OCTOBER 1999

WRL Research Report 99/1

Simple and Table Feline: Fast Elliptical Lines for Anisotropic Texture Mapping

Joel McCormack
Keith I. Farkas
Ronald Perry
Norman P. Jouppi

COMPAQ **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Western Research Laboratory (WRL), located in Palo Alto, California, is part of Compaq's Corporate Research group. Our focus is research on information technology that is relevant to the technical strategy of the Corporation and has the potential to open new business opportunities. Research at WRL ranges from Web search engines to tools to optimize binary codes, from hardware and software mechanisms to support scalable shared memory paradigms to graphics VLSI ICs. As part of WRL tradition, we test our ideas by extensive software or hardware prototyping.

We publish the results of our work in a variety of journals, conferences, research reports and technical notes. This document is a research report. Research reports are normally accounts of completed research and may include material from earlier technical notes, conference papers, or magazine articles. We use technical notes for rapid distribution of technical material; usually this represents research in progress.

You can retrieve research reports and technical notes via the World Wide Web at:

http://www.research.digital.com/wrl/home

You can request research reports and technical notes from us by mailing your order to:

Technical Report Distribution
Compaq Western Research Laboratory
250 University Avenue
Palo Alto, CA 94301 U.S.A.

You can also request reports and notes via e-mail. For detailed instructions, put the word "Help" in the subject line of your message, and mail it to:

wrl-techreports@pa.dec.com

Simple and Table Feline: Fast Elliptical Lines for Anisotropic Texture Mapping

Joel McCormack*, Keith I. Farkas*, Ronald Perry†, and Norman P. Jouppi*

Abstract

Texture mapping using trilinearly filtered mip-mapped data is efficient and looks much better than point-sampled or bilinearly filtered data. These properties have made it ubiquitous: trilinear filtering is offered on a $99 Nintendo 64 video game unit and on a multimillion dollar SGI InfiniteReality. But trilinear filtering represents the projection of a pixel filter footprint from screen space into texture space as a square, when in reality the footprint may be long and narrow. Consequently, trilinear filtering severely blurs images on surfaces angled obliquely away from the viewer.

This paper describes a new texture filtering technique called Feline (for Fast Elliptical Lines). Like other recent hardware anisotropic filtering algorithms, Feline uses an underlying space-invariant (isotropic) filter with mip-mapped data, and so can be built on top of an existing trilinear filtering engine. To texture a pixel, it uses this space-invariant filter at several points along a line in texture space, and combines the results. With a modest increase in implementation complexity over earlier techniques, Feline more accurately matches the desired projection of the pixel filter in texture space, resulting in images with fewer aliasing artifacts. Feline's visual quality compares well against Elliptical Weighted Average, the best efficient software anisotropic texture filtering algorithm known to date, but Feline requires much less setup computation and far fewer cycles for texel fetches. Finally, since it uses standard mip-maps, Feline requires minimal extensions to standard 3D interfaces like OpenGL.

* Compaq Computer Corporation, Western Research Laboratory, 250 University Avenue, Palo Alto, CA 94301. [Joel.McCormack, Keith.Farkas, Norm.Jouppi] @compaq.com.

† Formerly of Digital Equipment Corporation (acquired by Compaq), now at Mitsubishi Electric Research Laboratories, Inc., Cambridge Research Center, 201 Broadway, Cambridge, MA 02139. perry@merl.com.

This report is a superset of *Feline: Fast Elliptical Lines for Anisotropic Texture Mapping*, published in *Proceedings of SIGGRAPH 99*.

© 1999 Association for Computing Machinery.
© 1999 Compaq Computer Corporation

1. Introduction

Ideally, computing a textured value for a pixel involves perspective projecting a filter from screen space (indexed by x and y coordinates) into texture space (indexed by u and v coordinates) to obtain a *warped prefilter*. Since the texture data are discrete samples, we also require a *reconstruction filter* to interpolate between texel samples. For mathematically tractable warped and reconstruction filters, we can combine the two to create a *unified filter* in texture space. Each texel inside the unified filter's footprint is weighted according to the unified filter's corresponding value in screen space, the weighted samples are accumulated, and the sum is divided by the filter's volume in texture space.

