A Cost-Effective VLSI Architecture for Anisotropic Texture Filtering in Limited Memory Bandwidth

Hyun-Chul Shin, Jin-Aeon Lee, and Lee-Sup Kim, *Senior Member, IEEE*

***Abstract*—Texture mapping is one of the techniques that express realism in three-dimensional (3-D) graphics. To produce high-quality images, various anisotropic filtering methods have been proposed for texture mapping. These methods require more texels than isotropic (trilinear) filtering method. In spite of increases to texture memory bandwidth, however, texture memory bandwidth is still a bottleneck of texture-filtering hardware. Consequently, an exact filtering method is required for good-quality images in a limited texture memory bandwidth. In this paper, we propose anisotropic texture filtering based on edge functions. Our method proposes an exact footprint-shape approximation with edge functions for generating weights. For real-time filtering, the weight plays a key role in effective filtering of the restricted texels loaded from memory. The normalized value of the edge function gives the distance relative to the contribution of texels to a final intensity. Calculating a Gaussian filter using this normalized value, generates a good weight. The quality of rendered images is superior to other anisotropic filtering methods with the same restricted number of texels. For images of the same quality, our method requires less than half the texels of other methods. Consequently, the improvement in performance is more than twice that of other methods. With low hardware overheads, our method can be implemented at a reasonable cost. In practice, the algorithm is demonstrated through VLSI implementation. The hardware, which is described by verilog and synthesized with a 0.35-μm 3.3-V standard cell library, is operated at 100 MHz and it generates 100 M texture-filtered RGB pixel-color values per second.**

***Index Terms*—Anti-aliasing, computer graphics, costs, filtering, performance.**

I. Introduction

TEXTURE mapping is a technique that effectively improves the realism of computer-generated scenes in three-dimensional (3-D) graphics. Trilinear filtering has been popular because of its simplicity and ease of hardware implementation. However, since this method approximates a quadrilateral footprint to a square footprint with one texel size, it is not suitable for filtering a long and narrow quadrilateral footprint to which a pixel is projected in texture space. The quality of the rendered image, therefore, deteriorates as the viewing angle changes from 0° to 90°.

As the desire for high-quality images increases, anisotropic texture filtering, which adjusts the shape and size of the footprint, has superceded trilinear filtering. For example, the NVIDIA GeForce 4 supports anisotropic texture filtering with anisotropy up to 8:1. Recently proposed methods, such as footprint assembly, fast elliptical lines, and fast footprint MIP-mapping generate good-quality images but still require many texels [7]. These methods show poor-quality images when the texels loaded from memory for real-time filtering are restricted. The restriction is due to the limitation of memory bandwidth and so on. For instance, the recent NVIDIA graphics card supplies up to eight texels per clock cycle at each texture pipeline through a cache. However, 64 texels are required if the anisotropy is up to 8:1.

In this paper, to exactly filter the limited texels loaded from memory, we propose edge-function-based anisotropic texture filtering. The edge function closely approximates a footprint shape and plays an important role in calculating good weights. The rendered images show the best quality of various anisotropic filtering methods using the same restricted number of texels. To produce images with the same quality, our method requires less than half the number of texels used in other methods. This economy of texels considerably improves performance.

In the next section, we discuss, in detail, previous works on anisotropic texture filtering. In Section III, we describe the algorithm of edge-function-based anisotropic texture filtering. In Section IV, we compare the image quality and the performance of proposed anisotropic filtering method with the image quality and performance of other filtering methods. In Section V, we describe the hardware architecture of the proposed filtering method and compare the cost of the hardware with that of other methods. Finally, we discuss the VLSI implementation of the proposed filtering method.

II. Previous Works

In this section, we first discuss elliptical weighted average filtering, followed by the footprint assembly and Feline methods. We then discuss fast footprint MIP-mapping in detail.

A. Elliptical Weighted Average (EWA)

As shown in Fig. 1, the elliptical weighted average (EWA) filtering, proposed by Heckbert, approximates a footprint as an ellipse by approximating a pixel with a circle [3]–[5], [12]. The ellipse approximates the projection of a circular Gaussian pixel filter from screen space to texture space. The normalized ellipse function denoted as $d^2(u, v)$ represents the distance squared from the center of the pixel when the texel position is mapped back to screen space. This value indicates the contribution of the texel to a final intensity and is used to index the Gaussian filter weight ROM. Texels whose d^2 is less than one were sampled,

Manuscript received May 27, 2003; revised August 12, 2005.

H.-C. Shin is with the SOC CT Group, LG Electronics Company, Seoul 137-724, Korea (e-mail: hcshinjc@lge.com; hcshinjc@gmail.com).

J.-A. Lee is with Samsung Electronics Company, Suwon 446–711, Korea (e-mail: jalee@samsung.com).

L.-S. Kim is with the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Taejon 305–701, Korea (e-mail: lskim@ee.kaist.ac.kr).

Digital Object Identifier 10.1109/TVLSI.2006.871761

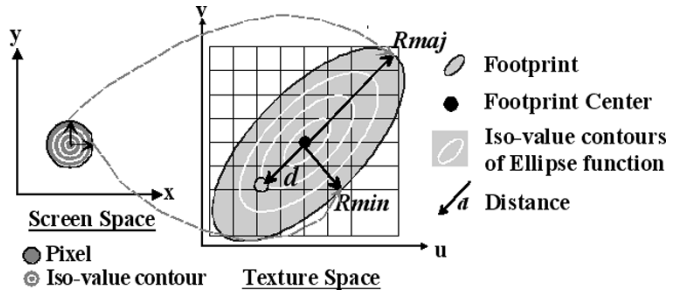

Fig. 1. Elliptical weighted average.

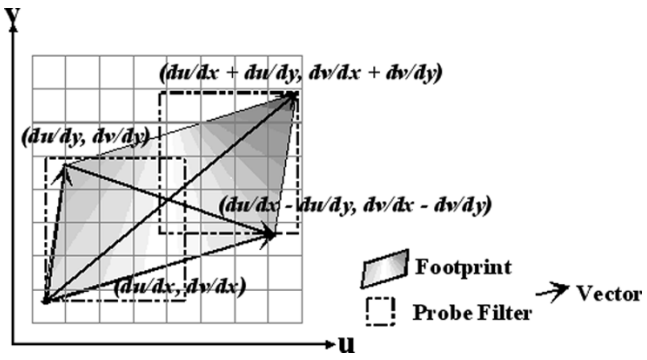

Fig. 2. Footprint assembly.

Gaussian weighted, and accumulated. The result was divided by the sum of the weights.

Although this method requires intensive computation power and texel values, it generates high-quality images, which provide a benchmark for various other filtering techniques.

B. Final Stage Footprint Assembly: TEXRAM

The footprint assembly (TEXRAM) method [10] approximates a footprint with a parallelogram formed by the two vectors, $(du/dx, dv/dx)$ and $(du/dy, dv/dy)$, as shown in Fig. 2. In this method rough samples are taken of the area inside the parallelogram by assembling trilinear filter probes along the major axis, which is the longer of the two vectors. The probe filter width is determined by the minor axis, which is approximated by choosing the shortest of the two side vectors and the diagonals $(du/dx + du/dy, dv/dx + dv/dy)$ and $(du/dx - du/dy, dv/dx - dv/dy)$. As shown in (1), the number of probes is given as the ratio of the major axis length to the minor axis

$$\text{Nprobes}_T = \frac{\text{Major axis length}}{\text{Minor axis length}}. \tag{1}$$

This value is rounded to the nearest power of two (2^n) and each probe is equally weighted. Using this method, high-quality images are generated even when the projection angle is nearly 90°. Although the hardware size is also amenable to implementation in VLSI, the intensive computation requires many texels [1]. In particular, for a highly distorted projection, this method requires more probes and the rendering performance severely deteriorates. When the number of loaded texels is limited, the method inevitably generates poor-quality images. This phenomenon is due to poor filtering caused by the adoption of trilinear filtering and an equal weighting of each probe.

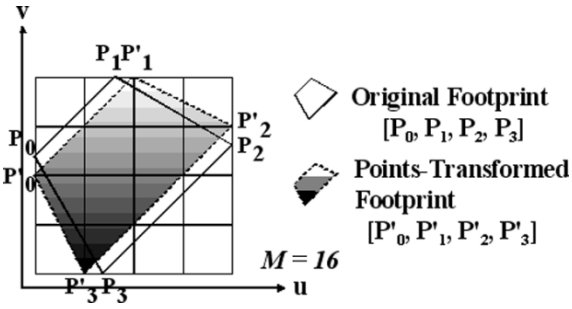

Fig. 3. Error of footprint shape.

C. Fast Elliptical Lines: Feline

The filtering of the fast elliptical lines (Feline) method is closer to the EWA filtering method because it compensates for the TEXRAM method [8]. The length of the major axis in the TEXRAM method is usually much shorter than the major diameter of an ellipse, which is the true sampling line length. Thus, the Feline method computes a more appropriate length for the major axis and it applies to the number of probes

$$\text{Nprobes}_F = 2 * \left(\frac{\text{Major axis length}}{\text{Minor axis length}}\right). \tag{2}$$

The Feline method replaces an equal weight of the TEXRAM method with a Gaussian weight [9]. However, although an image rendered by the Feline method exhibits far fewer aliasing artifacts than an image rendered by the TEXRAM method, the computation is more intensive, requiring twice the number of texels than the TEXRAM method.

D. Fast Footprint MIP-Mapping: FFPMM

The fast footprint MIP-mapping (FFPMM) method filters texels in a rectangle that covers a quadrilateral footprint with weights [6]. This method considers that the loaded texels are inevitably restricted due to constraints such as memory bandwidth. The number of texels that can be loaded from memory for real-time filtering is denoted as M. To load more texels that contribute to filtering at the given limit M, the FFPMM method determines the level of detail (LOD) by using the aspect ratio of a footprint. To filter the loaded texels efficiently, the weight of the area coverage is used. Therefore, the performance of the system and the quality of the images do not deteriorate severely if texels are restricted. The quality of the images compares with the images produced by the TEXRAM.

