Footprint Area Sampled Texturing

Baoquan Chen, *Member*, *IEEE Computer Society*, Frank Dachille, and
Arie E. Kaufman, *Fellow*, *IEEE*

Abstract—We study texture projection based on a four region subdivision: magnification, minification, and two mixed regions. We propose improved versions of existing techniques by providing exact filtering methods which reduce both aliasing and overblurring, especially in the mixed regions. We further present a novel texture mapping algorithm called FAST (Footprint Area Sampled Texturing), which not only delivers high quality, but also is efficient. By utilizing coherence between neighboring pixels, performing prefiltering, and applying an area sampling scheme, we guarantee a minimum number of samples sufficient for effective antialiasing. Unlike existing methods (e.g., MIP-map, Feline), our method adapts the sampling rate in each chosen MIP-map level separately to avoid undersampling in the lower level l for effective antialiasing and to avoid oversampling in the higher level $l+1$ for efficiency. Our method has been shown to deliver superior image quality to Feline and other methods while retaining the same efficiency. We also provide implementation trade offs to apply a variable degree of accuracy versus speed.

Index Terms—Texture mapping, antialiasing, anisotropic filtering, forward mapping, backward mapping, footprint area sampling, hardware.

1 Introduction

High quality and efficiency are usually conflicting goals in texture mapping. Many existing methods [2], [14], [22] focus on delivering high quality and deemphasize efficiency. These methods tend to consider either the exact contribution of each texel or the exact filtering of each pixel. (We call these techniques "exact antialiasing techniques.") Although exact texture filtering methods tend to generate high image quality, they are not perfect; a problem especially occurs for the *mixed regions* (to be defined below). In this paper, we present a new framework to elucidate the deficiency of existing exact texturing methods and provide solutions for them. Another severe problem with the exact texture filtering method is its arbitrary cost since the filter for an infinite horizontal plane perpendicular to the image plane must consider an infinite number of texels. A solution to this is to prefilter the textures and conduct the texel convolution or pixel filtering in some level of the prefiltered textures rather than in the original textures. In fact, since only discrete levels are pregenerated, two levels of the prefiltered textures are chosen for computing the fractional level in between the chosen two levels. While the existing prefiltering methods deliver better efficiency than the exact filtering methods, they compromise on the image quality. In this paper, we further offer a novel algorithm to improve the quality of the latest prefiltering method while maintaining its efficiency.

Exact texture filtering methods compute the closest estimate of the projection shape of texels/pixels from one image domain to another—texture or screen image. The elliptical weighted average (EWA) filter methods [12] approximate the projection of a pixel circular footprint in texture space with an ellipse. In Feibush et al.'s method [7], a square-shaped pixel is backward projected to a quadrilateral in texel space. Glassner [11] adaptively approximates this backward project shape using Crow's [5] summed-area table. Gangnet et al.'s method [9] uses the major axis to determine the supersampling rate for both axes and average the samples for each pixel. While the above methods emphasize a backward projection, there are other methods featuring a forward projection [4]. Ghazanfarpour and Peroche [10] forward project each texel onto the screen. At each pixel, a circular footprint with a Gaussian profile is used to filter projected texels. Since a texture image is two-dimensional, texels can be minified in one axis, yet magnified in the other axis. Thus, a texture mapped image can be subdivided into four regions based on the minification factor in the two axes. The minification factor determines and measures minification and magnification, expressed in *texels per pixel*. If minification is greater than one texel/pixel, we call it *minification*; otherwise, it is termed *magnification*. Fig. 1 delineates four regions: The cyan region is the *magnification region* having magnification in both axes, the green and blue regions are the *mixed regions* having minification in one axis and magnification in the other, and the red region is the *minification region* having minification in both axes. We refer to the axis with the smaller minification factor as the *minor axis* and the other as the *major axis*. The ratio between the major and minor axes is called the *anisotropic ratio* or *eccentricity*. Areas with the same $texel/pixel$ value in both axes are isotropic; otherwise, they are anisotropic because of the oblique projection of textures. Most previous texture mapping methods simply assume that a texel is either minified or magnified and do not specifically address the special anisotropic characteristics of the mixed regions—a combination of minification and magnification.

For example, EWA creates holes for pixels in the mixed regions. A higher quality EWA method [15] increases the elliptical pixel footprint axes by 1 unit, convolving more

- *B. Chen is with the Department of Computer Science and Engineering, University of Minnesota at Twin Cities, 4-192 EE/CS Bldg, 200 Union St. SE, Minneapolis, MN 55455. E-mail: baoquan@cs.umn.edu.*
- *F. Dachille and A.E. Kaufman are with the Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400. E-mail: frank@visitronix.net, ari@cs.sunysb.edu.*

Manuscript received 17 July 2002; accepted 2 Dec. 2002.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 116974.

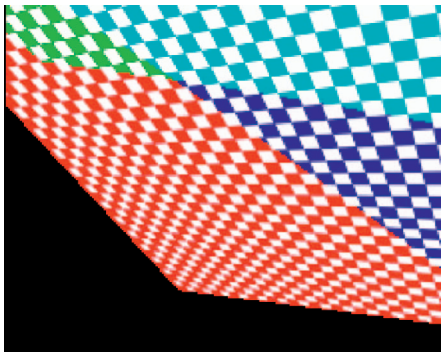

Fig. 1. Four regions of a texture mapped checkerboard plane: magnification (cyan), minification (red), and mixed regions (green and blue).

texels. The side effect of enlarging the axes is overblurring along the major axis (minification direction). Similar to EWA, the backward mapped quadrilateral of Feibush et al. and Glassner's methods may cover no texels in magnification and mixed regions. For that reason, it is suggested by Feibush et al. that, for the magnification region, the pixel color be computed using backward bilinear interpolation. However, for the mixed region, this would cause aliasing for the minification direction. Gangnet et al.'s method oversamples along the minor axis, which worsens in mixed regions. Ghazanfarpour and Peroche's method, however, generates holes in mixed regions. They thus propose to check for holes and perform backward bilinear interpolation. Yet, this may cause aliasing for the minification direction just as in the method of Feibush et al. All these methods, having arbitrary cost, create an inaccurate textured image by either introducing holes, blurring, or aliasing. Here, we elucidate the deficiencies of existing antialiasing methods by examining them in the context of the four region subdivision. We further contribute by proposing improved versions of the existing methods by solving the problem of holes in mixed regions (Section 2). For the most part, only a small "fix" is needed to make a significant improvement on the visual quality. Section 2 also provides a framework to evaluate and guide new texture mapping algorithm design, as the one proposed in this paper (Section 3).