Figure 1, inspired by Lansdale [8], gives an intuitive view of this process. A pixel filter is a "window" onto a portion of the texture map; the window's opacity at each point corresponds to the filter's weight. The grid represents a texture map; the shaded rectangle the screen. We view an elliptical portion of the texture map through a round pixel filter. (In degenerate cases, a circle projects to an arbitrary conic section, but for our purposes an ellipse suffices.) Since the pixel "window" can display a single blob of color, the fundamental problem of texture mapping is to compute a representative color at each pixel.

Figure 2 shows a typical pixel filter in screen space—a Gaussian with weighting $e^{-\alpha (x^2 + y^2)}$, truncated to zero beyond a radius of one pixel, and with an α of 2. Tick marks on the x and y axes are at one pixel intervals; the x-y grid is at $1/10$ pixel intervals. Figure 3 shows an exemplary perspective projection of this filter into texture space, where the tick marks on the u and v axes are spaced at one texel intervals, and the grid is at ½ texel intervals. Note the distorted filter profile: each contour line is an ellipse, but the

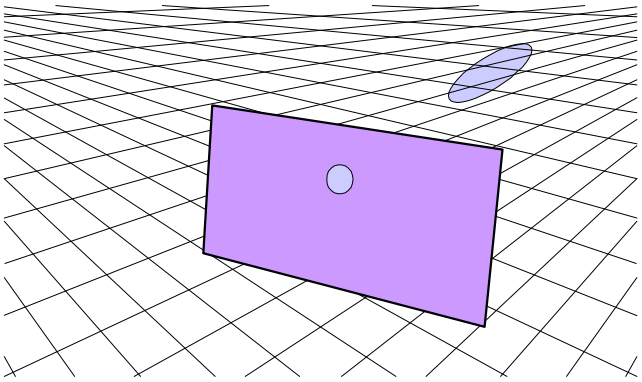

Figure 1: Viewing an elliptical texture area through a circular pixel window.

ellipses representing lower sample weights are increasingly offset from the filter center.

(In this and all other graphs of texture space filters, we normalize the filter volume to one, and highly exaggerate the vertical axis by a constant scale factor. This allows direct comparisons between graphs.)

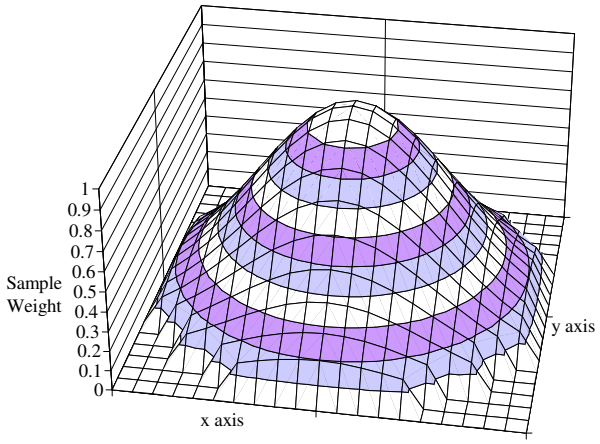

Figure 2: A circular Gaussian filter in screen space.

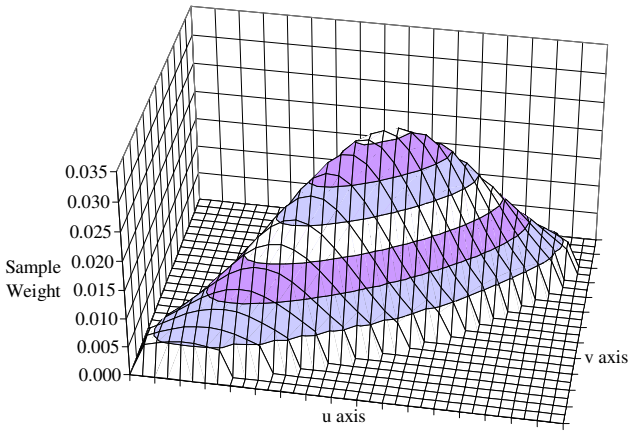

Figure 3: A perspective projection of a Gaussian filter into texture space.