Nonetheless, weight generation has shortcomings. For real-time filtering, the weight is precalculated with the four corner points of the footprint. These points are transformed to integer positions before the precalculation and the change in the position of the four corner points cause an error in the footprint shape and weight, as shown in Fig. 3. Since the weight is multiplied by the texel at filtering, the error causes a degradation in the

quality of the rendered images. In terms of the image quality, an area coverage filter is inferior to a Gaussian filter. In addition, the size of the weight table increases rapidly as the limit M increases. The number of possible combinations for the positions of the four points determines the size of the weight table. Since the number of combinations increases rapidly as the limit M increases, it is too large to be implemented as real hardware.

III. Edge-Function-Based Anisotropic Texture Filtering

Texture mapping plays a key role in the overall performance of graphics hardware and the quality of rendered images. In spite of the texture cache, the limit of the memory bandwidth is still a bottleneck for texture mapping hardware. To maintain image quality and performance, texture filtering should be efficient enough to render an image of high quality using only the restricted texels loaded from memory for real-time filtering.

Efficient texture filtering should load more texels that contribute to filtering at the restricted number of texels. In addition, efficient filtering should post-filter the loaded texels with the weights. The post-filtering is done while taking into account the contribution of the texel to a final intensity, which is relative to the distance from the center of a pixel at screen space when the texel is mapped back to screen space [13]. For example, the ellipse function of EWA indicates this distance. The post-filtering plays a more important role in efficient filtering.

In this paper, we propose edge-function-based anisotropic texture filtering (EFATF). For correct post-filtering, EFATF uses the edge function representation of a footprint. The edge function approximates the footprint shape clearly. The normalized value of an edge function gives the distance from the center of the pixel at screen space when the texel is mapped back. Applying this distance to a good filter, such as a Gaussian filter, generates a good weight. To load more texels that contribute to filtering, EFATF uses an LOD selection scheme with the aspect ratio of a footprint. The LOD scheme of FFPMM is efficient.

The edge function approximation of a footprint and the weight generation from the edge function are now discussed.

A. Footprint Approximation Based on Edge Function

Other filtering methods poorly approximate the footprint. In the case of the TEXRAM and Feline methods, the number of probes $(\mathrm{Nprobes}_{T(F)})$ is limited in order to maintain the performance. To meet this limited number, the length of the minor axis is widened and the footprint is under sampled, that is, poorly approximated. In the FFPMM method, to prevent the size of the weight table from increasing dramatically, the four corner points of the footprint are inevitably restricted to integer positions, causing the footprint-shape approximation error that induces the degradation of the image quality.

To more accurately approximate the footprint shape that greatly affects the weight, our algorithm represents the edges of a footprint with edge functions as shown in Fig. 4.

The edge function is used in the anti-aliasing of polygon rendering that is one of the important algorithms of 3-D graphics [11]. Since the edge function represents the edge with the direction, it has a negative value on the left side of the edge and a positive value on the right side with respect to the direction of the edge, as shown in Fig. 5. That is, the sign of an edge function at an arbitrary point, determines whether the point is inside or outside the edge. The absolute value of an edge function at a point, represents the distance from the edge to the point.

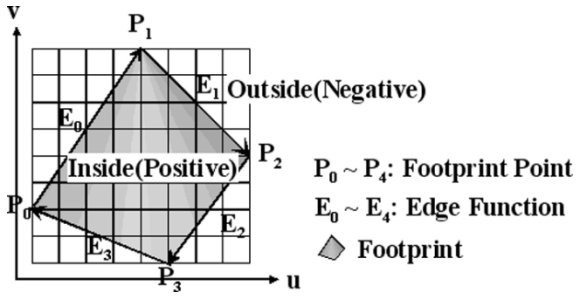

Fig. 4. Footprint representation with edge function.

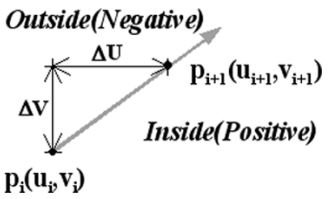

Fig. 5. Edge function representing the edge that directs from P_i to P_{i+1}.

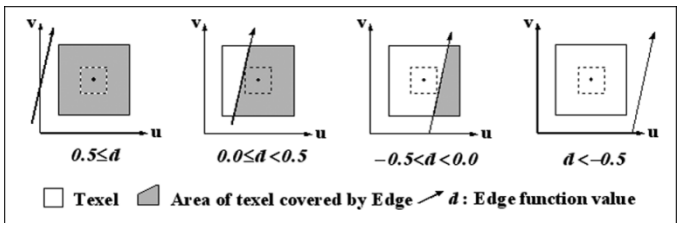

Fig. 6. Relation between an area of texel and an edge function.

The edge function is expressed as follows:

$$E_i(u,v) = (u_{i+1} - u_i) * \mathrm{deu}_i + (v_{i+1} - v_i) * \mathrm{dev}_i \qquad (3)$$

with $\mathrm{deu}_i * \Delta U_i + \mathrm{dev}_i * \Delta V_i = 0$.

The expressions of deu_i, dev_i, ΔU_i, and ΔV_i are as follows:

$$\mathrm{deu}_i = \frac{(\Delta V_i)}{(|\Delta U_i| + |\Delta V_i|)},\ \mathrm{dev}_i = \frac{(-\Delta U_i)}{(|\Delta U_i| + |\Delta V_i|)},$$
$$\Delta U_i = u_{i+1} - u_i \text{ and } \Delta V_i = v_{i+1} - v_i.$$

Both deu_i and dev_i are calculated using the formula of Manhattan distance instead of Euclidean distance. Manhattan distance is more useful in representing the area coverage of the texel because all edges that have a given distance from the center of the texel form a square. The texel that is covered by the positive, or inside plane, defined by the footprint edge, is determined by the edge function value as shown in Fig. 6. If the edge function value is less than -0.5, the texel is not covered by the inside plane defined by the footprint edge. Finally, the four edge function approximates the footprint shape clearer, by selecting the texels whose four edge functions are greater than -0.5, as

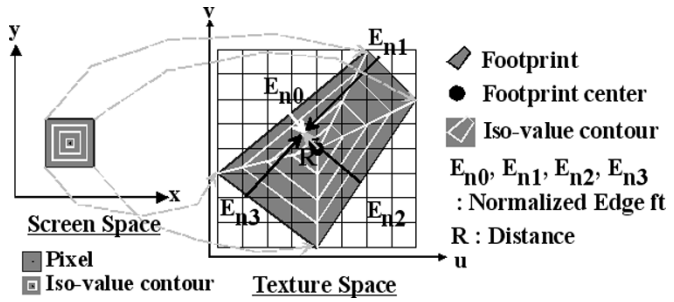

Fig. 7. Relation between the distance and the normalized edge function.

shown in Fig. 4. This edge function also plays a key role in the generation of filter weights through normalization.

Therefore, in contrast to the TEXRAM, Feline, and FFPMM methods, our scheme does not cause any degradation of the image quality due to the weight generation based on the poor approximation of the footprint shape.

B. Filter Weight Generation

For correct filtering, the relative contribution of each texel to a final intensity should be considered. The contribution is determined by the distance from the center of the pixel in screen space where the texel is mapped. However, the distance is not easily obtained and other methods do not consider the contribution of the texel. The cross-sectional shape of the filter is also important [4]. Theoretically, the ideal low-pass filter $\mathrm{sinc}(x)$ should be used, but it is impractical due to its infinite width-of-impulse response. In practice, a finite-impulse response (FIR) filter should be used, such as the box, triangle, cubic B-spline, and Gaussian filter.

In our method, we used a Gaussian filter, which is the best of the FIR filters. Other filters produce sharper images without introducing more aliasing artifacts. However, these filters have a radius of two or three pixels, which increases the work required to compute a textured pixel by a factor of four or nine. None of these are as mathematically tractable as the Gaussian filter [8].

A Gaussian filter is expressed as a function of the distance R, which is from the center of the pixel at screen space to where the texel is mapped back. The distance R is normalized to one. To take into account the contribution of each texel in calculating weights, we proposed a scheme in which the distance could be obtained easily through normalization of an edge function.

In the EWA method, a circular pixel is projected to an elliptical footprint. As shown in Fig. 1, the iso-value contour of the distance from the pixel center is also projected to a concentric ellipse in a texture space. Similarly, a square pixel is projected to a quadrilateral footprint in a texture space. The iso-value contour of the distance from the pixel center is a square. This is represented as the concentric quadrilateral in a texture space as shown in Fig. 7.

The concentric quadrilateral iso-value contour appears to be formed by the same values of four normalized edge functions that range from zero to one. The normalization is performed by dividing the edge function by the value of the edge function at the center of the footprint where the pixel center is projected. As shown in Fig. 7, the distance R is the complement to the normalized edge function. In detail, the distance R at an arbitrary point, is the complement to the smallest of four normalized edge functions. Consequently, the distance R is computed by subtracting the smallest normalized edge function from one. Furthermore, in the footprint inclusion test of the texel, normalized edge functions are used in place of nonnormalized edge functions. If all four normalized edge functions of an arbitrary point are positive, the point is located inside a footprint.

Being similar to the elliptical function of the EWA method, the edge function of EFATF approximates a footprint shape and represents the distance from the center of a pixel when a texel is mapped back to screen space. Therefore, in contrast to the TEXRAM, Feline, and FFPMM methods, our method calculates a better filter weight. The computation cost is reduced through a parallelogram approximation of a footprint, which works well in most cases [8]. This is discussed in detail in Section V.

IV. Results and Comparisons

A. Image Quality

To compare the image quality of various anisotropic filtering algorithms, we used the mean square signal-to-noise ratio (SNR), which is as follows:

$$\mathrm{SNR} = 10 \log_{10} \left(\frac{\sum_u \sum_v [I_T(u,v)]^2}{\sum_u \sum_v [I_T(u,v) - I_R(u,v)]^2} \right) \quad (4)$$

with

$I_T(u,v)$ test image that measures the quality;

$I_R(u,v)$ reference image used for the measurement.