Exact texture filtering is arbitrarily costly since the filter for an infinite horizontal plane perpendicular to the image plane must consider an infinite number of texels. Prefiltering methods solve this by prefiltering the textures into discrete levels and choose the most appropriate level(s) for texturing. For example, the MIP-map method [20] trades memory (a prefiltered texture pyramid of about one-third more storage) for constant time and isotropic filtering that trades reduced aliasing for increased blurring. Because of the isotropic filtering, MIP-mapping exhibits very poor behavior in the anisotropic transformation regions. To remedy this problem, the NIL-maps method [8] provides a technique to approximate a space-variant filter kernel surface in texture space using a sum of suitably chosen basis functions. The convolution of these functions with a texture can be precomputed and stored in a pyramidal structure called NIL-maps. Although the pyramid structure helps to bound the computation cost, this method is still considered expensive. On average, an arbitrary filter requires a large number of basis functions to approximate. Heckbert further suggested the integration of an EWA implementation with a MIP-map image pyramid [15]. Here, the minor rather than major axis of the projected ellipse (or parallelogram approximation) determines the MIP-map level on which the EWA filter is then applied. Both NIL-maps and MIP-map EWA generate superior quality than that of MIP-map. However, they are still too costly to use because they both strive to approximate the exact filter shape in texture space. The footprint assembly algorithm [17] strives to achieve anisotropic filtering through multiple isotropic filters. Rather than sampling according to the exact elliptic footprint shape, each pixel is supersampled along its major axis direction, using the sample rate determined by the eccentricity value. Similar to MIP-map EWA, the minor axis determines the MIP-map level. Footprint assembly provides higher visual quality than the MIP-map method, but less complexity than NIL-maps and MIP-map EWA. However, to facilitate hardware implementation, footprint assembly simplifies the approximation of ellipse parameters, leading to noticeable artifacts. A recent method, called Feline [16], aiming to improve the footprint assembly, optimizes the computation of the major axis direction, the eccentricity value, and the sample distribution. Feline represents the most recent technique which can be efficiently implemented by hardware and delivers high visual quality. Nevertheless, Feline is still far from alias-free. As the number of samples used in Feline is computed to guarantee sufficient sampling on the fractional level of MIP-maps, however, this sampling rate will lead to undersampling in the chosen lower level (l), but oversampling in the chosen higher level ($l+1$). In the second part of this paper, we present a solution to guarantee minimum but sufficient sampling rate at both levels for effective antialiasing. Significantly, we employ an area sampling scheme and adapt the number of samples separately at the two chosen MIP-map levels. Integrating with several other techniques such as utilizing the coherence between neighboring pixels, we present a novel method called FAST (Footprint Area Sampled Texturing) to address issues of processing efficiency, image quality, and the feasibility of hardware implementation (Section 3).

2 Exact Antialiasing Techniques

The key issue for antialiased texture mapping is for each screen pixel to determine the contributing texels and to convolve the texels with a proper filter to obtain the final pixel color. This filtering can be applied in either screen space on pixels using forward mapping or texture space on texels using backward mapping. We present a framework that allows either forward or backward mapping using both footprint or point projections. We analyze the behavior of these methods using the four region subdivision, especially in the critical mixed regions of Fig. 1. The fundamental problem of these regions can be illustrated by a simple example of image scaling. When an image is scaled without ratio constraint, it can be magnified in one direction but minified in the other direction. Here, we use an example of scaling a 5×5 resolution image to a resolution of 9×3. As a convention in the following illustrations, all pixels and texels are placed on the grid points. We also assume a circular footprint.

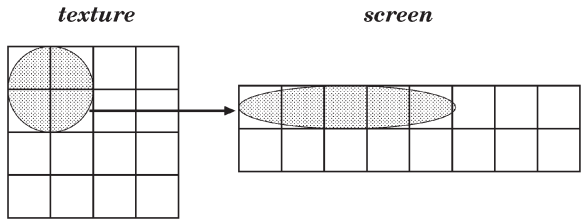

Fig. 2. Forward projected texel footprint may cover no pixels in naive forward footprint projection methods.

2.1 Forward Footprint Projection

First, we consider each texel as a circle and apply forward projection, creating a screen space conic. We then scan convert the conic, convolving the energy of the texel with a filter. This effectively "splats" the energy of the texel to the pixels within its projected footprint. In the magnification and mixed regions, the entire conic may actually fall in between the pixels, as shown in Fig. 2. We propose to expand the conic so that it splats to at least one pixel by clamping the axes and diagonals to a minimum of one pixel unit. Similarly, Swan et al. [18] successfully applied this clamping on the projected footprint of each voxel in 3D volume rendering by splatting. However, they have only applied it to the minification region, without explicitly mentioning the mixed region. We propose that this clamping must be conducted separately for each axis. The effectiveness of this approach has been recently demonstrated by Zwicker et al. [23] in volume rendering.

Computing and evaluating a conic or even the approximated ellipse shape are expensive operations, as well as convolving texel energy. Thus, this method has practical limitations.

2.2 Forward Point Projection

Second, we consider each texel as a point and project it into screen space. A filter is placed at each pixel for filtering the projected texels. This is accurate in the minification region as no footprint is ever approximated. However, holes may appear in the magnification and mixed regions. As can be seen in Fig. 3, no texels are projected into pixel p footprint, thus a hole appears at pixel p. We propose to supersample in texture space along the magnification axis and forward project subtexels into screen space. Fig. 4 shows the problem with the approach of Ghazanfarpour et al. [10] for filling holes and the effectiveness of our proposed solution using only a simple Bartlett filter in screen space.

2.3 Backward Footprint Projection

Third, we consider each screen pixel as a circle and back project it into texture space, where it becomes a conic. We

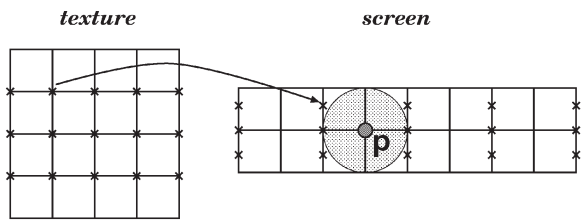

Fig. 3. No texel may be projected into a pixel footprint in naive forward point projection methods.

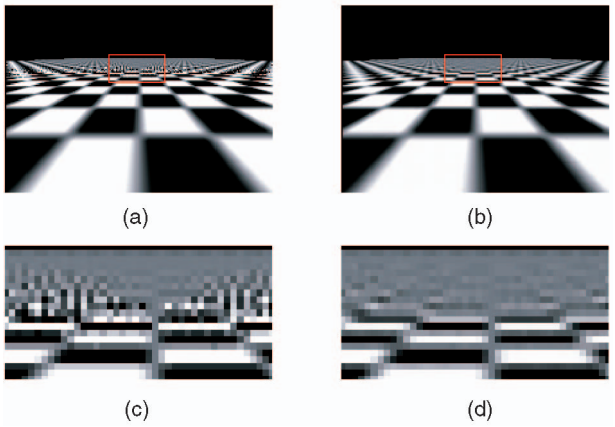

Fig. 4. Forward point projection texture filtering with (a) backward bilinear interpolation to fill in the holes [10] (notice aliasing due to subsampling in one direction) and (b) our solution with supersampling in texture space to fill in the holes. (c) and (d) are zoom-ins of the marked rectangles of (a) and (b), respectively.

then approximate the conic by an ellipse and scan convert it in texture space, convolving it with a projected filter kernel, similar to EWA. As shown in Fig. 5, in the mixed regions, the backward mapped footprint may cover no texels, leaving a hole at that pixel. We propose to clamp the minimum length of the ellipse axes and diagonals to one texel unit in order to guarantee texel coverage without blurring. This is superior to Heckbert's solution of unilaterally increasing the pixel footprint axes by one texel unit. Fig. 6 shows overblurred EWA (in the minification axis) and the less blurred result of our method.

2.4 Backward Point Projection

Fourth, we consider each screen pixel as a point and project it into texture space. An interpolation is conducted at the projected location to obtain the color value. This method is naturally free of holes.

Because of the sampling rate difference in screen and texture image space and the perspective projection, screen pixels may undersample the texture image, causing an unappealing visual effect of aliasing. For antialiasing, two methods are possible. The first is called a *prefiltering* of the texture image according to the sampling rate of the screen image. MIP-mapping is a practical prefiltering method, which will be reviewed in the next section. Another method is called *postfiltering*, which supersamples the screen image and then filters on subpixels to obtain the final pixel value. Hardware accelerated texture supersampling was used in the accumulation buffer [13]. However, the supersampling rates are globally specified without adapting them to the

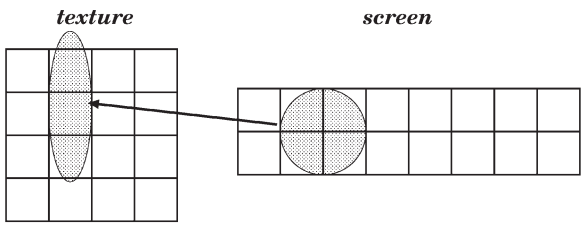

Fig. 5. The backward projected footprint may cover no texels in naive backward footprint projection methods.