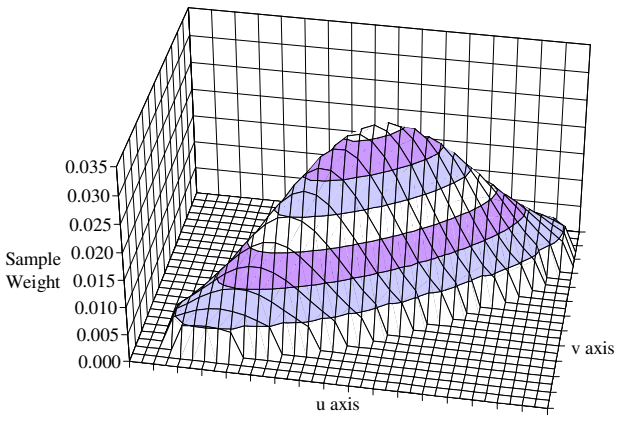

Figure 4: An affine projection of a Gaussian filter into texture space.

Mapping the texel positions in Figure 3 back into pixel positions in Figure 2 (let alone creating a unified filter), so that relative weights can then be applied to the texel values, is a gruesome affair. Rather than using a perspective projection, Heckbert and Greene [4][6] suggest using a locally parallel (affine) projection, as shown in Figure 4. This drastically simplifies computing the footprint and weights of the projected filter. This simplification is visually insignificant. The modest weight differences between Figure 3 and Figure 4 would be extremely hard to detect in any image. Further, to get the slight distortion shown in Figure 3 requires a nearly edge-on view of the surface being texture mapped, in which all detail is lost anyway.

Our algorithm approximates the elliptical filter shown in Figure 4 by performing several isotropic (e.g. trilinear or mip-mapped Gaussian) filtering operations, called *probes*, along the major axis of the ellipse. In comparison to other hardware anisotropic filtering methods, Feline better approximates the elliptical filter by more accurately determining the length of the line along which probes should be placed, spacing probes at better intervals, widening probes under certain conditions, and Gaussian weighting the probe results.

"Simple Feline" uses approximations for the ellipse's major and minor axes that, under ordinary perspective distortions, yield visual results that are as good as using the exact values. More extreme perspectives may occur when environment mapping or otherwise projecting images onto surfaces in a 3D scene (e.g. rendering light from a stained glass windows on a floor). Under such conditions, Simple Feline's approximations of the ellipse may deviate substantially from the true values, and result in noticeable blurring. We thus also describe a more sophisticated algorithm, "Table Feline," which better approximates the ellipse's major and minor axes. Both versions of Feline require just a few additional computations over previous algorithms.

In this paper, we first discuss previous work, including the best efficient software technique, and shortcomings of recent hardware anisotropic filtering techniques. We next describe the desired computations for using several probes along a line, show how to make these computations amenable to hardware, and discuss techniques to reduce the number of probes per pixel. Finally, we present several pictures comparing the various methods of filtering.

2. Previous Work

We first describe Elliptical Weighted Average (EWA), the most efficient direct convolution method known for computing a textured pixel. This provides a quality benchmark against which to compare other techniques. (We do not describe other software efforts like [2] and [3], as we feel that EWA either supersedes these algorithms, or that they are so slow as to be in a different class.) We discuss trilinear filtering, which is popular but blurry. We delve more deeply into Texram, a chip that performs anisotropic filtering by repeated applications of an isotropic

filter along a line, and discuss its weaknesses. We briefly mention other algorithms apparently similar to Texram, but which are not described in sufficient detail to analyze.