We used two textures with a resolution of 256 × 256 texels such as checkerboard patterns and text fonts. These textures clearly show the quality degradation of images that are inadequately loaded and filtered. The resolution of rendered images was 640 × 480 pixels. The reference images were rendered using EWA filtering. The test images were rendered using TEXRAM, Feline, FFPMM, and EFATF with a parallelogram approximation of a footprint. The limit M, which is the number of texels fetched, changed from 8 to 64. In TEXRAM and Feline, the maximum number of probes is given by dividing M by eight since the probe filter requires eight texels. In detail, we used trilinear filtering as the probe filter. The portion of an image with aliasing artifacts is easily recognized even though the portion is very small. Accordingly, the quality of the portion with aliasing artifacts has a significant meaning.

Figs. 8–11 show the image quality of each filtering algorithm with two textures. The image quality of the checkerboard patterns are lower than the quality of the text fonts because the checkerboard patterns contain higher frequencies.

As shown in Figs. 8–11, the results of the TEXRAM and Feline methods have a poor quality; that is, a low SNR when M is small. However, as the limit M increases, the SNR improves rapidly. The rate of increase for the SNR in the Feline method is lower than in the TEXRAM method because Feline requires more probes than TEXRAM. Consequently, at the same

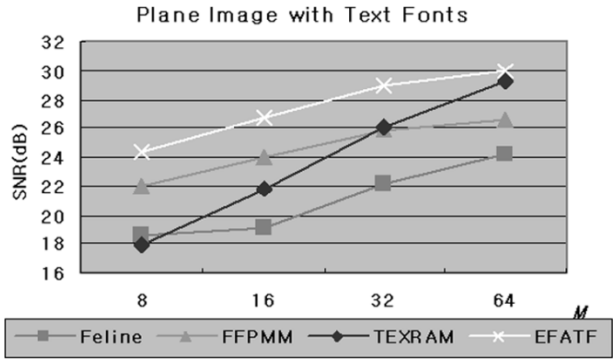

Fig. 8. Plane image quality of filtering algorithms for text fonts.

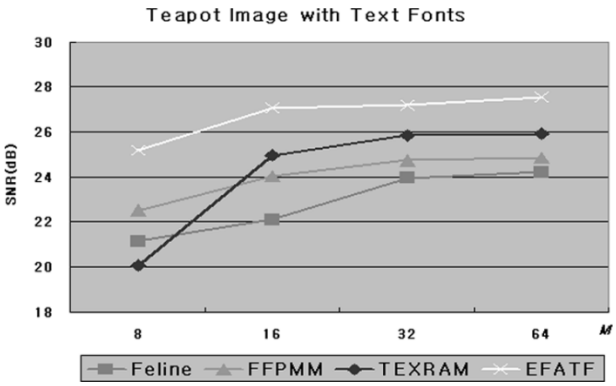

Fig. 9. Teapot image quality of filtering algorithms for text fonts.

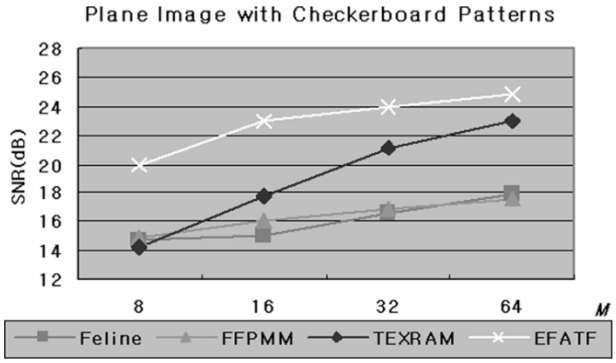

Fig. 10. Plane image quality of filtering algorithms for checkerboard patterns.

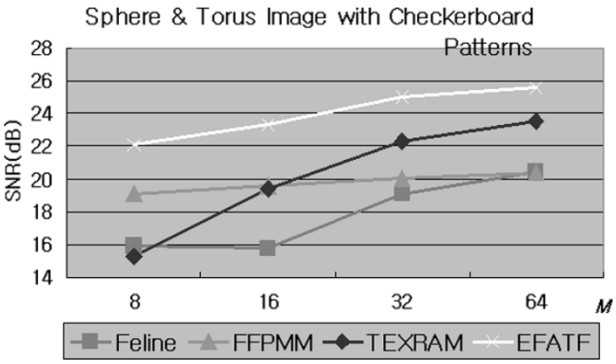

Fig. 11. Sphere & torus image quality of filtering algorithms for checkerboard patterns.

limit M, the SNR of Feline is lower than that of TEXRAM. TEXRAM performs well when the loaded texels are sufficiently

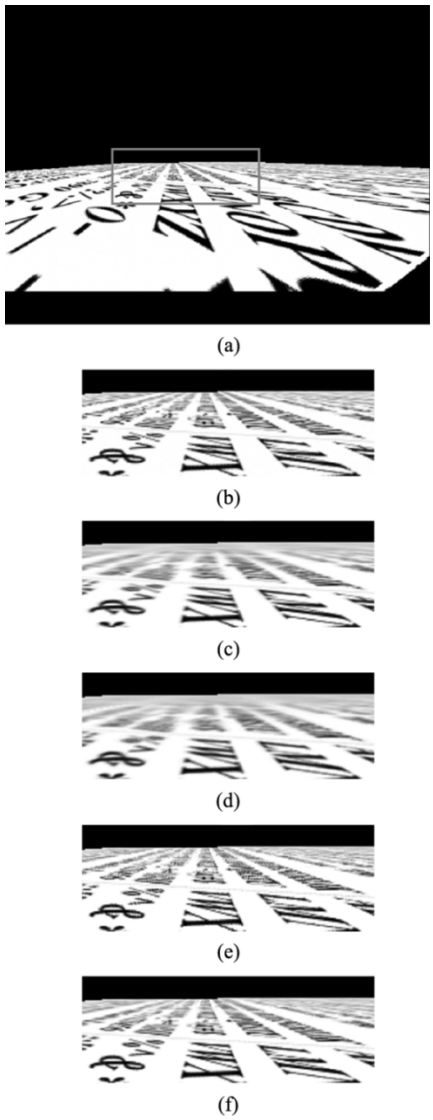

Fig. 12. Plane images of anisotropic filtering algorithms for text fonts (M : 16). (a) Whole image of EWA for text fonts. (b) Portion of image of EWA for text fonts. (c) Portion of image of TEXRAM for text fonts (M : 16). (d) Portion of image of feline for text fonts (M : 16). (e) Portion of image of FFPMM for text fonts (M : 16). (f) Portion of image of EFATF for text fonts (M : 16).

available. The difference between EFATF and TEXRAM, diminishes as the limit M increases.

Compared with the SNR of the TEXRAM method, the SNR of the FFPMM method is higher at a small M but lower at a large M, as shown in Figs. 8–11. The higher quality at a small M is due to an effective LOD selection scheme that considers the restricted number of loaded texels. The lower quality at a large M is caused by the weight error, which prevents the image quality from being improved. The difference between EFATF and FFPMM increases as the limit M increases.

EFATF achieves a higher SNR value than any other filtering algorithm at each limit M because the weight generation is based on a more exact footprint-shape approximation with an edge function and the relative contribution of each texel is applied to a good filter through the normalization of an edge function. The SNR of the EFATF method is higher than that of the TEXRAM method by about 1–6 dB; it is higher than that of the

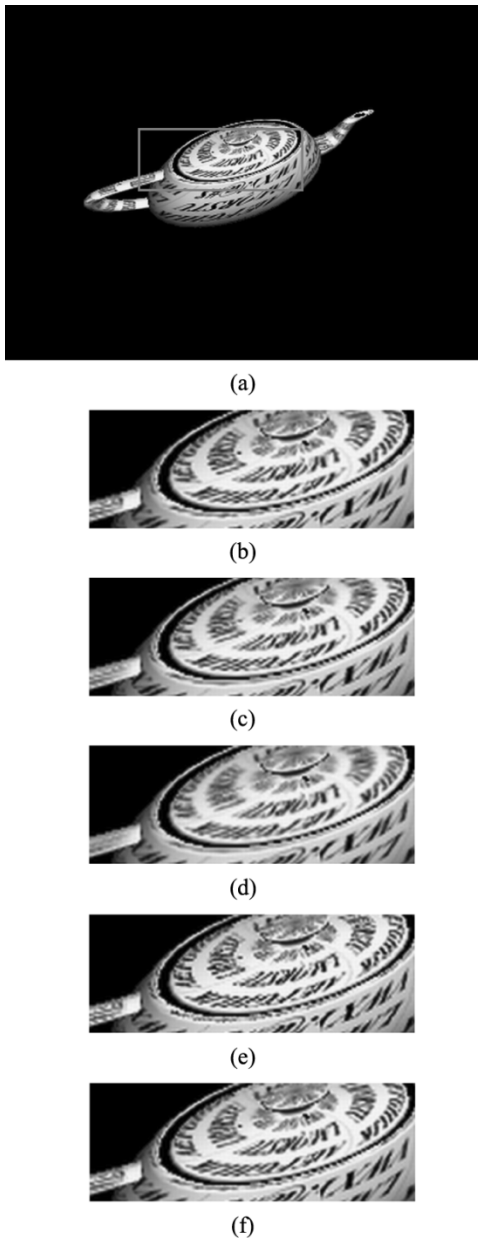

Fig. 13. Teapot images of anisotropic filtering algorithms for text fonts (M : 16). (a) Whole image of EWA for text fonts. (b) Portion of image of EWA for text fonts. (c) Portion of image of TEXRAM for text fonts (M : 16). (d) Portion of image of feline for text fonts (M : 16). (e) Portion of image of FFPMM for text fonts (M : 16). (f) Portion of image of EFATF for text fonts (M : 16).