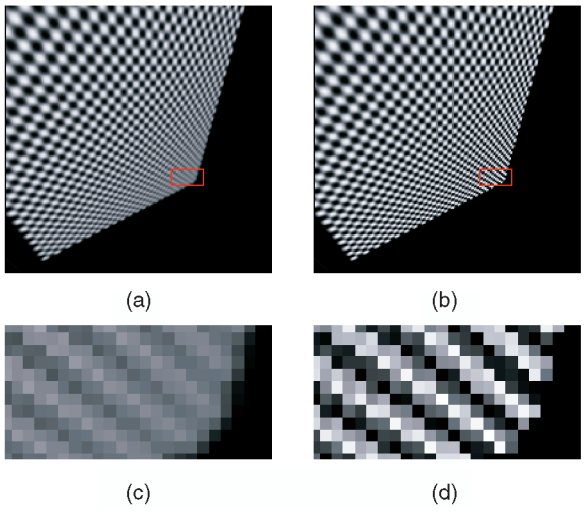

Fig. 6. Solutions to the undersampling problem in backward footprint projection: (a) overblurred EWA [15] and (b) out clamping method. (c) and (d) are zoom-ins of the marked rectangles of (a) and (b), respectively.

local minification. The method of Gangnet et al. [9] is adaptive, but, for every pixel, it oversamples along the minor axis (see Fig. 7). This leads to extra computation, especially in the mixed regions. To alleviate this problem, we propose *adaptive sampling* to adapt the sampling rate in each direction separately (see Fig. 14). Visual differences are not apparent between the oversampling of Gangnet et al. and our more efficient adaptive sampling method.

A filter size of at least one unit radius is necessary for effective antialiasing. For some filters, such as the Gaussian filter, an even larger filter size is required [22]. This means the footprints of neighboring pixels overlap each other. The overlapped region is sampled multiple times if using a straightforward adaptive sampling method. In the next section, we introduce a novel backward method which guarantees a minimum but efficient number of samples for effective antialiasing.

3 The FAST Algorithm

Current hardware implementations favor backward mapping. It has the important property of processing in raster scanline order, which is compatible with the hardware implementation of other pipeline stages. Our straightforward adaptive sampling method, described in Section 2.4, promises high quality, but could be very inefficient.

We propose a method called FAST which addresses issues of processing efficiency, texel memory bandwidth, and image quality. We achieve our goals by utilizing a number of techniques:

1. Prefilter using MIP-maps to lower the cost of each pixel. Differing from the existing methods, we adapt the sampling rate in each chosen MIP-map level separately to avoid undersampling in level l for effective antialiasing and oversampling in level $l+1$ for efficiency.
2. Provide a scheme called footprint area sampling to determine the number of samples.
3. Utilize the coherence between neighboring pixels to guarantee a minimum number of subpixel samples, which also minimizes the number of texel accesses.
4. Provide a finer trade off between image quality and efficiency by clamping the maximum number of samples for each pixel.

In the following section, we discuss the previous prefiltering methods, point out the defects of these approaches, and, finally, propose our solution.

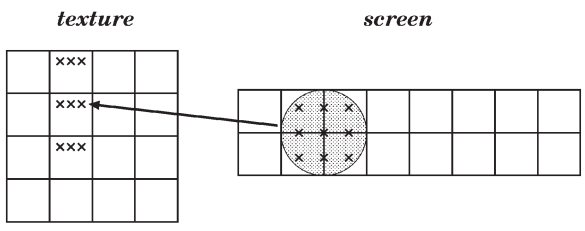

Fig. 7. The minor axis is over supersampled in naive backward point projection methods.

3.1 Prefiltering

The most popular method of prefiltering is the MIP-map method which builds up in preprocessing the different levels of representation of the texture image. Starting from the original image, called level 0, this method filters the input image to form a new image of half resolution, called level 1. This procedure continues on level 1 until the image resolution reaches 1×1. In texture filtering, the pixel is projected to the texture space and the maximum minification factor of the two axes is calculated. A pyramid level l image is selected that contains the prefiltered texel data for a minification ratio nearest to that calculated. A trilinear interpolation is performed by performing first bilinear interpolation in both level l and level $l+1$ image space and then a linear interpolation on the obtained two pixel colors to get the final color. The linear interpolation weight is computed based on the fractional level f calculated from the maximum minification factor. This final linear interpolation softens the transition between different levels. MIP-mapping is illustrated in Fig. 8a. Level selection, especially its implementation in hardware, is discussed in detail in [6].

As a known problem, MIP-map prefiltering and the trilinear interpolation are based on an isotropic square filter shape. The shape of the filter is always rectangular, but the size is space-variant. However, the mapping of a pixel can be anisotropic. Therefore, such an isotropic filter causes blurring in one axis while reducing aliasing in the other.

Heckbert [15] has proposed a 4D image pyramid that extends in the two axes to approximate arbitrary rectangles. However, this leads to a higher storage requirement. Yet, the improvement over the MIP-map pyramid is limited because a rectangle still cannot approximate arbitrarily aligned shape. Recently, Blythe and McReynolds [3] introduced an anisotropic texture filtering technique which filters the texture anisotropically so that the filtered MIP-maps aspect ratio approximates the projected aspect ratio of the geometry. In essence, this approach is similar to Heckbert's approach [15]. A number of disadvantages are inherent to this approach. As the aspect ratio varies from polygon to polygon even on a single model, it also changes over time for each polygon when the view changes, it

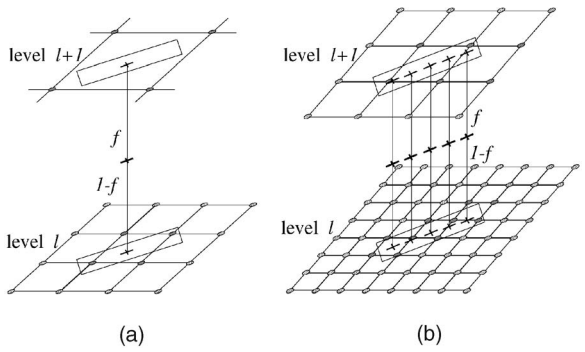

Fig. 8. Prefiltering methods: (a) MIP-mapping; (b) Footprint Assembly/ Feline.

requires that either a prohibitive set of resampled MIP-maps with different aspect ratios have to be generated in advance, resulting in excessive memory expenses, or the MIP-maps have to be regenerated on the fly, resulting in extravagant computation cost. Furthermore, because the anisotropic filtering is conducted only in the two major axes of the texture image, generated anisotropic MIP-maps will thus never match exactly with the arbitrary aspect ratio of the projected geometry.

The footprint assembly method accommodates the space variance by performing multiple trilinear filter operations along the direction of anisotropy. A pixel area, one-unit square centered at the pixel, is projected to the texture space and the major and minor axes are calculated to approximate the parallelogram projection shape. Each pixel is super-sampled along its major axis direction. Here, the minor instead of major axis determines the MIP-map level. The eccentricity determines the number of samples along the major axis. Fig. 8b illustrates the principle of this filtering operation. Feline improves the footprint assembly in a few effective ways by:

1. using a minimum of one-unit radius sphere (corresponding to a two-unit square) to approximate a pixel so that the adjacent footprints have enough overlap to overcome aliasing,
2. approximating the major axis as the major axis of the projected ellipse rather than the parallelogram,
3. allowing the number of samples to be any integer rather than 2^n, and
4. weighting the samples using a Gaussian curve rather than simply averaging them.