2.1. Elliptical Weighted Average

Paul Heckbert's and Ned Greene's Elliptical Weighted Average (EWA) algorithm [4][6] exactly computes the size, shape, and orientation of an elliptical filter like the one shown in Figure 4. If the center of the filter in texture space is translated to (0, 0), then the filter in texture space can be characterized as:

$$d^2(u, v) = Au^2 + Buv + Cv^2$$

The value d^2 represents the distance squared from the center of the pixel when the texel position is mapped back into screen space. Thus, d^2 can index a table of weights that is unrelated to the affine projection, but depends only upon the pixel filter.

EWA determines d^2 for each texel in or near the elliptical footprint. Texels inside the footprint ($d^2 \le 1$) are sampled, weighted, and accumulated. The result is divided by the sum of the weights, which is the elliptical filter's volume in texture space.

Given the partial derivatives $\partial u/\partial x$, $\partial v/\partial x$, $\partial u/\partial y$, and $\partial v/\partial y$, which represent the rates of change of u and v in texture space relative to changes in x and y in screen space, the biquadratic coefficients for computing d^2 are:

$A_{nn} = (\partial v/\partial x)^2 + (\partial v/\partial y)^{2;}$
$B_{nn} = -2 * (\partial u/\partial x * \partial v/\partial x + \partial u/\partial y * \partial v/\partial y);$
$C_{nn} = (\partial u/\partial x)^2 + (\partial u/\partial y)^2;$
$F = A_{nn} * C_{nn} - B_{nn}^2/4;$
$A = A_{nn}/F;$
$B = B_{nn}/F;$
$C = C_{nn}/F;$

Pixels that map to a large area in texture space can be handled by using mip-maps [11], where each level of a texture's mip-map is ½ the height and width of the previous level. Heckbert [6] suggests sampling from a single mip-map level in which the minor radius is between 1.5 and 3 texels. He later implemented unpublished code in which the minor radius is between 2 and 4 texels, in order to avoid subtle artifacts.

Even using mip-maps, highly eccentric ellipses may encompass an unacceptably large area. This area can be limited by computing the ratio of the major radius to the minor radius, and if this ratio is too large, widening the minor axis of the ellipse and computing the corresponding coefficients A, B, and C. The combination of mip-maps and ellipse widening allows EWA to compute a textured pixel with a (large) constant time bound.

Choosing a mip-map level and testing for very eccentric ellipses requires computing the major and minor radii of the ellipse:

$root = \text{sqrt}((A - C)^2 + B^2);$
$A' = (A + C - root)/2;$
$C' = (A + C + root)/2;$

$majorRadius = \text{sqrt}(1/A');$
$minorRadius = \text{sqrt}(1/C');$

Widening a highly eccentric ellipse requires seven multiplies, a square root, an inverse root, and a divide. These setup computations, plus logic to visit only texels in or near the ellipse and compute d^2, have thus far precluded hardware implementation of EWA.

The only complaint that can be leveled against EWA's visual quality is its choice of a Gaussian filter. Other filters produce sharper images without introducing more aliasing artifacts (see Wolberg [12] for an excellent discussion). 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. And as Lansdale [8] points out, none of these filters are as mathematically tractable as the Gaussian for unifying the reconstruction filter and projected pixel filter (i.e., the warped prefilter).

2.2. Trilinear Filtering

Trilinear filtering emphasizes simplicity and efficiency at the cost of visual quality. Rather than computing the shape of the projected filter footprint, it uses a square filter in texture space. By blending two 2 x 2 bilinear filters from adjacent mip-map levels, trilinear filtering crudely approximates a circular filter of an arbitrary size. Figure 5 shows a trilinear filter that (poorly) approximates the EWA filter shown in Figure 4. The axis tick marks are spaced one texel apart, while the grid is spaced at ½ texel intervals. Strictly speaking, because it blends two 2 x 2 bilinear filtering operations, a trilinear filter samples a square area of 2^n x 2^n texels. However, most of the filter volume resides inside a circle with the nominal filter radius. In the 2D pictures below, we thus show a trilinear filter's footprint as a circle of the nominal radius.

