IEICE Electronics Express, Vol.8, No.13, 1064–1070

The design of a texture mapping unit with effective MIP-map level selection for real-time ray tracing

Woo-Chan Park[1a)], **Dong-Seok Kim**[1], **Jeong-Soo Park**[2], **Sang-Duk Kim**[2], **Hong-Sik Kim**[3], **and Tack-Don Han**[2]

[1] *Department of Computer Engineering, Sejong University*
98 Gunja-dong, Gwangjin-gu, Seoul
[2] *Department of Computer Science, Yonsei University*
134 Shinchon-dong, Seodeamun-gu, Seoul 120–749, Korea
[3] *Advanced Design Team, R&D Division, Hynix Semiconductor Inc.,*
San 136–1 Ami-ri Bubal-eub, Icheon-si, Gyeonggi-do 467–701, Korea
a) *pwchan@sejong.ac.kr*

Abstract: We propose effective texture-mapping hardware for real-time ray tracing. Therefore, we introduce a novel method to select the MIP-map level of texture images, which requires only the total length of the intersected ray and the pre-calculated value. The proposed architecture can support the texture MIP-mapping by integrating simple hardware logic in existing ray-tracing hardware.

Keywords: graphics processor, ray tracing, texture mapping
Classification: Integrated circuits

References

[1] J. P. Ewins, M. D. Waller, M. White, and P. F. Lister, "MIP-Map Level Selection for Texture Mapping," *IEEE Trans. Vis. Comput. Graphics*, vol. 4, no. 4, pp. 317–329, Dec. 1998.
[2] L. Gritz and J. Hahn, "BMRT: A Global Illumination Implementation of the RenderMan Standard," *J. Graphics Tools*, vol. 1, no. 3, pp. 29–47, 1996.
[3] H. Igehy, "Tracing Ray Differentials," *Proc. SIGGRAPH*, pp. 179–186, 1999.
[4] P. H. Christensen, J. Fong, D. M. Laur, and D. Batali, "Ray Tracing for the Movie 'Cars'," *Proc. IEEE Symp. Interactive Ray Tracing*, pp. 1–6, Sept. 2006.
[5] W. C. Park, J. H. Nah, J. S. Park, K. H. Lee, D. S. Kim, S. D. Kim, J. H. Park, Y. S. Kang, S. B. Yang, and T. D. Han, "An FPGA Implementation of Whitted-style Ray Tracing Accelerator," *Proc. IEEE Symp. Interactive Ray Tracing*, p. 187, 2008.
[6] P. Heckbert, "Texture Mapping Polygons in Perspective," *New York Institute Technol. Tech. Memo 13*, April 1983.

DOI: 10.1587/elex.8.1064
Received May 18, 2011
Accepted June 14, 2011
Published July 10, 2011

IEICE Electronics Express, Vol.8, No.13, 1064–1070

1 Introduction

Most contemporary graphics processing units (GPUs) are based on a z-buffer algorithm. For realistic rendering, ray tracing algorithms have typically been used to support global illumination effects by simulating the physical features of lights and rays. However, the huge computation cost is a major drawback. Recently, many acceleration techniques for real-time ray tracing have been announced.

Texture mapping is a fundamental feature that generates high-quality computer graphics images. In current GPUs, MIP-mapping is a common technique to reduce texture aliasing. In [1], various practical methods for MIP-map level selection were investigated within the context of the rasterization technique used in OpenGL. Several MIP-map level selection methods using a ray tracing algorithm and based on the ray differential have been proposed in [2, 3]. These were developed for production (off-line) renderers, such as Pixar's RenderMan [4].

In this paper, we propose a texture-mapping hardware architecture for real-time ray tracing. Therefore, we provide an effective algorithm for fast MIP-map level selection that calculates each MIP-map level based only on the total ray length of the intersected ray and a pre-calculated value stored in the intersected triangle itself. This algorithm can provide MIP-map level selection using simple hardware logic. We also implement the proposed architecture by integrating it into the existing ray tracing hardware in [5]. Experimental results show our approach significantly reduces the computation requirements compared to [2].