More details are given in reference [16]. Fig. 9 shows the difference between footprint assembly and Feline on the filter shape and the samples used. It is claimed that Feline achieves higher visual quality than footprint assembly with little additional cost.

However, Feline is not alias-free. While Feline guarantees sufficient sampling rate for the real (fractional) level of a pixel, it ends up undersampling in the chosen lower level (l), but oversampling in the chosen higher level ($l+1$). This behavior is inherent to the basic filtering element of trilinear interpolation that Feline uses, which presents a problem even for an isotropic projection. As an example, we want to scale an image down three times in each dimension. We first precompute the MIP-maps of the image. According to MIP-mapping, we choose levels 1 and 2 for trilinear interpolation. For each target pixel, we sample in each level and then average the results. The sample points in level 1 are 1.5 unit distance away, indicating an undersampling that causes aliasing. This aliasing effect remains after the interlevel linear interpolation. This is more clearly illustrated in the frequency domain, shown in Fig. 10. Fig. 10a is the spectrum of the original analog signal. Since MIP-maps are created through low-pass filtering, the spectrum becomes narrower when the MIP-map level increases. When the sampling rate in level l is insufficient, spectra overlap, an indication of aliasing (Fig. 10b). In level $l+1$, the sampling rate is higher than the Nyquist rate, thus the spectra are further separated apart, as in Fig. 10c. Fig. 10d shows the spectrum after the linear interpolation, indicating that aliasing remains in the final result. In addition, Feline, similar to many other methods such as EWA, has footprint

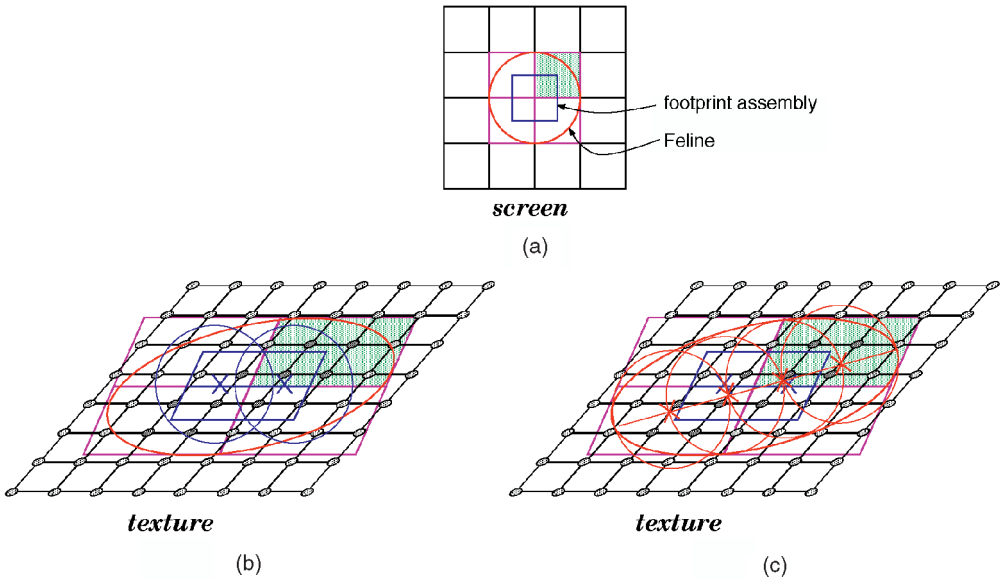

Fig. 9. Comparison between Footprint Assembly and Feline: (a) screen space, (b) texture space of Footprint Assembly, (c) and Feline.

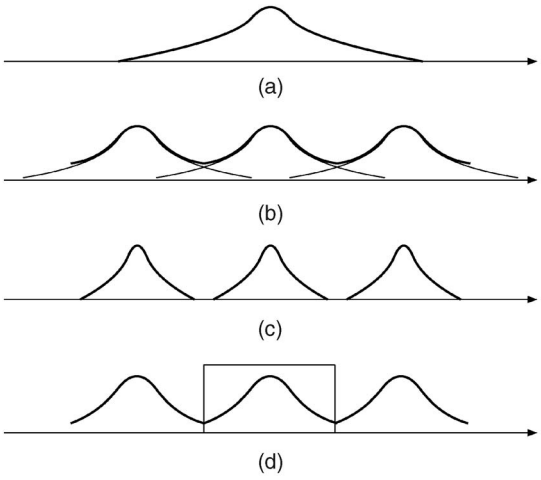

Fig. 10. Spectrum of (a) original signal, (b) sampling in level l, (c) sampling in level $l+1$, (d) linear interpolation between two levels ((b) and (c)).

overlap between adjacent pixels, resulting in sampling the texture multiple times—a redundancy increasing the computation cost and texel access bandwidth. The basic idea of our method is to adapt sampling rate in each level *separately*. This avoids undersampling in the chosen MIP-map level l for effective antialiasing and oversampling in level $l+1$ for efficiency, thus minimizing the number of samples for effective antialiasing. Yet, to reduce the computation cost and the texel access bandwidth, we utilize the coherence between adjacent pixels by computing only a quarter of this square and "splat" the result to the pixels whose footprints overlap with this quarter (shaded green square in Fig. 9a). We discuss this technique immediately.

3.2 Scanline Coherence

When the pixel coverage overlaps, subpixels between neighboring pixels contribute to all surrounding pixels. For example, in Fig. 11a, subpixels within the square region $aefg$ contribute to pixels a, e, f, and g. The naive implementation of our adaptive sampling method (Section 2.4) calculates the subpixels of each pixel separately, which means that the subpixels are calculated four times. This requires not only costly computation, but also a high memory bandwidth. To address this issue, coherence between neighboring pixels must be exploited. Texel access coherence has been used in hardware design [21] by caching the most recently used texels. We want to push this coherence utilization to its limit. Generally, pixel coherences exist between neighboring pixels within and between scanlines.

Fig. 11a shows our scanline coherence. Each subpixel is calculated only once and the RGB colors are splatted to the four neighboring pixels. The contribution weight for each neighboring pixel is determined by the filter used. Fig. 11b shows a Gaussian filter implementation with a circular footprint coverage of one unit radius. The filter weight of a subpixel to a pixel is precomputed and indexed by the subpixel's coordinate offsets $(\triangle x, \triangle y)$ to the pixel. Since the filter is implemented with a table lookup, we can use a filter of any kind. The contribution weights of all subpixels are accumulated in each pixel for the final normalization. As an example, for pixel a in Fig. 11, the subpixels in the top-right shaded square region are processed and their values are distributed ("splatted") to the four neighboring pixels a, e, f, and g. Upon finishing processing this quarter, the remaining three quarters of the pixel coverage have already been calculated and accumulated by pixel b, the neighboring pixel in the same scanline, and c and d, neighboring pixels in the previous scanline. Thus, for pixel a, we can sum up the accumulated partial color of the first three quarters and that of the current quarter and then normalize the sum by the accumulated weight as the final color.