A trilinear filter blurs or aliases textures applied to surfaces that are angled obliquely away from the viewer. These artifacts arise because the fixed shape of the trilinear filter poorly matches the desired elliptical filter footprint, and so the trilinear filter samples data outside the ellipse, doesn't sample data inside the ellipse, or both.

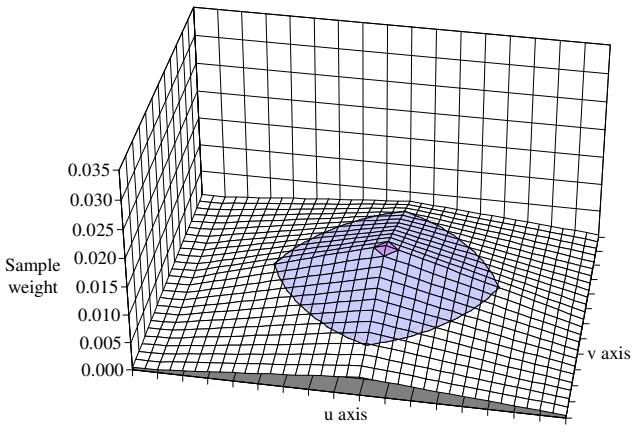

Figure 5: A trilinear filter approximation to Figure 4.

2.3. Texram

Texram [10] provides higher visual quality than trilinear filtering with less complexity than EWA. Texram uses a series of trilinear filter probes along a line that approximates the length and slope of the major axis of EWA's elliptical footprint.

The Texram authors considered computation of the ellipse parameters too costly for hardware, and so substituted simplified approximations. These approximations underestimate the length of the major axis of the ellipse, causing aliasing; overestimate the length of the minor axis, causing blurring; and deviate from the slope of the major axis, causing yet more blurring and aliasing. Nonetheless, these errors are visually insignificant under typical perspective projections, as discussed further in Section 3.2 below.

Texram has other problems that do manifest themselves as visible aliasing artifacts. It usually samples along a line that is much shorter than the ellipse, and can space the trilinear probes too far apart. Texram always uses 2^n equally weighted probes, which causes poor high-frequency rejection along the major axis. These problems make Texram's visual quality noticeably inferior to EWA.

Texram uses the four partial derivatives to create two vectors in texture space: $(\partial u/\partial x, \partial v/\partial x)$ and $(\partial u/\partial y, \partial v/\partial y)$. The authors claim to sample roughly the area inside the parallelogram formed by these two vectors, by probing along a line that has the length and slope of the longer of the two vectors. This line can deviate from the slope of the major axis of EWA's elliptical filter by as much as 45°. This is not as bad as it sounds. The largest angular errors are associated with nearly circular filters, which are relatively insensitive to errors in orientation.

Texram's sampling line can be shorter than the true ellipse's major axis by nearly a factor of four. One factor of two comes from Texram's use of the length of the longer vector as the length of the sample line. Note that if orthogonal vectors are plugged into the ellipse equations in Section 2.1 above, the major radius is the length of the longer vector, and so the ellipse's major diameter is actually *twice* the length of this vector. Texram's error is apparently due to an older paper by Paul Heckbert [5], in which he suggested using a filter diameter that is really a filter radius.

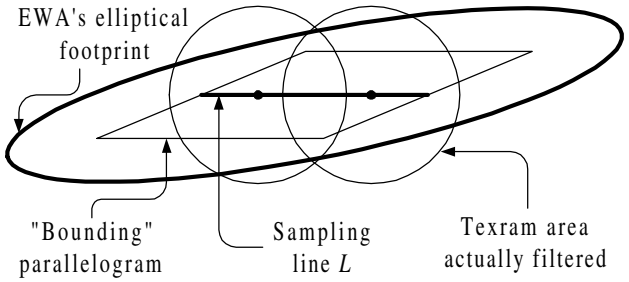

Figure 6: Texram area sampled vs. EWA.

Another factor of two comes from non-orthogonal vectors. If the two vectors are nearly parallel and equal in length, the elliptical footprint is very narrow and has a major radius nearly twice the length of either vector. Again, this is not as bad as it sounds: under typical perspective distortions the longer vector is at least 93% the length of the true ellipse radius.