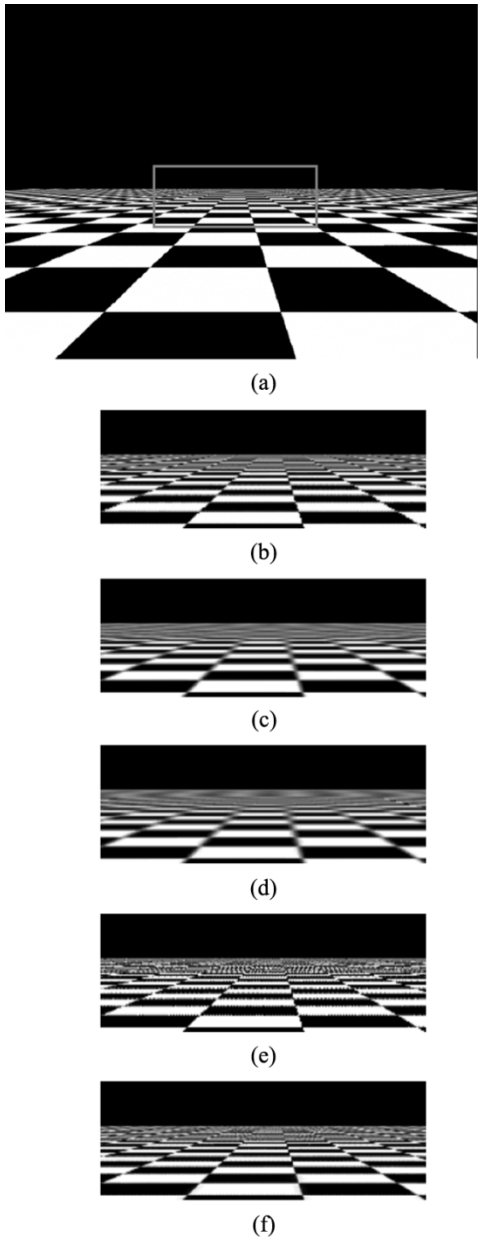

Fig. 14. Plane images of anisotropic filtering algorithms for checkerboard patterns (M : 16). (a) Whole image of EWA for checkerboard patterns. (b) Portion of image of EWA for checkerboard patterns. (c) Portion of image of TEXRAM for checkerboard patterns (M : 16). (d) Portion of image of feline for checkerboard patterns (M : 16). (e) Portion of image of FFPMM for checkerboard patterns (M : 16). (f) Portion of image of EFATF for checkerboard patterns (M : 16).

Feline method by 5–8 dB, and higher than that of the FFPMM method by 2–7 dB.

Fig. 12(a) and (b) shows the entire plane image with the text fonts using the EWA method, as well as, a portion of the image. Fig. 12(c)–(f) shows portions of the images with other algorithms. Fig. 13 shows the teapot images with the text fonts. Fig. 14 shows the plane images with the checkerboard patterns. Fig. 15 shows the sphere & torus images with the checkerboard patterns. The entire images as shown in Figs. 12(a)–15(a), are shrunk to half their original sizes.

From Figs. 8–11, the relation of SNR between EFATF and the other filtering algorithms is shown in Table I. In Table I, the algorithm on the right side of the equation has the highest SNR of three algorithms, TEXRAM, Feline, and FFPMM at the given limit M. For example, for the SNR of EFATF at M of 8 in the plane scene with text fonts, FFPMM at M of 16 achieves the same SNR as EFATF at M of 8. TEXRAM and Feline require the SNR at larger M than 16 to achieve the SNR of EFATF at M of 8. If the number of texels required by three algorithms to achieve the same SNR as EFATF is greater than 64, the relation is not represented because the current graphics card supports the anisotropy up to 8:1, that is, 64 texels. If some graphics cards support the anisotropy up to 16:1, these use bilinear filtering instead of trilinear filtering as the probe filter.

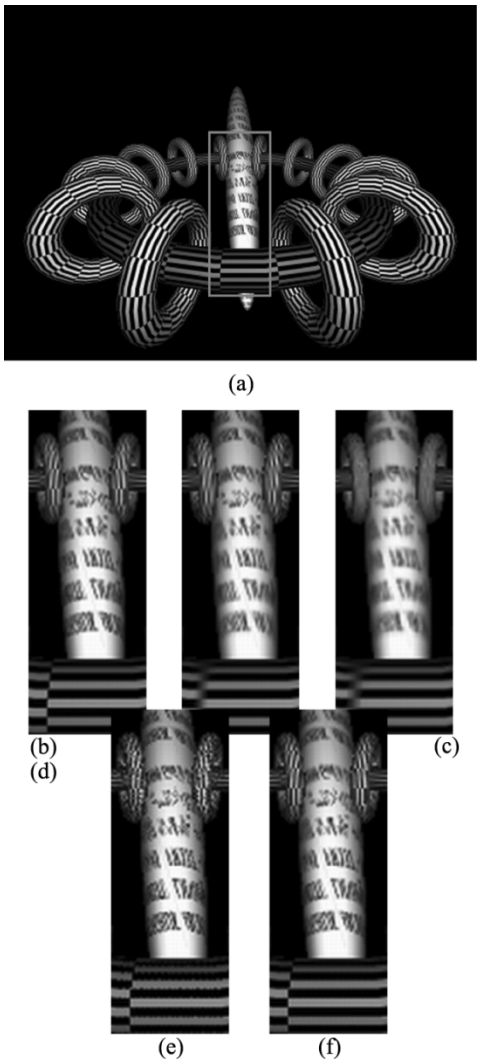

Fig. 15. Sphere & torus images of anisotropic filtering algorithms for checkerboard patterns $(M : 16)$. (a) Whole image of EWA for checkerboard patterns. (b) Portion of image of EWA for checkerboard patterns. (c) Portion of image of TEXRAM for checkerboard patterns $(M : 16)$. (d) Portion of image of Feline for checkerboard patterns $(M : 16)$. (e) Portion of image of FFPMM for checkerboard patterns $(M : 16)$. (f) Portion of image of EFATF for checkerboard patterns $(M : 16)$.

TABLE I
SNR RELATION BETWEEN EFATF AND OTHER FILTERING ALGORITHMS

Text fonts
Plane
$SNR_{\text{ETATF, }M=8} = SNR_{\text{FFPMM, }M=16}$
$SNR_{\text{ETATF, }M=16} = SNR_{\text{TEXRAM, }M=32}$
Teapot
$SNR_{\text{ETATF, }M=8} = SNR_{\text{TEXRAM, }M=16}$

Checkerboard patterns
Plane
$SNR_{\text{ETATF, }M=8} = 1/2*(SNR_{\text{TEXRAM, }M=16} + SNR_{\text{TEXRAM, }M=32})$
$SNR_{\text{ETATF, }M=16} = SNR_{\text{TEXRAM, }M=64}$
Sphere & Torus
$SNR_{\text{ETATF, }M=8} = SNR_{\text{TEXRAM, }M=32}$
$SNR_{\text{ETATF, }M=16} = 1/4*(SNR_{\text{TEXRAM, }M=32} + 3*SNR_{\text{TEXRAM, }M=64})$

B. Performance

The anisotropic texture filtering methods are performed at the specialized texture mapping hardware. The specialized texturing hardware is implemented as pipeline architecture. The current VLSI technology is quite capable of handling the computational overheads of texturing hardware. On the contrary, in spite of the texture cache, the limit of the memory bandwidth is still a bottleneck in texturing hardware. In detail, the texture cache guarantees a fixed supply of texels per clock cycle. However, since the number of texels supplied by the texture cache is small, it takes the texture cache several cycles to supply the number of texels that anisotropic filtering methods require. This phenomenon being a bottleneck for the texturing hardware determines the time required for anisotropic filtering. For instance, the recent NVIDIA graphics card supplies up to eight texels per clock cycle at texture pipeline through a cache. If the anisotropy is up to 8:1, the 64 texels are required. It takes a cache 8 cycles to supply 64 texels. Similarly, it takes the texturing hardware eight cycles to generate a filtered value.

TABLE II
CYCLE TIME OF FILTERING ALGORITHMS WITH TEXT FONTS

Scenes	M	Feline	FFPMM	TEXRAM	EFATF
Plane	8	75869	75869	75869	75869
	16	139785	151409	136837	113838
	32	234590	232462	227272	185891
	64	356794	321331	306723	283676
Teapot	8	12976	12976	12976	12976
	16	25912	25952	25888	24804
	32	37528	51904	34000	38940
	64	49424	103808	48336	54076

TABLE III
CYCLE TIME OF FILTERING ALGORITHMS WITH CHECKERBOARD PATTERNS

Scenes	M	Feline	FFPMM	TEXRAM	EFATF
Plane	8	62700	62700	62700	62700
	16	122672	107487	121480	101110
	32	244664	184426	241648	175818
	64	328440	272124	302888	245226
Sphere & Torus	8	58665	58665	58665	58665
	16	99672	104424	99656	97979
	32	224272	231979	207808	221373
	64	356352	437312	322072	358613

The cycle time required for anisotropic filtering is represented as the ratio of the number of texels required by the anisotropic filtering to the number of texels supplied by the texture cache per clock cycle. This is as follows:

$$CT(Cycle\_Time) = \frac{NtexAF}{NtexTC} \tag{5}$$

with

$NtexAF$ number of texels required by anisotropic filtering;

$NtexTC$ number of texels supplied by texture cache.

Tables II and III show the cycle time of each filtering algorithm assuming that a cache supplies eight texels per cycle. The time required for anisotropic filtering is obtained by multiplying the cycle time by the clock period. The clock period of recent graphics chips is about 3–4 ns.

TABLE IV
CYCLE TIME RELATION BETWEEN EFATF AND OTHER FILTERING METHODS

Text fonts

Plane

$CT_{ETATF,M=8} : CT_{FFPMM,M=16} = 75869 : 151409 = 1 : 2$

$CT_{ETATF,M=16} : CT_{TEXRAM,M=32} = 113838 : 227272 = 1 : 2$

Teapot

$CT_{ETATF,M=8} : CT_{TEXRAM,M=16} = 12976 : 25888 = 1 : 2$

Checkerboard patterns

Plane

$CT_{ETATF,M=8} : 1/2*(CT_{TEXRAM,M=16} + CT_{TEXRAM,M=32}) = 62700 : 181564 = 1 : 2.9$

$CT_{ETATF,M=16} : CT_{TEXRAM,M=64} = 101110 : 302888 = 1 : 3$

Sphere & Torus

$CT_{ETATF,M=8} : CT_{TEXRAM,M=32} = 58665 : 207808 = 1 : 3.54$

$CT_{ETATF,M=16} : 1/4*(CT_{TEXRAM,M=32} + 3*CT_{TEXRAM,M=64}) = 97979 : 293506 = 1 : 3$

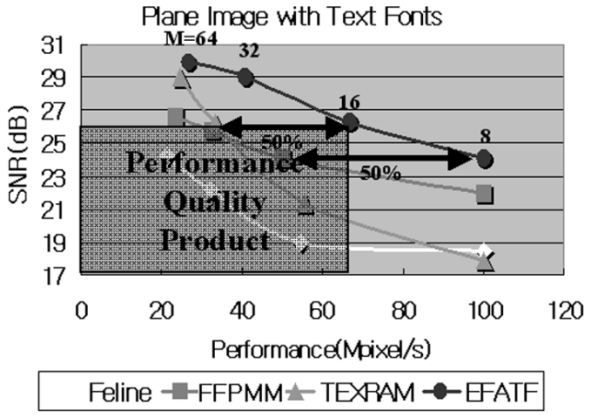

Fig. 16. Relationship between image quality and performance at plane image with text fonts.