To hold the partial contribution of all subpixels during the scanline processing, intuitively, two scanline buffers are required: one for the current scanline L_i and another for the next scanline L_{i+1}. But, for a compact implementation, a scanline buffer of size $length+1$ working as a FIFO is sufficient, where $length$ is the length of the scanline. Once the final colors are produced for pixels in scanline L_i, their memory can be immediately released for storing the partial colors of pixels in scanline L_{i+1}. Each pixel of the scanline buffer stores the accumulated color RGB and weight. The (u, v) coordinates in texture space are also stored in the buffer. This is used for pixel vector calculation, to be discussed in the next section. Since the operations are conducted in two MIP-map levels, hence, two sets of buffers are needed, resulting in two scanline buffers. This is still feasible for hardware implementation if we use bucket

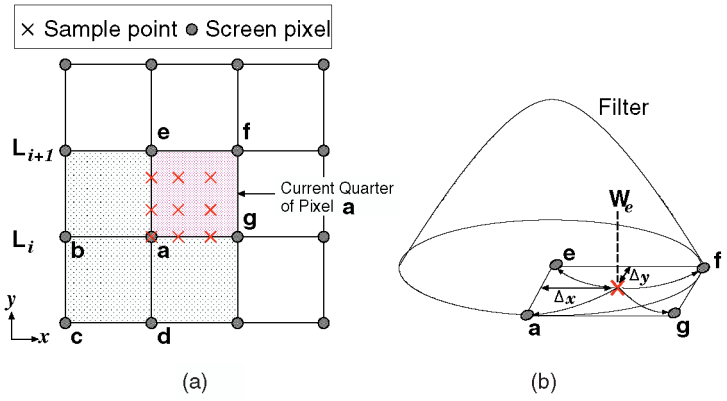

Fig. 11. Utilizing coherence between scanlines and neighboring pixels.

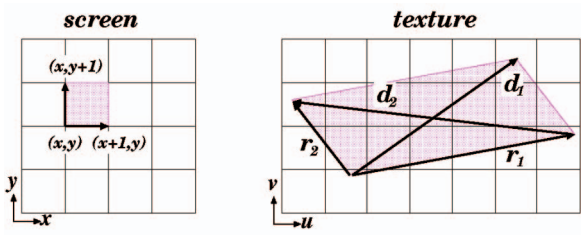

Fig. 12. Pixel quarter area projection.

rendering [19], in which the framebuffer is subdivided into coherent regions that are rendered independently (also known as tiled rendering or chunking). Bucket rendering enables a deeper working set memory to be kept on chip.

3.3 Footprint Area Sampling

For our case, the processing area of each pixel (x, y) is the top-right square, called the *current pixel quarter* (shaded square in Fig. 12, also shaded green square in Fig. 9a). It is important to determine the minimum number of samples in each MIP-map level for effective antialiasing and to avoid supersampling. When the current pixel quarter is projected into texture space, the projected area can be approximated as a parallelogram (Fig. 12). In a backward footprint projection method, such as EWA, this approximated region decides the convolution texels; therefore, a parallelogram approximation is too crude there. However, for our situation, a parallelogram approximation is fairly sufficient because it is only used to determine the MIP-map level and the sample rate. The real filter coverage is defined by the filter itself and is conducted in screen space. Next, we discuss the pixel projection geometry and the schemes on determining the MIP-map level and sample rates. For clarity and the reader's convenience, we repeat some basic steps here, which have already been described in previous work.

The projection parallelogram is described by axis vectors r_1, r_2 and diagonal vectors d_1, d_2, where $d_1 = r_1 + r_2$ and $d_2 = r_2 - r_1$. The vector $r_1 = (\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x})$ approximates the movement in texture space of a unit x-step in screen space, while $r_2 = (\frac{\partial u}{\partial y}, \frac{\partial v}{\partial y})$ approximates a y-step.

Let F be the perspective transformation function from (x, y) screen coordinates to (u, v) texture coordinates:

$$u = F_u(x, y); \qquad v = F_v(x, y). \tag{1}$$

r_1 and r_2 can be represented as:

$$\begin{aligned} r_1 &= (F_u(x+1, y) - F_u(x, y), F_v(x+1, y) - F_v(x, y)) \\ r_2 &= (F_u(x, y+1) - F_u(x, y), F_v(x, y+1) - F_v(x, y)). \end{aligned} \tag{2}$$

Because the projections of neighboring pixels were previously computed and stored in the scanline buffers, we can apply (2) by taking the vectors to neighboring pixels in texture space which are already calculated [6]. Therefore, this method requires no extra Jacobian calculations as the other methods do [12], [17]. Special care needs to be taken for the boundary pixels to be able to utilize this method. For example, for the first row pixels, we have to compute the projections of an additional "previous" row of pixels and store them in the scanline buffer. Similarly, for the rightmost pixel of a scanline, we need to compute the projection of an additonal pixel to its right. However, note that, for these additional pixels, only their projections are calculated.

The magnitudes of these vectors are approximated with:

$$\begin{aligned} |r_1| &= max\left(\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial v}{\partial x}\right|\right) \\ |r_2| &= max\left(\left|\frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial y}\right|\right) \\ |d_1| &= max\left(\left|\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\right|\right) \\ |d_2| &= max\left(\left|\frac{\partial u}{\partial y} - \frac{\partial u}{\partial x}\right|, \left|\frac{\partial v}{\partial y} - \frac{\partial v}{\partial x}\right|\right). \end{aligned} \tag{3}$$

The magnitude of the minor vector m can be approximated by:

$$m = min(|r_1|, |r_2|, |d_1|, |d_2|). \tag{4}$$

The minor vector m is used to compute the MIP level l and the scaling factor f of level l:

$$l = \lfloor log_2 m \rfloor; \qquad f = 2^l. \tag{5}$$

The vectors r_1 and r_2 in level l are then scaled as:

$$r_1' = \frac{r_1}{f}; \qquad r_2' = \frac{r_2}{f}. \tag{6}$$

We then compute the "area" of the parallelogram and use it to determine the total number of samples in each pixel quarter as:

$$N = \lceil |r_1' \cdot r_2'| \rceil = \left\lceil \frac{|r_1 \cdot r_2|}{f^2} \right\rceil. \tag{7}$$

Hence, we have the name area sampling. This is similar to NIL-maps [8] in determining the initial number of patches (for approximating an arbitrary filter kernel surface); the difference is that here all samples are in the same MIP-map level. Once the total number of samples is determined, we then distribute these samples evenly in the pixel quarter using the jittered sampling scheme.

We must consider the special situation over the mixed region, where the minor axis of r_1' and r_2' is less than 1.0 (magnification). We must clamp it to 1.0 before we apply (7) so that we guarantee sufficient sampling in the pixel quarter area. For example, if $r_1' \perp r_2'$ and $|r_1'| = 10.0$ and $|r_2'| = 0.1$, the result of applying (7) is $N = 1$, while the required sampling rate is really 10. We call this clamping *the mixed region correction*.

In level $l + 1$, the number of samples can be calculated from the scaled vector of r_1 and r_2 in level $l + 1$, as in (7). However, it can be simply calculated as:

$$N' = \left\lceil \frac{N}{4} \right\rceil. \tag{8}$$

Fig. 13 shows an example of determining the number of samples for Feline (Fig. 13a) and our FAST area sampling method (Fig. 13b). In this example, $|r_1| = 16.0$ and $|r_2| = 2.4$; the two vectors are perpendicular to each other. For the Feline method, the number of trilinear sample points along the anistropic axis is determined by the anisotropic ratio between the two vectors, which is rounded to 7. In our

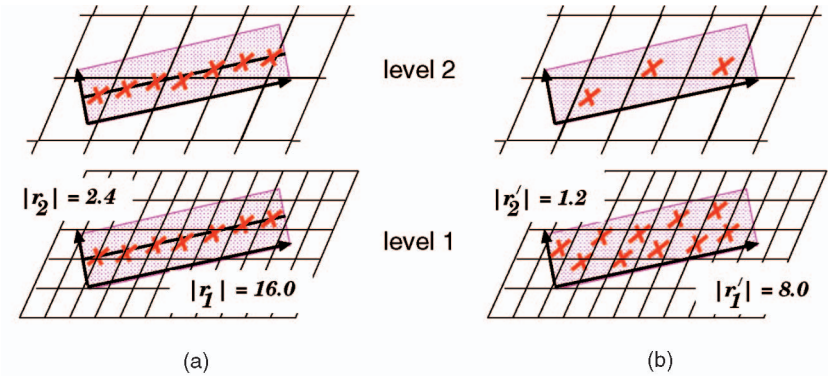

Fig. 13. (a) Feline versus (b) our FAST area sampling.