Texram approximates the radius of the minor axis of the ellipse by choosing the shortest of the two parallelogram side vectors and the two parallelogram diagonals $(\partial u/\partial x + \partial u/\partial y, \partial v/\partial x + \partial v/\partial y)$ and $(\partial u/\partial x - \partial u/\partial y, \partial v/\partial x - \partial v/\partial y)$. If the side vectors are nearly parallel and the shorter is half the length of the longer, this approximation can be too wide by an arbitrarily large factor.

One of the Texram authors was unsure which values round up or down in the division that computes the number of probes. We have assumed values in the half-open interval [1.0 to 1.5) round to one probe, values in [1.5 to 3) round to two probes, values in [3 to 6) round to four probes, etc. Texram does not adjust the probe diameter when it rounds down (as discussed in Section 3.1 below), and so can space probes too far apart. In this case, Texram's composite filter looks like a mountain range with individual peaks. These peaks cause aliasing, and can beat against repeated texture patterns to create phantom patterns.

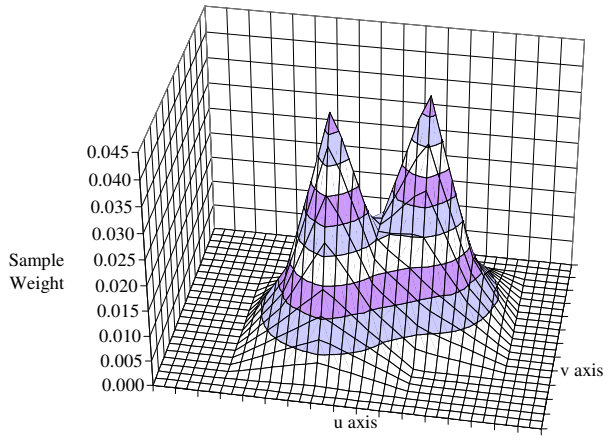

Figure 7: Worst-case 2-probe Texram filter.

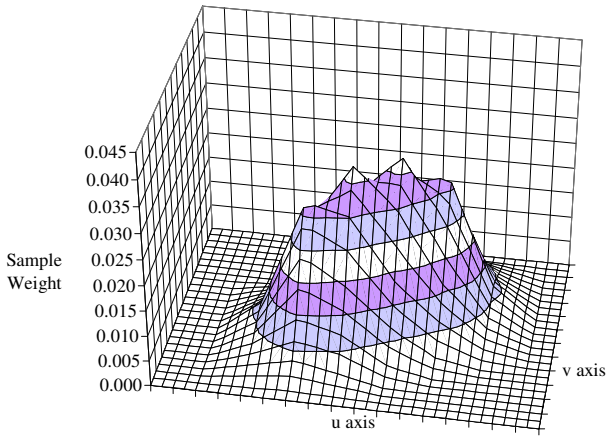

Figure 8: Best-case 4-probe Texram filter.

Figure 6 shows an extreme example of these errors, in which ($\partial u/\partial x$, $\partial v/\partial x$) is (13, 0) and ($\partial u/\partial y$, $\partial v/\partial y$) is (12, 5). The area sampled by EWA is shown as the large heavily outlined ellipse, while Texram's trilinear filter footprints are shown as circles.

To approximate the elliptical filter shown in Figure 4, Texram computes that it would ideally use 2.97 probes. Figure 7 shows the resulting filter if Texram rounds down to 2 probes. Since it does not widen the trilinear probes, the two mountain peaks are quite distinct, and texels near the center of the filter are severely underweighted. We forced an experimental version of Texram to always round up the number of probes to a power of two; its images exhibited almost as much aliasing as the original version that rounds up or down. Figure 8 shows the resulting filter if Texram rounds the 2.97 probes up to 4 probes, and helps explain why aliasing remains. The probes don't extend far enough along the major axis of the ellipse, and the probes are equally weighted. This creates a mesa-like filter that would look at home in Monument Valley, rather than the smoothly sloped Hawaiian shield volcano filter of EWA.