From the SNR relation in Table I and the cycle time in Tables II and III, the relation of cycle time between EFATF and the other filtering methods for the same quality of images is generated and is shown in Table IV. From Table IV, to produce images of the same quality, the other methods require at least double the number of cycles of EFATF with text fonts and at least triple the number of cycles with checkerboard patterns.

The performance of texturing hardware is inversely proportional to the cycle time and is as follows:

$$P = \frac{100}{CT(Cycle\_Time)}. \tag{6}$$

For the same quality of images, the performance gain of EFATF compared with that of other anisotropic filtering increases by more than 200% for text fonts and 200% $\sim$ 300% for checkerboard patterns.

In addition, the relationship between image quality and performance is shown in Figs. 16–19. The image quality is represented as SNR and the performance is represented as the number of texture-filtered pixels, assuming that the number of texture filtered pixels are 100 M pixels per second at a given limit $M = 8$. The image quality is inversely proportional to the performance. The performance quality product is represented as a rectangle. From Figs. 16–19, the performance quality product of EFATF is best of all anisotropic filtering algorithms at each limit M.

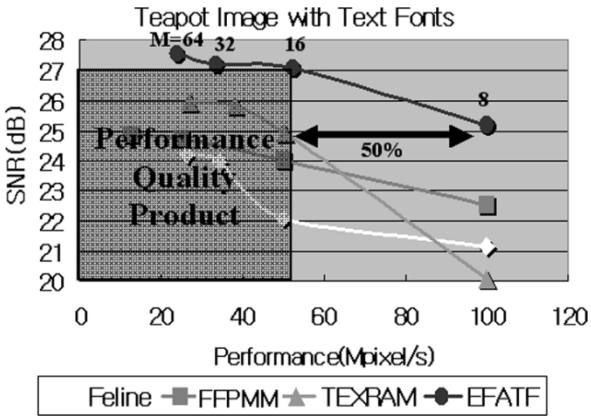

Fig. 17. Relationship between image quality and performance at teapot image with text fonts.

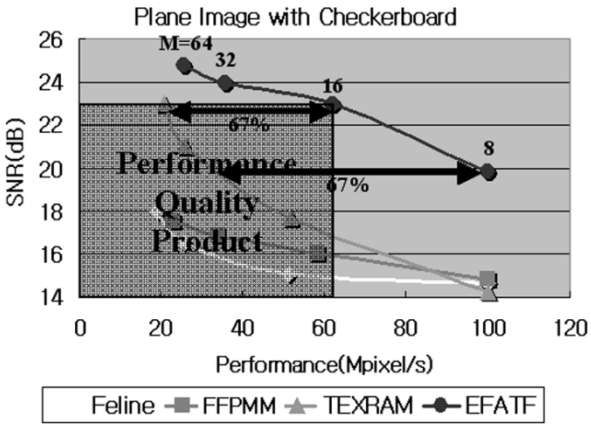

Fig. 18. Relationship between image quality and performance at plane image with checkerboard patterns.

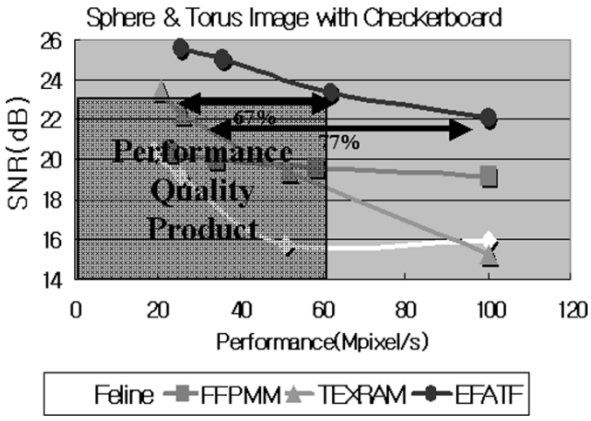

Fig. 19. Relationship between image quality and performance at sphere & torus image with checkerboard patterns.

V. HARDWARE ARCHITECTURE

In this section, we describe the texture mapping hardware. In addition, the hardware cost is compared with the cost of other anisotropic filtering algorithms.

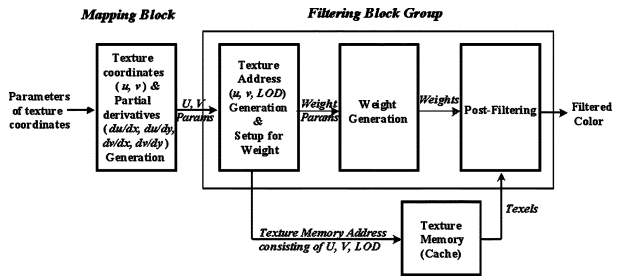

Fig. 20. Hardware architecture of texture mapping.

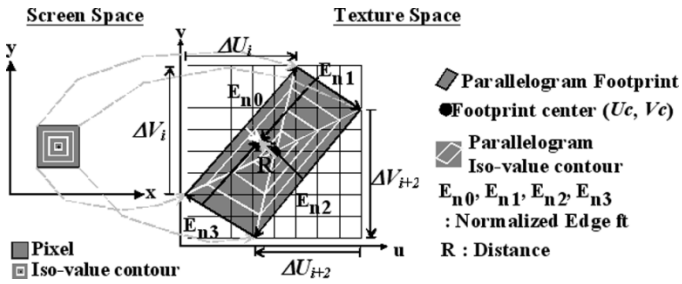

Fig. 21. Parallelogram approximation of a footprint.

A. Hardware Architecture of Texture Mapping

Texture mapping hardware consists of the mapping part and filtering part, as shown in Fig. 20.

The mapping unit calculates the texture coordinates to which the screen coordinates correspond, along with the partial derivatives of the texture coordinates. The calculation requires the division operation. If the values are calculated with errors due to the lack of precision of the bit width, the rendered image shows the distortions. Hence, high precision is required. The mapping unit is common to trilinear and anisotropic filtering algorithms. Using the texture coordinates and their partial derivatives, the filtering unit generates the texture memory address and weights used during the post-filtering. The filtering unit, which comprises the generation of the texture address and the setup for the weights, weight generation, and post-filtering blocks, has different operations whose sequence follows the texture filtering algorithms.

The bit width of the texture mapping unit is settled by performing precision modeling within a C/C++ simulation environment. This process allows fixed-point precision to be applied to each part of the filtering procedure. From the simulation, the bit width of the mapping unit is different from the filtering unit. The bit width of the mapping unit is 32 bits with 18-bit fractional precision. To prevent distortion of the image, high precision is required to calculate the texture coordinates and their partial derivatives. The filtering unit requires 24 bits with 13-bit fractional precision. The integer parts of the texture coordinates are required to calculate the texture memory address. The moderate precision of the fractional parts of the texture coordinates and the partial derivatives are sufficient to calculate the weight. During the simulation, the image quality is hardly degraded when the bit width of the filtering unit is reduced to 24 bits. The sub-blocks of the filtering unit are now discussed.

1) Texture Address Generation and Setup for Weights: In the bock, our filtering method selects the LOD and then generates the address and sets up the weights in parallel. In relation to the weight setup, the partial derivatives of a normalized edge function and its initial value at the center of a footprint are calculated because the normalized edge function is expressed as the linear equation of (u, v) coordinates.

The normalized edge function is expressed as

$$E_{ni}(u,v) = \frac{E_i(u,v)}{E_i(u_c, vc)} = \frac{\{(u-u_i)*\mathrm{deu}_i + (v-v_i)*\mathrm{dev}_i + 0.5\}}{\{(u_c-u_i)*\mathrm{deu}_i + (v_c-v_i)*\mathrm{dev}_i + 0.5\}} \quad \text{with } (u_c, v_c) \text{ at the footprint center, } i = 0 \sim 3. \tag{7}$$

The value 0.5 is included in (7) because a texel whose edge function value is more than -0.5 should be included in a footprint. By substituting deu_i and dev_i of (3)

$$E_{ni}(u,v) = \frac{\{(u-u_i)*\Delta V_i - (v-v_i)*\Delta U_i + 0.5*(|\Delta U_i| + |\Delta V_i|)\}}{\{(u_c-u_i)*\Delta V_i - (v_c-v_i)*\Delta U_i + 0.5*(|\Delta U_i| + |\Delta V_i|)\}} = \frac{\{\Delta V_i*(u-u_i) - \Delta U_i*(v-v_i) + 0.5*(|\Delta U_i| + |\Delta V_i|)\}}{Dc_i}.$$

with $Dc_i = \{(u_c-u_i)*\Delta V_i - (v_c-v_i)*\Delta U_i + 0.5*(|\Delta U_i| + |\Delta V_i|)\}$.

The linear equation of a normalized edge function is, therefore, expressed as follows:

$$E_{ni}(u,v) = \left(\frac{dE_{ni}}{d_u}\right)*(u-u_i) + \left(\frac{dE_{ni}}{d_v}\right)*(v-v_i) + C \tag{8}$$

with

$$\frac{dE_{ni}}{d_u} = \frac{\Delta V_i}{Dc_i}, \quad \frac{dE_{ni}}{dv} = -\frac{\Delta U_i}{Dc_i}$$
$$\text{and } C = 0.5 * \frac{(|\Delta U_i| + |\Delta V_i|)}{Dc_i}.$$

The values of (8) at texels are given by interpolating the initial value of $E_{ni}(u,v)$ with the coefficients (dE_{ni}/du) and (dE_{ni}/dv). The initial value is calculated at the nearest integer coordinates of the center of the footprint. The integer coordinates denoted by (u_{cI}, v_{cI}) are expressed as $u_{cI} = u_c + u_{\mathrm{cerr}}$ and $v_{cI} = v_c + v_{\mathrm{cerr}}$. From $u = u_{cI}$ and $v = v_{cI}$ in (8), the initial value is as follows:

$$E_{ni}(u_{cI}, v_{cI}) = 1 + \left(\frac{dE_{ni}}{d_u}\right) * u_{\mathrm{cerr}} + \left(\frac{dE_{ni}}{d_v}\right) * v_{\mathrm{cerr}}. \tag{9}$$

In this simplified equation, the error term u_{cerr} and v_{cerr} range from -0.5 to $+0.5$. That is, a small portion of the integer part of u_{cerr} and v_{cerr} is used. Therefore, 24 × 15-bit multiplication is used in place of 24 × 24-bit multiplication.