FAST area sampling method, the minor axis r_2 is used to determine the MIP-map level (5), which is 1 for this case. In level 1, the scaled two vectors become $|r_1'| = 8.0$ and $|r_2'| = 1.2$, respectively. The total sampling rate is determined as $N = \lceil 8.0 \times 1.2 \rceil = 10$. The number of samples in level 2 is $N' = \lceil \frac{10}{4} \rceil = 3$. In total, Feline spends 14 samples for the two MIP-map levels, while FAST spends 13 samples.

For highly anisotropic pixels, the supersampling rate for the pixel quarter can be quite high. To bound the processing time of each pixel, we need to clamp the sampling rate to a prespecified value N_c. Simply doing this leads to aliasing due to undersampling in the area. Our solution is to trade off blurring with aliasing by increading the MIP-map level. The new MIP-map level l' and the scaling factor f' are computed as:

$$l' = \left\lfloor log_2 \frac{r_1 \cdot r_2}{N_c} \right\rfloor; \qquad f' = 2^{l'}. \tag{9}$$

This method provides a trade off between image quality and efficiency. The total number of samples that we really use in level l', $\overline{N}$, can be calculated based on the newly calculated f' from (7). The number of samples in level $l' + 1$ is then $\frac{\overline{N}}{4}$.

4 Results and Discussions

To evaluate our FAST area sampling method, we compare it with existing MIP-map and Feline methods. We have rendered a room scene using the different methods. The scene consists of five large texture mapped polygons representing walls, a floor, and ceiling. Each texture image has different features. The floor image is the typical checkerboard benchmark image for texture mapping. The roof image is high frequency line stripes. The right wall image is full of text, which is also challenging. The left wall image is a mandrill image. The back wall image is a typical camera shot conveying a meaningful scene. All images have a resolution of 256×256. The resulting image resolution is 240×240. The program runs on an SGI O2 with an R10000 CPU and 128MByte memory.

We first compare the quality among these methods. Figs. 14a-i show a still frame from an animation. The traditional MIP-map method creates the lowest quality since only a single trilinear interpolation and an isotropic filter is implemented (Fig. 14a). All images are blurry and the checkerboard image demonstrates noticeable aliasing. Straightforward implementation of the adaptive sampling method creates the highest quality image with less blurring and better antialiasing (Fig. 14b). Utilizing coherence only in the adaptive sampling method (without prefiltering) promises the same high quality (Fig. 14c). Feline delivers better quality than MIP-mapping by introducing anisotropic filtering (Fig. 14d). This can be observed from the text image, but aliasing is still present in the checkerboard image. By setting the clamping factor to 4, the quality is similar (Fig. 14e). However, by further reducing the clamping factor to 2, the image becomes blurrier (Fig. 14f). Our FAST method, with coherence utilization, prefiltering, and the footprint area sampling scheme, creates images of indistinguishable quality as compared to straightforward adaptive sampling (Fig. 14g). By setting the clamping factor to 4, the quality is similar (Fig. 14h). However, by setting the clamping factor to 2, the image becomes a bit blurrier and more aliased (Fig. 14i). The difference can be seen more clearly from zoom-ins in Figs. 14j-l. The quality of animations is consistent with that of the extracted frames for each method.

In creating images using FAST, we have used a Bartlett filter to filter subpixels. We have also used a Bartlett filter with 4×4 coverage to build up all MIP-maps, instead of a 2×2 box filter as typically used in the MIP-mapping method, we argue that using only a 2×2 box filter causes aliasing in MIP-maps because of its insufficient filter size.

In practice, we can specify the clamping value per primitive based on the texture content. For high frequency textures, a higher clamping value is required for effective antialiasing; while, for low frequency textures, such as the back wall image, no artifacts are noticeable even if the clamping value is set to 2. Our experiments show that clamping to 4 is usually a good compromise. Clamping to 6 is normally sufficient for high quality.

Next, we compare the speed among the methods. The common processing functions are shared among the methods for fair comparison. The rendering time for each of the three methods—MIP-map, Feline, and FAST—is recorded and plotted in Fig. 15. As can be seen from the graph, MIP-mapping is always the most efficient, because it spends only one trilinear interpolation for each pixel, invariant of the region. It is also clear from the graph that MIP-mapping has a property of constant cost. However, this constant time is belied by the image quality (Fig. 14a).

FAST and Feline are very similar in rendering speed. It can also be seen in Fig. 15 that these two methods are sensitive to the camera position because they both try to