2 The traditional MIP-Map selection algorithm

In current GPUs, MIP-mapping with bi-linear or tri-linear filtering is commonly used to reduce spatial aliasing artifacts [1]. A MIP-map is a pre-filtered multi-resolution image pyramid. The highest resolution image is level 0 at the base of the MIP-map pyramid. The next level is generated by averaging groups of four neighboring texels with a level 0 texture image.

Several texels may map to one screen-space pixel. The ratio of this texture space to screen space scaling is called texture minification [1]. Using this per pixel-based texture minification ratio, the MIP-map level can be easily calculated. If the MIP-map level is too high, the image may be excessively blurred; if it is too low, the aliasing artifacts would be noticeable. The following equation has been chosen as the best solution from among several different methods in [6]. In

$$lod = \log_2(\max(|du|, |dv|)), \tag{1}$$

du and dv are the partial derivatives of a texture coordinate u and v with respect to a screen coordinate x and y. These partial derivatives are approximated using the longer of the two edges of the projected parallelogram of the texture space.

DOI: 10.1587/elex.8.1064
Received May 18, 2011
Accepted June 14, 2011
Published July 10, 2011

Another technique to determine the texture minification ratio, called pixel clipping, has been discussed in [1]. This technique uses an area-based approach and involves clipping the texture-mapped polygon. When the projected quadrilateral or parallelogram is thin, the rendered image of the former is somewhat more blurred than that of the latter.

3 Previous MIP-Map selection algorithms for ray tracing

Conventional ray tracing consists of the following stages. First, a primary ray is generated, originating from the eye and passing through each pixel. Second, an acceleration structure, such as a kd-tree, is traversed until the ray encounters a leaf node containing triangles. Third is the intersection test stage, which tests ray intersections with all the triangles in the corresponding leaf node. Finally, if there is any intersection, the shading stage calculates the color, including texture mapping, on the ray-triangle hit point. If shadows or secondary rays are required, new rays are generated and then passed back to the second stage.

Ray differential-based techniques have been provided in [2, 3] to select the proper MIP-map level for production (off-line) renderers, such as Pixar's RenderMan. A ray differential is defined as the difference between a ray and its neighboring rays [3]. The differentials are monitored as the rays are propagated. After calculating du and dv by approximation using the ray differentials, a MIP-map level is determined according to equation (1).

The ray differential method described in [3] can be used to calculate approximated du and dv on curved surfaces. For example, it can keep track of the differentials of the reflected ray even when it hits a highly curved surface. Because it requires many operations per ray, it is unsuitable for real-time ray tracing.

Similar to the rasterization algorithm, BMRT [2] determine du and dv by calculating the projection of texture coordinates onto the image plane. Suppose $\boldsymbol{P}_l$ is the total distance from the camera position to the sample point $\boldsymbol{P}$, $\boldsymbol{d}_s$ is the distance between screen space samples, and $\boldsymbol{d}_{est}$ is the world space distance between screen samples at point $\boldsymbol{P}$. The $\boldsymbol{d}_{est}$ can be easily calculated by multiplying $\boldsymbol{P}_l$ and $\boldsymbol{d}_s$. Using the result of that calculation, BMRT calculate the estimates of du and dv by dividing $\boldsymbol{d}_{est}$ with $|\boldsymbol{dPdu}|$ and $\boldsymbol{d}_{est}$ with $|\boldsymbol{dPdv}|$, respectively. Here, $\boldsymbol{dPdu}$ and $\boldsymbol{dPdv}$ are the partial derivatives of the surface. Even though this algorithm is simple to implement, its computational complexity is still too great for real-time processing.

4 Proposed algorithm

The basic idea behind the proposed algorithm is as follows. Initially, we calculate P_b to represent a base texture position—the relative distance from the camera position where the areas of a pixel and a texel are equal. It is view-independent because it is a relative distance from the camera position, so that it does not need to be re-calculated unless triangle coordinates are changed.

DOI: 10.1587/elex.8.1064
Received May 18, 2011
Accepted June 14, 2011
Published July 10, 2011

We assume the intersected triangle is perpendicular to the view vector; the corresponding MIP-map level can then be easily calculated using the following equation:

$$lod = \log_2\left(\frac{P_l}{P_b}\right) = \log_2(P_l) - \log_2(P_b). \quad (2)$$

In the pre-processing stage, $log(P_b)$ is calculated for each texture-mapped triangle. Processing is performed only once in the static scene, but is performed for each frame whenever coordinates are changed in the dynamic scene. After the pre-processing stage, only the subtraction operation after the *log* operation is required to complete equation (2).

