results. From the simulation results, we can see that the *A-index* has smaller cache miss and total cycles. Compared to the conventional index, *u-index*, the *A-index* has improved performance about 21.6 % in cache miss and 8.8 % in total cycles. It is due to the usage of wider cache slots obtained by changing the cache index adaptively.

(a) Scene 1 (b) Scene 2 (c) Scene 3

Figure 6. Quake3 demo scenes

Table 1. Cache miss

	Cache miss		
	u-index	*v-index*	*A-index*
Scene 1	56765	55739	44364
Scene 2	39747	38780	33263
Scene 3	50018	36493	37284
Average	48843	43671	38304

Table 2. Total cycles

	Total cycles		
	u-index	*v-index*	*A-index*
Scene 1	2461193	2548911	2210866
Scene 2	2165945	2243072	2015042
Scene 3	2793268	2604522	2539512
Average	2473469	2465502	2255140

We also demonstrate an influence of our idea on the cache size. It is summarized in Figure 7. From the results, we can see that it is possible to maintain or increase cache performance with only a half-sized cache if we use the *A-index*. As the cache becomes large, we get less gain. It is due to the increased number of cache slots. As the number of cache slots increases, occurrence of conflict miss is reduced because increased index bits scatter cached data into wider cache slots.

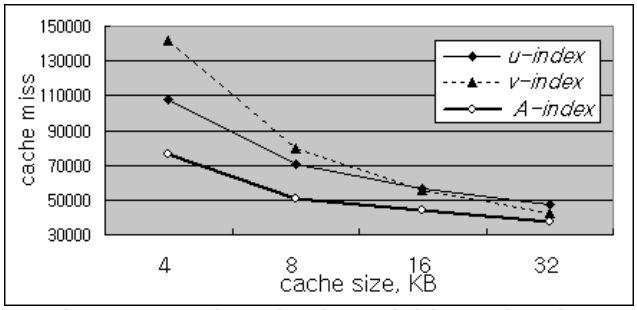


Figure 7. Cache miss in variable cache size

4.2. Performance Estimation

Miss-rate of a cache is one of the most important factors in performance of the system using the cache. It is not easy to estimate performance or cache miss of a texture cache because a texture cache stores image data which is deeply dependent on a rendering scene. Generally, it is necessary to simulate the whole cache operations for a few tens or hundred of test scenes to design a cache satisfying the required cache miss. However, we can save the simulation time if we can estimate cache miss by simple calculations. Cache miss can be split into three parts based on triangle spans as shown in (2). Here, we focus on the *intra-span replacement* which gives us a clue to estimate cache miss. It is possible to estimate cache miss by calculating the *intra-span replacement*, R_{intra}, given by the following:

$$\begin{aligned} R_{intra} &= \sum_{i=1}^{K} r_i, \quad i=1,2,\cdots,K \\ &= \sum_{i=1}^{K} \left\langle l_s - S_{eff} \right\rangle, \quad S_{eff} = w \cdot S_{phy} \cdot HE_i \\ &= \sum_{i=1}^{K} \left\langle l_s - w \cdot S_{phy} \cdot HE_i \right\rangle, \quad \langle x \rangle \equiv \max(0, x) \end{aligned} \tag{3}$$

The *intra-span replacement* can be split into *intra-span replacements per span*, r_i's. K is the number of spans, l_s and w mean span length and the number of ways. S_{eff} and S_{phy} are the number of effective cache slots for a given span and physical cache slots per way. *HE* is a hashing efficiency of a span and can be modeled by *the length of the touched blocks projected on the U or V coordinate* divided by *the number of touched blocks*. *HE* has a value of 0~1 and means efficiency of the used indexing method. Cache replacement is needed when conflict occurred. Conflict occurring in a single span is explained well in Figure 8. When span-length exceeds effective cache slots, intra-span replacement occurs. If not, there is no intra-span replacement. By (3), we can calculate the number of *intra-span replacement* for a given scene.

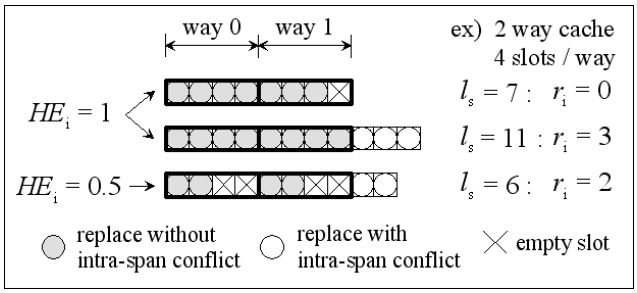
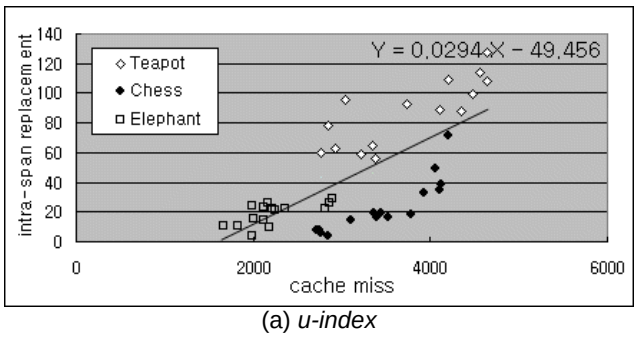