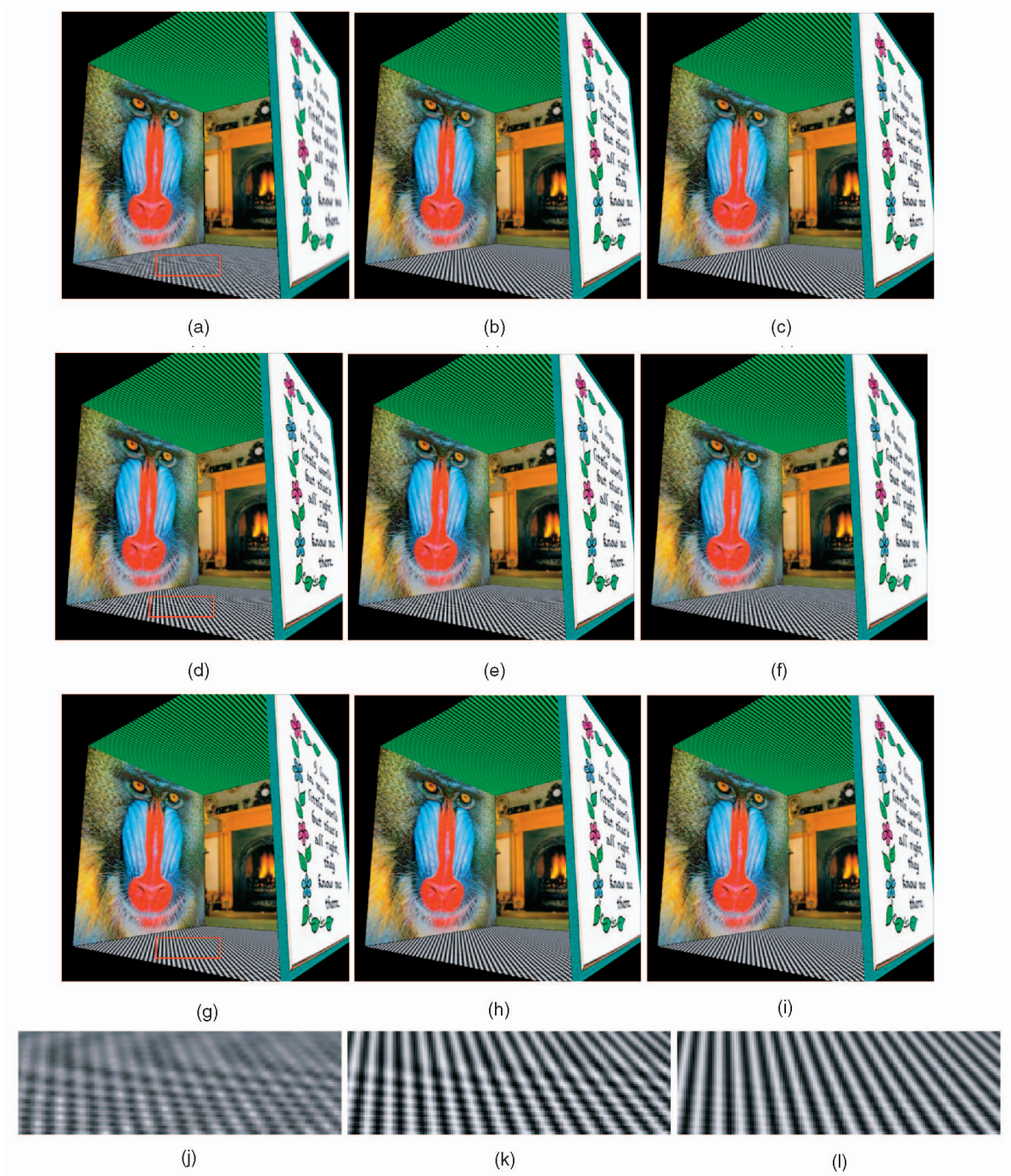

Fig. 14. Texture mapping using (a) traditional MIP-map, (b) adaptive sampling without coherence, (c) adaptive sampling with coherence, (d) Feline without clamping, (e) Feling with clamp = 4, (f) Feline with clamp = 2, (g) FAST without clamping, (h) FAST with clamp = 4, (i) FAST with clamp = 2. (j), (k), and (l) are zoom-ins of the marked racteangles of (a), (d) and (g), respectively.

adapt the number of samples according to the local minification property of each pixel. Table 1 shows the average rendering time per frame for each method. Here, we compare with two additional methods: straightforward adaptive sampling with and without coherence. We can see that the adaptive sampling ("Adaptive") method is the most inefficient. Utilizing coherence ("Coherence") leads to more than twice the speedup. When decreasing the clamping value, we can see that the rendering time of Feline decreases faster than the FAST method. This is because, in Feline, decreasing the clamping value directly reduces the anisotropic ratio and, thus, the sample rate. But, for FAST, decreasing the clamping value may not lead to a higher MIP-map level selection; therefore, the total number of samples may stay the same.

Another property we evaluated is the number of samples processed for each frame, which is plotted in Fig. 16. Since each trilinear interpolation requires twice the number of texels as bilinear interpolation, we normalized a trilinearly interpolated sample as two bilinearly interpolated samples. MIP-mapping spends two samples per pixel for the regions with minification in at least one axis. In the 110 frame

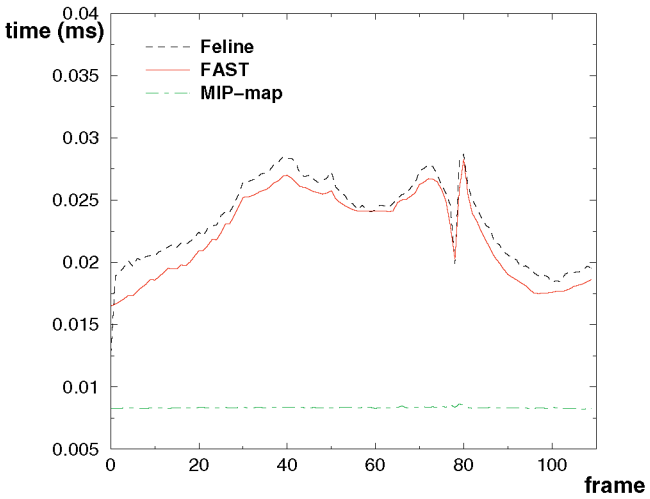

Fig. 15. Average rendering time per pixel (milliseconds) for 110 frames of a room scene animation.

animation, all pixels are in these regions, thus the curve for MIP-mapping stays completely flat. The graph illustrates that FAST utilizes slightly fewer samples than Feline, but, on average, they are very close (Table 1). The last column of Table 1 shows the required memory bandwidth relative to the MIP-map method. When the clamping value is 4 or higher, FAST only creates less than one time memory bandwidth overhead. This could be easily met by the current architecture design of managing memory bandwidth [1], such as capturing the memory bandwidth on chip, texel caching, and aggressive memory compression.

5 Concluding Remarks

We have proposed and evaluated different antialiasing methods for texture mapping based on a four region subdivision of the texture mapped area. Critical mixed regions are studied to challenge previous methods and guide our design. By utilizing coherence, prefiltering, and an area sampling scheme, our method FAST has achieved the following features:

- **High quality:** FAST delivers higher visual quality than MIP-map and Feline and almost indistinguishable image quality as compared to straightforward adaptive sampling.
- **Efficiency:** FAST maintains the same efficiency as Feline.
- **Low memory bandwidth requirement:** FAST requires memory bandwidth comparable to Feline.
- **Trade off between image quality and efficiency:** FAST can adjust the clamping value supporting a fine trade off between accuracy and speed. FAST still promises competitive image quality when reducing the clamping value, yet raising the efficiency.

Although we describe our methods in 2D, they can all be extended to 3D, especially benefiting problems such as 3D texture mapping, volume deformation, volume rendering, and hypertexturing. We plan to explore these in our future work.

From our experiments, we have observed that different texture images with different frequencies require different clamp values to obtain the same visual quality. We are currently investigating on adapting sample rate based on the local frequency of texture images.

TABLE 1
Comparison of the Performance of the Different Methods

Method	Clamp	Time (sec)	Samples per Frame	Relative Bandwidth
MIP-map	n/a	0.51	122,742	1
Adaptive	n/a	5.99	2,063,430	16.8
Coherence	n/a	2.24	456,840	3.7
Feline	none	1.37	343,104	2.8
FAST	none	1.32	250,318	2.0
Feline	6	1.27	309,250	2.5
FAST	6	1.27	236,298	1.9
Feline	4	1.22	292,724	2.4
FAST	4	1.24	227,276	1.9
Feline	2	1.06	242,613	2.0
FAST	2	1.10	201,093	1.6

Acknowledgments

This work was supported by US National Science Foundation grants MIP9527694 and CCR0306438, US Office of Naval Research grant N000149710402, and Intel. This work was conducted while Baoquan Chen was at Stony Brook University. Baoquan Chen would also like to acknowledge a Computer Science Department StartUp grant, Grant-in-Aid of Research, Artistry and Scholarship from the Office of the Vice President for Research and Dean of the Graduate School of the University of Minnesota. This work was

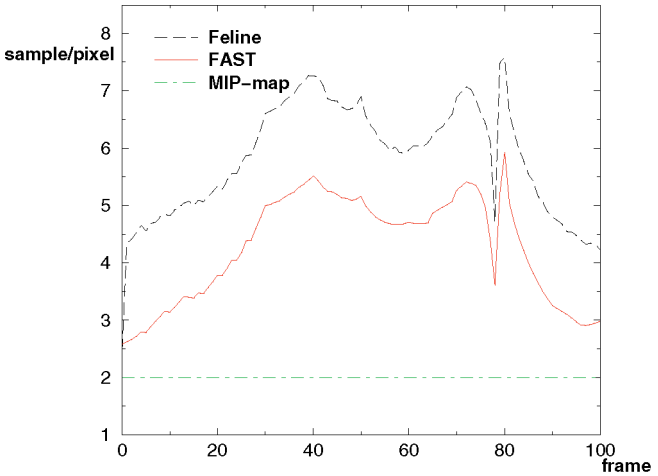

Fig. 16. Number of sample points per pixel for 110 frames of a room scene animation.

supported in part by the US Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAD19-01-2-0014, the content of which does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred.