The process to calculate $log(P_b)$ consists of the following five steps. The first step is to determine the number of texels within the triangle with respect to the texture space. This can be calculated by multiplying the triangle size (or area) of the texture space with the texture resolution ($RES_{texture}$), as shown in the following equation. In

$$T_{XN} = \frac{((s_0 \cdot t_1) + (s_1 \cdot t_2) + (s_2 \cdot t_0) - (t_0 \cdot s_1) - (t_1 \cdot s_2) - (t_2 \cdot s_0))}{2} \cdot RES_{texture},$$

(s_0, t_0), (s_1, t_1), and (s_2, t_2) are the three texture coordinates for each vertex.

In the second step, we calculate the ratio of texture resolution to screen resolution:

$$X_{PR} = \frac{RES_{texture}}{RES_{screen}}.$$

In the third step, we calculate the number of texels within the triangle with respect to the screen space:

$$T_{XS} = T_{XN} \cdot X_{PR}.$$

In the fourth step, we calculate the triangle size:

$$(x_t, y_t, z_t) = \{(x_1, y_1, z_1) - (x_0, y_0, z_0)\} \times \{(x_2, y_2, z_2) - (x_0, y_0, z_0)\},$$

$$T_{size} = \frac{1}{2}\sqrt{x_t^2 + y_t^2 + z_t^2},$$

where (x_0, y_0, z_0), (x_1, y_1, z_1), and (x_2, y_2, z_2) are the three vertices coordinates of the triangle.

The value of P_b can be determined by the texel and pixel resolution, which are produced in the third and fourth steps, respectively. Thus, the final step determines $log(P_b)$ based on T_{XS} and T_{size}. T_{size} is a constant value for a scene, whereas T_{XS}s is increased to the second power as the distance between the camera position and the given triangle increases. Thus, $log(P_b)$ can be calculated as follows:

$$log_2(P_b) = log_2\left(\sqrt{\frac{T_{XS}}{T_{size}}}\right) = \frac{1}{2} log_2\left(\frac{T_{XS}}{T_{size}}\right).$$

DOI: 10.1587/elex.8.1064
Received May 18, 2011
Accepted June 14, 2011
Published July 10, 2011

5 Proposed hardware architecture

Figure 1 shows the proposed hardware architecture for texture mapping in the shading stage. As a result of the intersection test stage, *triangle_id*, representing the triangle identifier of the intersected triangle, *ray_length*, representing the distance between a ray origin and a ray-triangle hit point, and *texture_id*, representing the texture identifier of the intersected triangle, are generated and then input into the texture mapping unit. In the pre-processing step, $log(P_b)$ is calculated for each texture-mapped triangle, which is stored in the triangle cache with other triangle information.

As for the processing flow, initially, $log(P_b)$ of the intersected triangle is accessed from the triangle cache. At the same time, a new $\boldsymbol{P}_l$ is calculated by adding *ray_length* and the previously stored $\boldsymbol{P}_l$ to the ray stack. With these two result values, we calculate the MIP-map level according to equation (2). Finally, appropriate texture data corresponding to the MIP-map level is accessed from the texture cache, at which point filtering is performed to generate a texture-mapped color.

Because the number of secondary rays for a pixel varies according to the number of ray bounces, the ray stack is essential to support ray tracing. In traditional ray tracing, two rays (a reflection and a refraction ray) at most can be generated for a secondary ray. If both rays are generated, one is sent to the following stage and the other is stored in the ray stack. The current $\boldsymbol{P}_l$ is included in the ray information stored in the ray stack. Thus, recently accumulated $\boldsymbol{P}_l$ can be maintained by the ray stack.