Figure 8. Intra-span conflicts

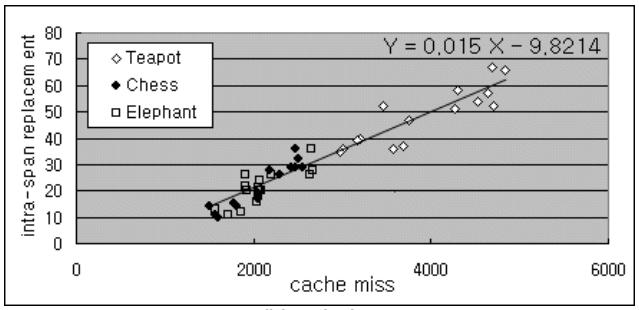
(a) Teapot (b) Chess (c) Elephant

Figure 9. Test models

We investigated the relation of *intra-span replacement* and cache miss obtained from HDL simulations on three test models (shown in Figure 9) which have different texel-access directions. We obtained data of 15 sample points from 15 different camera positions for each test model. The results are shown in Figure 10. By 45 points from the three models, we obtained estimated linear equations with the smallest RMS error in the cases of using the *u-index* and the *A-index* respectively.



(a) *u-index*



(b) *A-index*

Figure 10. Estimated linear equations

We estimated the number of cache misses from the estimated linear equations and measured estimation errors about the three test models respectively. It is shown in Table 3. From the results, we figure that it is reasonable to approximate cache miss from *intra-span replacement*. Especially, the *A-index* has smaller deviation in the number of *intra-span replacements* than the *u-index* according to texel-access direction. So, we can estimate cache miss with smaller error by using the *A-index* rather than the *u-index*, a conventional indexing method. In the Teapot model, the *u-index* has a little smaller cache miss than the *A-index* because the Teapot model is *u-major* dominant. In other models, however, the *A-index* has smaller cache miss. In the *u-index*, a projected angle of a mapped span into a coordinate axis is 0 ~ 90° because the projection is applied to only the *u*-axis, However, the angle is bounded 0 ~ 45° in the *A-index* because the projection can be applied to *u*-axis or *v*-axis. Therefore, the value of the *HE* has smaller deviation in the *A-index* than in the *u-index*. As a result, the *A-index* has smaller cache miss and smaller estimation error than the *u-index*.

Table 3. Estimation errors

		u-index	*A-index*
Teapot 1604 triangles, 12.0 pixs/triangle, *u-major dominant*	Avg. $ miss	3748.8	3919.2
	Est. $ miss	4639.1	3885.9
	RMS error	999.4	348.5
	%	**26.7**	**8.9**
Chess 8310 triangles, 5.6 pixs/triangle, *v-major dominant*	Avg. $ miss	3474.9	2090.3
	Est. $ miss	2487.2	2108.1
	RMS error	1035.2	243.0
	%	**29.8**	**11.6**
Elephant 4063 triangles, 18.1 pixs/triangle, -	Avg. $ miss	2229.2	2091.2
	Est. $ miss	2317.1	2081.4
	RMS error	282.3	254.6
	%	**12.7**	**12.2**

5. Conclusions

In this paper, we have proposed *A-index*, a method to reduce memory bandwidth required for fetching texture image data by reducing cache miss through adaptive selection of an index according to the mapped direction of a span. To prove our idea, we designed a texture mapping hardware model including a texture cache. From cycle-accurate HDL simulations, we examined the number of cache misses and the number of total cycles for rendering some test scenes. In the case of using the *A-index*, cache miss and total cycles were saved by 21.6 % and 8.8 % respectively. Also, it is possible to maintain or increase cache performance with only a half-sized cache if the *A-index* is used. By the decomposition of cache miss based on triangle spans, we showed that it is possible to estimate the number of cache misses in a texture cache by calculating *intra-span replacement*. In addition, we found that we can estimate cache miss more precisely with small errors if the *A-index* is used.

Acknowledgment

This work was supported by SystemIC 2010 Project and SAMSUNG Electronics.

References

[1] Paul S. Heckbert, “Survey of Texture Mapping”, IEEE Computer Graphics and Applications November 1986;56-67.

[2] Ziyad S. Hakura and Anoop Gupta, “The Design and Analysis of a Cache Architecture for Texture Mapping”, Proc. of the 24th International Symposium on Computer Architecture May 1997;108-119.

[3] H. Igehy, M. Eldridge, and K. Proudfoot. “Prefetching in a Texture Cache Architecture”, SIGGRAPH/Eurographics Workshop on Graphics Hardware, 1998.

[4] M. Cox, N. Bhandari, and M. Shantz, “Multi-level Texture Caching for 3D Graphics Hardware”, Proc. of the 25th International Symposium on Computer Architecture, 1998.

[5] Andreas Schilling, Gunter Knittel, and Wolfgang Straβer, “Texram: A Smart Memory for Texturing”, IEEE Computer Graphics and Applications, 16(3), pp. 32-41, May 1996.

[6] Tony Givargis, “Improved Indexing for Cache Miss Reduction in Embedded Systems”, Design Automation Conf., pp. 875-880, June 2003.

[7] Alan Watt, 3D Computer Graphics, Addison-Wesley Publishing Company, Second edition, 2000.

[8] Lance Williams, “Pyramidal Parametrics”, Computer Graphics (Proc. SIGGRAPH 83), Vol. 17, No. 3, pp.1-11, July 1983.

[9] Inho Lee, et al., “A Hardware-like High-level Language Based Environment for 3D Graphics Architecture Exploration”, ISCAS 2003, Tailland, May, 2003.