Since TEXRAM's parallelogram approximation of the footprint shape works well in most cases [8], it is used to calculate the corner point positions of the footprint, as shown Fig. 21. Fig. 21 also shows that ΔU_i and ΔV_i of the parallel edges, have the opposite sign and same absolute value. In addition, $(u_c - u_i)$

and $(v_c - v_i)$ of the parallel edges have the same feature as ΔU_i and ΔV_i. This relationship is expressed as

$$\begin{aligned} \Delta U_i &= -\Delta U_{i+2} \\ \Delta V_i &= -\Delta V_{i+2} \\ (u_c - u_i) &= -(u_c - u_i + 2) \\ (v_c - v_{i+2}) &= -(v_c - v_{i+2}) \\ i &= 0, 1. \end{aligned} \tag{10}$$

As a result, the partial derivatives of the normalized edge functions that are parallel, are the opposite sign and same absolute value. The partial derivatives of two intersecting edges are calculated and the operational cost of calculating the partial derivative is accordingly reduced by half. In generating the partial derivatives in (8), two division operations are required. Since the denominators of the partial derivatives are common for each edge, the two division operations are replaced by the reciprocation of the common denominator of the partial derivatives and the two multiplication operations that use the reciprocal. Consequently, the hardware cost is further reduced.

The texture mapping unit operates in parallel with other units, which generally have short latency in comparison with the texture mapping unit. Since the buffer size for the output of the other units increases in proportion to the latency of the texture mapping unit, the latency of the division operation must be short and the convergence division method is selected [2]. In practice, a divider is implemented in the pipeline for the throughput and the convergence division method is used for the reciprocation. Since the value of the dividend is always one at reciprocation, the divider architecture is simplified and the hardware cost is reduced.

2) Weight Generation: To generate the weights for the M texels, which are generated in parallel, the normalized edge functions are first interpolated and then, compared to obtain the distance R for each of the M texels. The weights are obtained by indexing in parallel the weight tables of the Gaussian filter with each distance R.

As shown in Fig. 21, the sum of the normalized edge functions that were parallel to each other is two at any texel. This relation is expressed as

$$\begin{aligned} &E_{ni}(u,v) + E_{n(i+2)}(u,v) \\ &\quad = E_{ni}(u_c, v_c) + E_{n(i+2)}(u_c, v_c) = 2 \\ &\quad => \{1 - E_{ni}(u,v)\} \\ &\quad = -\left\{1 - E_{n(i+2)}(u,v)\right\}, \quad i = 0, 1. \end{aligned} \tag{11}$$

The distance R_i' of each edge is defined as the complement to the normalized edge function of each edge, and (11) is converted to $R_i' = -R_{i+2}'$. Consequently, one of the two normalized edge functions that were parallel is interpolated, which reduces the hardware cost. Since the only positive distance R_i' is valid, we evaluated the absolute values that were interpolated. The distance R is determined by comparing two absolute values. The Gaussian filter is pre-computed and stored in a ROM with 64 bytes, or $2^6 \times 8$ bits, for speed. For parallel operation in a given limit M, the number of ROMs required is M.

3) Post-Filtering: Post-filtering is performed through 8×8-bit multiplication of the texel value by the weight. In our filtering method, we obtained the final filtered value by dividing the sum of the weighted color values by the sum of the weights. Because the division was performed for four-color channels (RGBA), we replaced the division by the sum of the weights with multiplication by the reciprocal of the sum of the weights.

TABLE V
COUNT OF OPERATIONS IN MAPPING UNIT (RECIP: RECIPROCATION; MUL: MULTIPLICATION)

	TEXRAM	Feline	FFPMM	EFATF
Recip	1(32b)			
Mul	15(32b)			
Add/Sub	4(32b)			
Shift	6(10)			

TABLE VI
COUNT OF OPERATIONS AT TEXTURE ADDRESS GENERATION AND SETUP FOR WEIGHT BLOCK OF FILTERING UNIT (COMP: COMPARATOR; INC/DEC: INCREMENT OR DECREMENT; 2'S COM: 2'S COMPLEMENT)

	TEXRAM	Feline	FFPMM	EFATF
Recip	1	2	0	2
Sqrt	2	2	0	0
Mul	9	13	1	7 + 4(24x15)
Add/Sub	8 + *M*(11b)	11+(*M*+1)(11b)	22	17
Shift	5(10) + 2(3) + 2(11b,10)	5(10) + 2(11b,10)	10(10) + (6) + 2(11b,10)	2(10) + 1(6) + 2(11b,10)
Comp	7 + *M*/8 + *M*(11b)	9 + *M*(11b)	14+ *M*(11b)	5 + *M*(11b)
Inc/Dec	1 + *M*/2(11b)	1 + *M*/2(11b)	*M*(11b)	*M*(11b)
2's Com	0	0	0	4
Mux	0	0	1	1
Table	0	64bytes	0	0

TABLE VII
COUNT OF OPERATIONS IN THE WEIGHT GENERATION BLOCK OF THE FILTERING UNIT

	TEXRAM	Feline	FFPMM	EFATF
Add/Sub	0	0	0	4 + 2x*M*
Comp	0	0	0	2(17b) + *M*(8b)
2's Com	0	0	0	2(17b)
Table	0	128x*M* Bytes	22KB(*M*: 8) 286KB(*M*: 16) 3.5MB(*M*: 32) 6.6MB(*M*: 64)	64x*M* Bytes

B. Hardware Cost

In this section, the hardware cost of various anisotropic filtering algorithms is compared. The computation cost of each block is calculated through the count of various operations, such as addition. We represented the hardware cost of each arithmetic unit as a relative ratio among various arithmetic units. The gate equivalents of the arithmetic units were estimated through synthesis using a 0.35-μm 3.3-V standard cell library.

Tables V–VIII show the count of various operations at each block. Since we assumed that operations were performed in parallel at a given limit M, the count of some operations are expressed as a function of the limit M. The division is counted as a reciprocal calculation and the multiplication by its reciprocal. The bit width of the filtering unit is taken as a reference. The terms 32 b, 11 b, and 8 b in parentheses, represent the other bit width. In Table VIII, 11 b represents the integer part of the bit

TABLE VIII
COUNT OF OPERATIONS IN POST-FILTERING BLOCK OF FILTERING UNIT

	TEXRAM	Feline	FFPMM	EFATF
Recip	0	0	0	1(14b)
Mul	7xM/2(8x8)	4xM(8x8)	4xM(8x8)	4(22x22)+ 4xM(8x8)
Add/Sub	7xM/2(8b) + (M-1)/2(11b) + 7xM/2(16b)	7xM/2(8b) + (M-1)/2(11b) + 7xM/2(16b)	4xM(8b) + 4x(M-1)(14b)	4x(M-1)(22b) + (M-1)(14b)
Shift	4(11b,3)	0	0	0

TABLE IX
HARDWARE COST RATIO OF ARITHMETIC UNITS COMPARED TO 24 BIT ADDER

	Hardware cost ratio
Add/Sub	1.0(24b), 1.4(32b), 0.61(22b), 0.42(16b), 0.34(14b), 0.25(11b), 0.18(8b)
Mpy	16.5(32x32), 8.84(24x24), 6.44(24x15), 7.12(22x22), 1.12(8x8)
Shifter	0.74(32b,10), 0.55(24b,10), 0.38(24b,6), 0.25(24b,3), 0.23(11b,10), 0.14(11b,3)
Recip	55.15(32b), 36.6(24b), 26.5(14b)
Sqrt	45.44(24b)
Cmp	0.49(24b), 0.35(17b), 0.19(11b), 0.09(8b)
Inc/Dec	0.69(24b), 0.11(11b)
2's Com	0.5(24b), 0.34(17b)
Mux	0.18(24b)
ROM Table	0.28(64B), 0.57(128B), 100 (22KB), 1301(286KB), 16311(3.5MB), 30758(6.6MB)

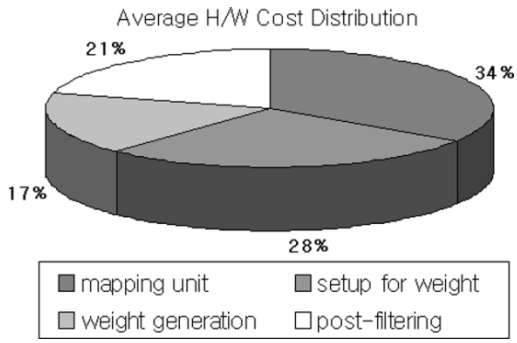

Fig. 22. Average hardware cost distribution.

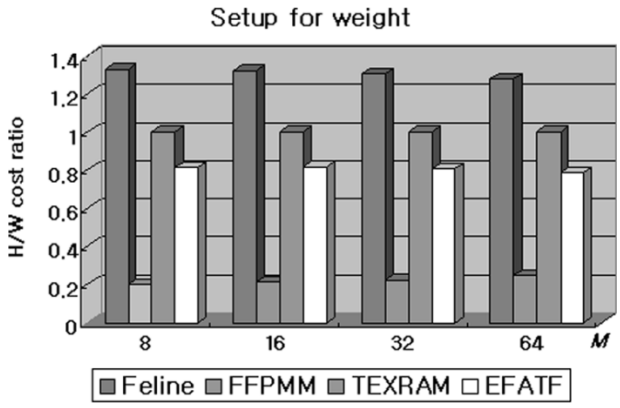

Fig. 23. H/W cost ratio of weight setup block of filtering unit.