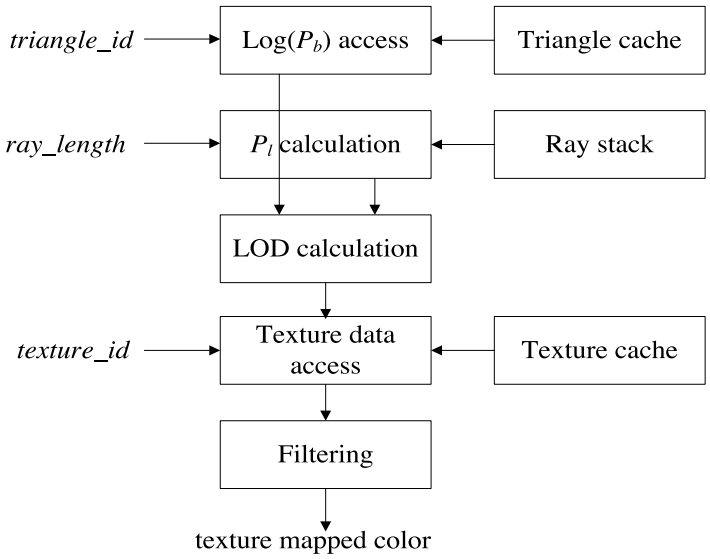

Fig. 1. The proposed hardware architecture for texture mapping

6 Experimental results

To evaluate the performance, we selected three scenes from well-known BART benchmarks: *kitchen*, *robot*, and *museum*. We performed various simulations,

DOI: 10.1587/elex.8.1064
Received May 18, 2011
Accepted June 14, 2011
Published July 10, 2011

as shown in Table I. For each scene, we estimated the distributions of MIP-map level selections. As a result, all scenes tend to select higher levels.

We also evaluated the computation requirements (for each floating point addition, multiplication, division, square root, and log operation) to calculate the MIP-map level of the proposed scheme and [2]. For our scheme, the computational requirements were estimated separately; P_b was calculated in the pre-processing step and the case that it is in ray tracing itself. In the latter case, note that (x_t, y_t, z_t) in the fourth step in calculating P_b is not considered because it is calculated in the ray-triangle intersection test stage for ray tracing. We also did not consider the resultant value of the second step because it is constant during rendering. As seen in Table I, the evaluation results show that our scheme outperforms [2].

We also integrated the proposed architecture into the ray tracing hardware of [5] and then implemented all on a Xilinx Virtex 5 LX330 chip with 84 MHz core speed. Our design occupies approximately 88% of the FPGA's logics cells and 78% of the FPGA's memory resources. The screen resolution is 800 by 480. The secondary rays can be generated recursively up to 10 bounces. Figure 2 shows two *kitchen* scene images. Some aliasing artifacts are noticeable on the floor in the left image, but are significantly reduced in the right image due to the proposed MIP-map level selection.

The texture cache hit rate and FPS of each benchmark are measured directly from running hardware and are shown in Table I. The texture cache hit rate is significantly increased due to the MIP-map level selection. Also, FPS is gracefully increased when MIP-map is on compared to when it is off.

Fig. 2. *Kitchen* scene images: MIP-map off (left) and MIP-map on (right)

Table I. Experimental Results

Name	MIP-Map selection rates (%) (level 0, 1, 2, 3)	Computation requirements to calculate a MIP-Map level (*add*, *mul*, *div*, *sqrt*, *log*)			Texture cache hit rates(%)		FPS	
		[2]	ours		[5]	ours	off	on
			pre-process	ray trace				
kitchen	13, 30, 9,4 8	61, 60, 1, 2,1	1, 0, 0, 0,1	12, 13, 2, 2, 2	74.5	93.7	3.4	3.7
robot	2, 1, 2, 95				73.9	98.6	5.1	6.3
museum	0, 58, 26, 16				79.6	97.2	6.1	6.4

DOI: 10.1587/elex.8.1064
Received May 18, 2011
Accepted June 14, 2011
Published July 10, 2011

7 Conclusion

In this paper, we proposed effective texture-mapping hardware for ray tracing with simple logic for MIP-map level selection. According to the simulation and implementation results, the proposed algorithm and hardware architecture can provide accurate MIP-map level selection and considerably reduces computational complexity compared to the previous algorithm.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education, Science, and Technology (2011-0002712).

DOI: 10.1587/elex.8.1064
Received May 18, 2011
Accepted June 14, 2011
Published July 10, 2011