REFERENCES

[1] A.C. Barkans, "High-Quality Rendering Using the Talisman Architecture," *Proc. 1997 SIGGRAPH/Eurographics Workshop Graphics Hardware,* pp. 79-88, Aug. 1997.
[2] J.F. Blinn and M.E. Newell, "Texture and Reflection in Computer Generated Images," *Comm. ACM,* vol. 19, pp. 542-546, 1976.
[3] D. Blythe and T. McReynolds, "Advanced Graphics Programming Techniques Using OpenGL," *SIGGRAPH '99 Course,* 1999.
[4] B. Chen, F. Dachille, and A. Kaufman, "Forward Image Warping," *Proc. IEEE Visualization '99,* pp. 89-96, Oct. 1999.
[5] F.C. Crow, "Summed-Area Tables for Texture Mapping," *Computer Graphics (SIGGRAPH '84 Proc.),* H. Christiansen, ed., vol. 18, pp. 207-212, July 1984.
[6] J.P. Ewins, M.D. Waller, M. White, and P.F. Lister, "Mip-Map Level Selection for Texture Mapping," *IEEE Trans. Visualization and Computer Graphics,* vol. 4, no. 4, pp. 317-329, Oct.-Dec. 1998.
[7] E.A. Feibush, M. Levoy, and R.L. Cook, "Synthetic Texturing Using Digital Filters," *Computer Graphics (SIGGRAPH '80 Proc.),* vol. 14, no. 3, pp. 294-301, July 1980.
[8] A. Fournier and E. Fiume, "Constant-Time Filtering with Space-Variant Kernels," *Computer Graphics (SIGGRAPH '88 Proc.),* vol. 22, no. 4, pp. 229-238, Aug. 1988.
[9] M. Gangnet, D. Perny, and P. Coueignoux, "Perspective Mapping of Planar Textures," *Computers and Graphics,* vol. 8, no. 2, pp. 115-123, 1984.
[10] D. Ghazanfarpour and B. Peroche, "A High-Quality Filtering Using Forward Texture Mapping," *Computers and Graphics,* vol. 15, no. 4, pp. 569-577, 1991.
[11] A. Glassner, "Adaptive Precision in Texture Mapping," *Computer Graphics (SIGGRAPH '86 Proc.),* vol. 20, pp. 297-306, Aug. 1986.
[12] N. Greene and P.S. Heckbert, "Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter," *IEEE Computer Graphics and Applications,* vol. 6, no. 6, pp. 21-27, June 1986.
[13] P. Haeberli and K. Akeley, "The Accumulation Buffer: Hardware Support for High-Quality Rendering," *Computer Graphics (SIGGRAPH '90 Proc.),* vol. 24, no. 4, pp. 309-318, Aug. 1990.
[14] P.S. Heckbert, "Survey of Texture Mapping," *IEEE Computer Graphics and Applications,* vol. 6, no. 11, pp. 56-67, Nov. 1986.
[15] P.S. Heckbert, "Fundamentals of Texture Mapping and Image Warping," MS thesis, Dept. of Electrical Eng. and Computer Science, Univ. of California, Berkeley, June 1989.
[16] J. McCormack, R. Perry, K.I. Farkas, and N.P. Jouppi, "Feline: Fast Elliptical Lines for Anisotropic Texture Mapping," *Computer Graphics (SIGGRAPH '99 Proc.),* pp. 243-250, Aug. 1999.
[17] A. Schilling, G. Knittel, and W. Strasser, "Texram: Smart Memory for Texturing," *IEEE Computer Graphics and Applications,* vol. 16, no. 3, pp. 32-41, May 1996.
[18] J.E. Swan II, K. Mueller, T. Möller, N. Shareef, R.A. Crawfis, and R. Yagel, "An Anti-Aliasing Technique for Splatting," *Proc. IEEE Visualization '97,* pp. 197-204, Nov. 1997.
[19] J. Torborg and J. Kajiya, "Talisman: Commodity Real-Time 3D Graphics for the PC," *Computer Graphics (SIGGRAPH '96),* pp. 353-364, Aug. 1996.
[20] L. Williams, "Pyramidal Parametrics," *Computer Graphics (SIGGRAPH '83 Proc.),* vol. 17, no. 3, pp. 1-11, July 1983.
[21] S. Winner, M. Kelley, B. Pease, B. Rivard, and A. Yen, "Hardware Accelerated Rendering of Antialiasing Using a Modified A-Buffer Algorithm," *Computer Graphics (SIGGRAPH '97 Proc.),* vol. 31, no. 3A, pp. 307-316, Aug. 1997.
[22] G. Wolberg, *Digital Image Warping.* Los Alamitos, Calif.: IEEE CS Press, 1990.
[23] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "EWA Volume Rendering," *Proc. IEEE Visualization '01,* pp. 29-36, Oct. 2001.

Baoquan Chen received the BS degree in electronic engineering from Xidian University, Xi'an (1991), the MS degree in electronic engineering from Tsinghua University, Beijing (1994), and a second MS degree (1997) and then PhD degree (1999) in computer science from the State University of New York at Stony Brook. He is an assistant professor of computer science and engineering at the University of Minnesota at Twin Cities, where he is also a faculty at the Digital Technology Center and Digital Design Consortium. His research interests generally lie in computer graphics and visualization, focusing specifically on volume visualization, real-time rendering/visualization by both hardware and software (including image-based, point-based rendering, and multiresolution techniques), and 3D data acquisition. His research is supported by the US National Science Foundation (NSF), US Army Research, Microsoft Research, and private donation. He has been on conference committees and/or served as a conference session chair of IEEE Visualization (2001, 2002), IEEE Volume Graphics (2001, 2003), and the Symposium on Volume Visualization and Graphics (2002). He won the NSF CAREER award in 2002 and Microsoft Proposal Award 2002. He is a member of the IEEE Computer Society. For more information see http://www.cs.umn.edu/~baoquan.

Frank Dachille received the BS degree in naval architecture and marine engineering from the Webb Institute of Naval Architecture in 1994. He graduated in 2002 with the PhD degree in computer science from the State University of New York at Stony Brook, where he developed architectures and algorithms for volume rendering at the Center for Visual Computing (CVC) headed by Dr. Arie Kaufman. He develops visualization and analytical applications for medical imaging while directing research and development at Viatronix, Inc. He created collaborative virtual reality environments and CAVE applications for two years at Concurrent Technologies Corporation. His current research interests include global illumination, medical volume visualization, volume rendering architectures, physics-based modeling, and virtual reality. For more information, see http://www.cs.sunysb.edu/~dachille.

Arie E. Kaufman received the BS degree (1969) in mathematics and physics from the Hebrew University of Jerusalem, the MS degree (1973) in computer science from the Weizmann Institute of Science, Rehovot, and the PhD degree (1977) in computer science from the Ben-Gurion University, Israel. He is the director of the Center for Visual Computing (CVC), a leading professor and chair of the Computer Science Department, and leading professor of Radiology at Stony Brook University. He was the founding Editor-in-Chief of the *IEEE Transactions on Visualization and Computer Graphics* (*TVCG*), 1995-1998. He has been the cochair for multiple Eurographics/Siggraph Graphics Hardware Workshops and Volume Graphics Workshops, the papers/program cochair for the ACM Volume Visualization Symposium and the IEEE Visualization Conferenecs, and the cofounder and a member of the steering committee of the IEEE Visualization Conference series. He has previously chaired and is currently a director of the IEEE Computer Society Technical Committee on Visualization and Computer Graphics. He is an IEEE fellow and the recipient of a 1995 IEEE Outstanding Contribution Award, the 1996 IEEE Computer Society's Golden Core Member, 1998 ACM Service Award, 1999 IEEE Computer Society's Meritorious Service Award, and 2002 State of New York Entrepreneur Award. He has conducted research and consulted for more than 30 years, specializing in volume visualization; graphics architectures, algorithms, and languages; virtual reality; user interfaces; and multimedia. For more information see http://www.cs.sunysb.edu/~ari.