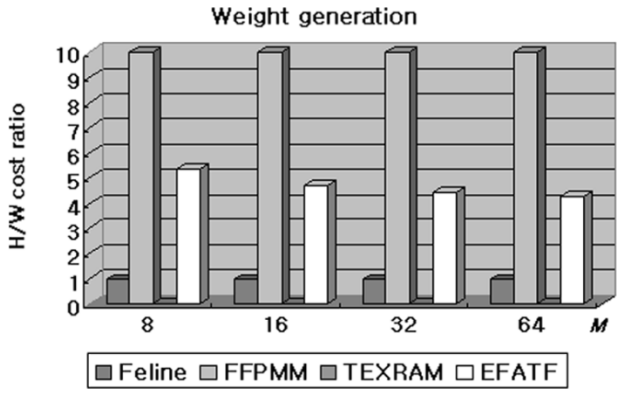

Fig. 24. H/W cost ratio of weight generation block of filtering unit.

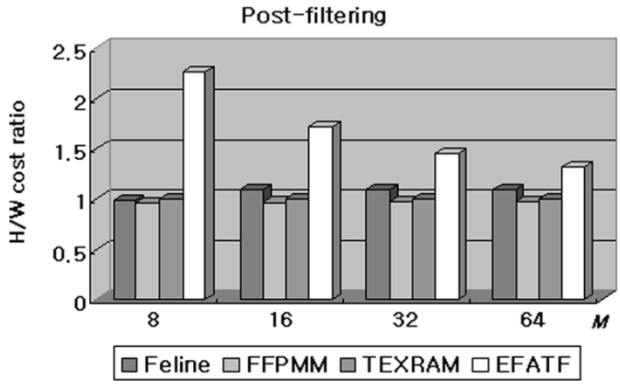

Fig. 25. H/W cost ratio of post-filtering block of filtering unit.

width, and the value shown in parenthesis at the shift operation represents the shift range.

Table IX shows the hardware cost of the arithmetic units required by filtering algorithms. The hardware size of a 24-bit adder was normalized to one. The hardware cost of square root logic was assumed to be equal to the divider. The one bit of ROM table was assumed to be a quarter of a gate equivalent. From Tables V to IX, Fig. 22 shows the average hardware cost distribution of each block of texture mapping hardware. On the average, the mapping unit occupies 34% of the entire texture mapping hardware. The setup for weight sub-block of filtering units occupies 28%. The post-filtering sub-block subsequently occupies 21% and the weight generation sub-block occupies 17%. Figs. 23–25 show the hardware cost ratio of each block of the filtering unit in texture mapping hardware, while the limit M changes from 8 to 64. The hardware cost ratio is given as the ratio of other anisotropic filtering algorithms to the TEXRAM algorithm. Since the hardware cost of the TEXRAM algorithm is zero in the weight generation sub-block, Fig. 24 shows the hardware cost ratio of each filtering algorithm to the Feline algorithm. On the contrary from Fig. 23, the EFATF method has less hardware cost than the TEXRAM method. The FFPMM method also has less hardware cost than the TEXRAM method. However, the simple setup for weight of the FFPMM method increases the hardware cost of the weight generation sub-block dramatically. In the weight generation sub-block, since the weight is obtained directly from the fractional part of the (u, v) texture coordinates, the TEXRAM method does not have the hardware cost as shown in Fig. 24. In this block, the EFATF method requires about 4.3 $\sim$ 5.3 times the hardware cost of the Feline method. However, because the Feline method requires only ROM tables by $128 \times M$ B, the hardware cost of EFATF method, in fact, is small. In the post-filtering sub-block, because

TABLE X
HARDWARE COST RATIO OF FILTERING ALGORITHMS COMPARED TO TEXRAM

	TEXRAM	Feline	FFPMM	EFATF
$M = 8$	1.0	1.13	0.86	1.07
$M = 16$	1.0	1.14	2.75	1.11
$M = 32$	1.0	1.15	22.52	1.16
$M = 64$	1.0	1.15	32.72	1.23

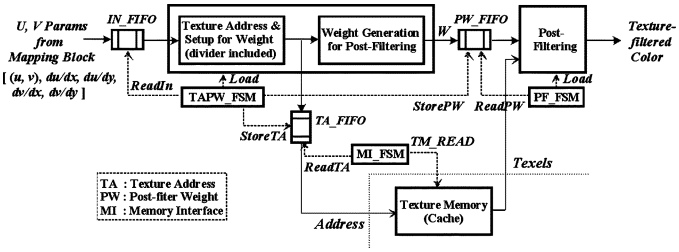

Fig. 26. Detailed hardware architecture of a filtering unit.

these methods require the division by the sum of weights, the EFATF method has more hardware cost than the TEXRAM method, as shown in Fig. 25. Table X shows the hardware cost ratio of each anisotropic filtering algorithm for the TEXRAM method. The cost is based on the data in Tables V–IX. The hardware cost ratio of the Feline method to the TEXRAM method changes from 1.13 to 1.15 as the limit M increases. For the FFPMM method, the hardware cost ratio changes from 0.86 to 32.72 due to the large size of the area weight table, which makes hardware implementation impossible. For the EFATF method, the hardware cost ratio changes from 1.07 to 1.23. In practice, the real additional overheads of EFATF to TEXRAM are from 7% to 11% because the limit M, which the graphics card supplies, ranges from 8 to 16.

In comparing the hardware cost, the EFATF method costs slightly more than the TEXRAM or Feline methods. However, the continuous development of VLSI technology is reducing the hardware overheads of EFATF.

VI. VLSI IMPLEMENTATION

The proposed algorithm was demonstrated through VLSI implementation. The hardware architecture of the proposed algorithm was described by Verilog and synthesized using a 0.35-μm, 3.3-V standard cell library. The layout was performed by an automatic place-and-route tool. The functionality was verified through test vectors. The only filtering block that differed from each filtering algorithm was implemented because the mapping block is common for all texture filtering algorithms.

Fig. 26 shows the detailed hardware architecture of the filtering unit, which consists of datapath blocks, finite state machines (FSMs) and first-in first-out (FIFO) buffers. Each datapath block is implemented in a pipeline for the throughput. The FSM controls each datapath and the FIFOs, while the MI_FSM, which is shown in Fig. 26, generates controls for the memory interface. By functioning as a buffer between adjacent datapath blocks, the FIFO reduces the dependency between adjacent blocks in the data path. Thus, the performance degradation of a particular block does not directly affect the adjacent blocks.

TABLE XI
TOTAL AREA AND POWER OF FIFOS ON THE DEPTH (GES: GATE EQUIVALENTS; DEPTH: IN_FIFO, TA_FIFO, PW_FIFO)

Depth	(2,2,4)	(4,4,8)	(8,8,16)
Area (GEs)	4,917(3.24%)	9,4287(6.20%)	18,633(12.26%)
Power (mW)	105.58(5.03%)	116.06(5.52%)	135.15(6.43%)

This hardware is also implemented to perform the parallel operation in a given limit M of eight.

A multiplier in a datapath is synthesized as a Wallace tree multiplier using the Synopsys Design Compiler. The DesignWare Foundation of Design Compiler has technology-independent soft-macros and enables a higher performance implementation. A divider is implemented in a pipeline for the throughput. Since the other units, which operate in parallel with the texture mapping unit, have short latency, it is desirable for the latency of division to be short. Accordingly, we implement the divider with the convergence division method instead of a high-radix method. In the convergence method, three cycles were required to calculate the reciprocal. Four cycles were needed to perform the division operation. The latency is the same as in the 32-bit fixed-point division operation in the mapping block. In the radix-4 division method, because two stages per cycle were feasible, four bits of quotient per cycle were generated. Seven cycles are, therefore, required for the 24-bit division operation and nine cycles were required for the 32-bit division. In the 24-bit division, the radix-4 method required 1.75 times the cycles of the convergence method. In the 32-bit division, the radix-4 method required 2.25 times the cycles.

In FIFO synthesis, the depth of FIFOs such as IN_FIFO and TA_FIFO, as shown in Fig. 26, ranges from two to eight. In particular, the depth of PW_FIFO, which buffers the post-filtering weights, ranges from four to sixteen when the texture cache miss and so on are considered. In this case, we assume that the miss penalty is very small. If the miss penalty is large, the depth of PW_FIFO is very large and the prefetching scheme must be used to prevent the performance degradation. Table XI shows the total area and power of FIFO as the depth of FIFO changes. The percentage in parentheses represents the ratio of FIFOs to the entire hardware. In area, the FIFO occupies 3.2%–12.26% of all the hardware. The power of FIFO corresponds to 5.03%–6.43% of all the hardware. The area and power of all the hardware are given in Table XIV.

A pipeline register occupies a considerable part of the entire hardware because the filtering unit takes the pipeline architecture. In general, the pipeline register with a control signal, such as a load, consists of a flip-flop and a multiplexer. The multiplexer of the pipeline register recirculates data when the register is idle, which means the load signal is low. Consequently, pipeline registers in synchronous systems are clocked every cycle even when idle, resulting in wasteful power dissipation. To reduce the wasteful power dissipation in registers, we used clock gating with the pipeline register, enabling the clock registers function to be disabled when idle. The number of pipeline registers in the entire hardware was about 3500. Table XII shows that registers with multiplexers required 43.96% more area than registers with a gated clock.

TABLE XII
AREA OF PIPELINE REGISTERS ($\mathrm{DIFF_{REG}}$: $\mathrm{REG_{MUX}} - \mathrm{REG_{GATED\ CLOCK}}$; TOTA: TOTAL AREA OF THE HARDWARE)

	$\mathrm{reg_{mux}}$	$\mathrm{reg_{gated\ clock}}$	$\mathrm{diff_{reg}}$ /$\mathrm{reg_{gated\ clock}}$	$\mathrm{diff_{reg}}$ / TotA
Area (GEs)	26,950 (17.74%)	18,721 (12.32%)	43.96%	5.42%

TABLE XIII
POWER OF THE PIPELINE REGISTERS ($\mathrm{DIFF_{REG}}$: $\mathrm{REG_{MUX}} - \mathrm{REG_{GATED\ CLOCK}}$; TOTP: TOTAL POWER OF THE HARDWARE)

		0.0	0.0625(1/16)	0.125(1/8)
Power (mW)	$\mathrm{reg_{mux}}$	643.15 (30.6%)	700.33 (33.32%)	728.57 (34.67%)
	$\mathrm{reg_{gated\ clock}}$	611.80 (29.1%)	601.39 (28.61%)	562.53 (26.77%)
	$\mathrm{diff_{reg}}$ / $\mathrm{reg_{gated\ clock}}$	5.12%	16.45%	29.52%
	$\mathrm{diff_{reg}}$ / TotP	1.49%	4.71%	7.90%

This phenomenon occurs because although the multiplexer is required per bit, the gated clock drives several flip-flops. The difference is less than 5.42% of the entire hardware. The percentage in parentheses represents the area ratio of the registers to the entire hardware.

Table XIII shows the power of pipeline registers. The parameter Probstall in Table XIII, represents the probability of pipeline stall. The pipeline may stall if the texture memory does not supply the required texels due to the texture cache miss. An FSM makes the datapath stall by generating a low-load signal. The probability of the pipeline stalling is proportional to the texture cache miss and the miss penalty. Specifically, the probability of the pipeline stalling is given as the multiplication of the miss rate by the ratio of the miss penalty to the clock period. From the texture cache related papers [14] and [15], the texture cache miss rate is less than 1% but the miss penalty is given as several tens times the clock period. Thus, the probability of the pipeline stalling is given as less than several tens percentage. Table XIII shows that as the probability of pipeline stall increases, the power of the register with the multiplexer increases but the register with the gated clock decreases. In the register with the multiplexer, a pipeline stall increases the switching activity of the load signal. The switching activity increases in the circuits with the multiplexers connected to a load signal. However, in the register with the gated clock, the pipeline stall disables the clock and decreases the switching activity of the circuits in the flip-flops connected to the clock. Consequently, as the stall probability changed, the power of the registers with the gated clock decreased to 5.12%$\sim$29.52% compared to the registers with the multiplexers. These differences are less than 1.49%–7.90% for the entire hardware. The percentage shown in parentheses in Table XIII represents the power ratio of the registers to the entire hardware.

The area and power of sub-blocks are given in Table XIV. The area and power were obtained if the depth of the FIFOs was (2,2,4) and the probability of a stall was zero. The value

TABLE XIV
AREA AND POWER OF SUB-BLOCKS (WS: WEIGHT SETUP; WG: WEIGHT GENERATE; PF: POST-FILTERING)

		WS	WG	PF	FSM	FIFOs	Total
Area	GEs	87,061 (49,853)	21,843	37,960	179	4,917	151,960
	%	57.28 (32.81)	14.38	24.98	0.12	3.24	100.0
Power	mW	1,068.36 (685.36)	286.02	640.50	1.09	105.58	2,101.55
	%	50.83 (32.61)	13.61	30.48	0.05	5.03	100.0

TABLE XV
VLSI DESIGN CHARACTERISTICS

Parameters	Values
Technology	0.35 um CMOS
Gate Equivalents	151,960 GEs
Core size	3 mm X 3 mm
Supply voltage	3.3 V
Operating frequency	100 MHz
Average power dissipation	2,100 mW

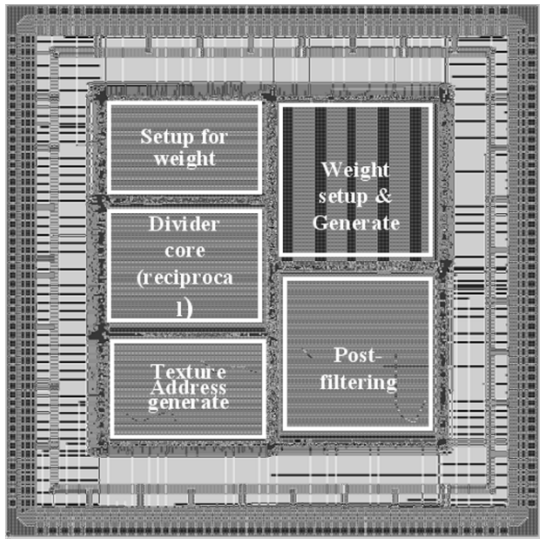

Fig. 27. Layout of the filtering pipeline hardware.

in parentheses represents the information about dividers. From this value, dividers occupy 32.81% and 32.61% of the entire hardware in area and power, respectively.

The VLSI design characteristics are summarized in Table XV. An operating frequency of 100 MHz at 3.3 V, the proposed anisotropic texture filtering hardware generates 100 M of texture-filtered RGB pixel color values per second. The area of the entire hardware is about 0.15 million gate equivalents. The total power dissipation is 21 mW/MHz if the filtering pipeline is fully operated without the pipeline stalling. The layout is shown in Fig. 27.

VII. CONCLUSION

Texture mapping hardware plays a key role in the overall pixel throughput of the rendering system and in the quality of rendered images. In spite of the texture cache, the limit of

the memory bandwidth is still a bottleneck of texture mapping hardware.

To solve the conflict between the quality and the performance, we propose the EFATF method. With the edge function, the proposed algorithm approximates a footprint shape more exactly. The normalized value of an edge function gives the distance from the center of the pixel at screen space when the texel is mapped back. The distance represents the contribution of the texel to a final intensity. Applying the distance to a Gaussian filter generates a good weight.

The quality of images rendered by the EFATF method is superior to that of other methods using the same number of texels. To achieve the same image quality, other methods require more than twice the number of texels of the proposed method. Compared with other methods, the performance gain of the proposed method is at least 200% for text fonts and 200%$\sim$300% for checkerboard patterns. The additional overheads of the proposed algorithm, compared with the overheads of the TEXRAM method, are about 7%–11% as the limit M increases from 8 to 16. The proposed method can be implemented in hardware at a reasonable cost. In practice, the VLSI implementation demonstrates the algorithm and the feasibility of hardware implementation. The hardware is described by Verilog and synthesized with a 0.35- μm, 3.3-V standard cell library. The hardware is operated at 100 MHz and generates 100-M texture-filtered RGB pixel color values per second under a sufficient texture memory bandwidth. The filtering hardware can be used as a system on a chip with intellectual property.

REFERENCES

[1] R. J. Cant and P. A. Shrubsole, "Texture potential MIP mapping," *ACM Trans. Graph.*, vol. 19, no. 3, pp. 164–18, Jul. 2000.

[2] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1993, ch. 8, pp. 153–158.

[3] N. Greene and P. Heckbert, "Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter," *IEEE Comput. Graph. Appl.*, vol. 6, no. 6, pp. 21–27, Jun. 1986.

[4] P. S. Heckbert, "Survey of texture mapping," *IEEE Comput. Graph. Appl.*, vol. 6, no. 11, pp. 56–67, Nov. 1986.

[5] P. S. Heckbert, "Fundamental of texture mapping and image warping," M.S. thesis, Comput. Sci. Div., Univ. California, Berkeley, CA, 1989.

[6] T. Hüttner and W. Straßer, "Fast footprint MIP mapping," in *Proc. EUROGRAPHICS/SIGGRAPH Workshop Graphics Hardware*, 1999, pp. 35–44.

[7] D. B. Kirk, "Unsolved problems and opportunities for high-quality, high-performance 3D graphics on a PC platform," in *Proc. EUROGRAPHICS/SIGGRAPH Workshop Graphics Hardware*, 1998, pp. 11–13.

[8] J. McCormack, R. Perry, K. I. Farkas, and N. P. Jouppi, "Feline: Fast elliptical lines for anisotropic texture mapping," in *Proc. SIGGRAPH Ann. Conf. Comput. Graphics*, 1999, pp. 243–250.

[9] J. McCormack, R. McNamara, C. Gianos, L. Seiler, N. Jouppi, K. Correll, T. Dutton, and J. Zurawski, Neon: A (big) (fast) single-chip 3D workstation graphics accelerator. Compaq Western Research Lab., Palo Alto, CA, WRL Research Report 98/1, 1999, Revised.

[10] A. Schilling, G. Knittel, and W. Straßer, "Texram: A smart memory for texturing," *IEEE Comput. Graph. Appl.*, vol. 16, no. 3, pp. 32–41, May 1996.

[11] A. Schilling, "A new simple and efficient antialiasing with subpixel masks," *ACM Comput. Graph.*, vol. 25, no. 4, pp. 133–141, Jul. 1991.

[12] L. Williams, "Pyramidal parametrics," *Comput. Graph.*, vol. 17, pp. 1–11, Jul. 1983.

[13] R. C. Landsdale, "Texture mapping and resampling for computer graphics," M.S. thesis, Dept. Elect. Eng., Univ. Toronto, ON, Canada, 1991.

[14] H. Igehy, M. Eldridge, and K. Proudfoot, "Prefetching in a texture cache architecture," in *Proc. SIGGRAPH/EUROGRAPHICS Workshop Graphics Hardware*, 1998, pp. 133–142.

[15] A. Vartanian, J.-L. Bechennec, and N. Drach-Temam, "Evaluation of high performance multicache parallel texture mapping," in *Proc. 12th Int. Conf. Supercomputing*, 1998, pp. 289–296.

Hyun-Chul Shin received the B.S. degree in electronics engineering from Pusan National University, Pusan, Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering and computer science from Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea, in 1998 and 2004, respectively.

He is a Senior Engineer at LG Electronics Company, Seoul, Korea. His research interests include multimedia VLSI design.

Jin-Aeon Lee received the B.S. degree in electronics engineering from Hanyang University, Seoul, Korea, in 1989, and the M.S. and Ph.D. degrees in electrical engineering and computer science from Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea, in 1994 and 2000, respectively.

He is a Principal Engineer at Samsung Electronics Company, Suwon, Korea, where he is responsible for Mobile Multimedia IP (2D/3D Graphics engines and Video Source coding IPs) developments.

Lee-Sup Kim (SM'05) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1982 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1986 and 1990, respectively.

He was a Post-Doctoral Fellow at the Toshiba Corporation, Kawasaki, Japan, during 1990–1993, where he was involved in the design of the high-performance DSP and single chip MPEG2 decoder. He has been at the Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea, since March 1993. In November 2002, he became a Full Professor. His research interests include multimedia VLSI design, hardware implementation of signal processing algorithms, and low-power IC design.