2.4. Other Hardware Algorithms

Microsoft's Talisman [1] uses a filtering algorithm "in the spirit of" Texram. Details are scant, but aliasing evident in the examples suggest that they may be haunted by Texram's problems. Evans & Sutherland holds U.S Patent #5,651,104 for using space-invariant probes along a line. The patent doesn't describe how to compute the probe line, but the diagrams imply a line that is at most a single pixel in length in screen space, which is once again so short that it will produce visible aliasing artifacts.

3. The Feline Algorithm

Like Texram, Feline uses several isotropic probes along a line L to implement an anisotropic filter. However, we compute a more appropriate length for the sampling line L, allow the number of probes to be any integer, don't space probes too far apart, and weight the probes using a Gaussian curve. In some circumstances we use a mip-mapped Gaussian filter for the probes. Feline requires little additional logic over Texram, yet achieves visibly superior results.

We first describe the desired computations to yield the locations and weights for a series of probe points along a line. We next describe two versions of our algorithm, which differ only in their approximations of the major and minor axes of the ellipse. "Simple Feline" inherits Texram's approximations of the major and minor radii, after which it implements the desired computations in a fashion suitable for hardware. Under highly distorted perspective projections, which may occur when environment mapping, Simple Feline's major and minor radii approximations result in blurring. "Table Feline" uses a two-dimensional table to compute the ellipse axes more accurately. We conclude with techniques to reduce the number of probes.

3.1. The Desired Computations

The combination of multiple isotropic probes should closely match the shape of the EWA filter. Thus, the probe points should occur along the major axis of the ellipse, the probes should be Gaussian weighted, and the probe filter width should be equal to the minor axis of the ellipse.

(Ideally, the probe filter width would be related to the width of the ellipse at each probe position. We initially did not investigate this because we didn't know how to optimize the trade-off between the probe diameter, probe weighting, probe spacing, and the number of probes. After implementing constant diameter probes, we saw no reason to pursue variable diameter probes. The "improvement" was unlikely to be visible, but would significantly increase the number of probes due to tighter spacing of small probes near the ends of the ellipse.)

We compute *majorRadius* and *minorRadius* as in Section 2.1 above, and then the angle *theta* of the major axis:

theta = arctan(B/(A-C))/2;
// If *theta* is angle of minor axis, make it
// angle of major axis
if ($A > C$) *theta* = *theta* + π/2;

If *minorRadius* is less than one pixel (that is, we are magnifying along the minor axis, and possibly along the major axis), the appropriate radii should be widened—there is no point in making several probes to nearly identical locations. Heckbert's Master's Thesis [6] elegantly addresses this situation. He unifies the reconstruction and warped prefilter by using the following computations for A and C rather than the ones shown in Section 2.1 above:

$A_{nn} = (\partial v/\partial x)^2 + (\partial v/\partial y)^2 + 1;$
$C_{nn} = (\partial u/\partial x)^2 + (\partial u/\partial y)^2 + 1;$

This makes the filter radius sqrt(2) texels for a one-to-one mapping of texels into pixels. (The filter radius approaches one texel as magnification increases.) While theoretically superior, this wider filter blurs more than the radius one trilinear filter conventionally used for unity mappings and for magnifications. In order to match this convention, and to make hardware implementation feasible, we instead clamp the radii to a minimum of one texel:

minorRadius = max(*minorRadius*, 1);
majorRadius = max(*majorRadius*, 1);

The space-invariant probes along the major axis have a nominal radius equal to *minorRadius*, and so the distance between probes should also be *minorRadius*. The end probes should be set in from the ellipse by a distance of *minorRadius* as well, so that they don't sample data off the ends of the ellipse. Therefore, the number of probes we'd like (*fProbes*), and its integer counterpart (*iProbes*), are derived from the ratio of the lengths of the major and minor radii of the ellipse:

fProbes = 2*(*majorRadius*/*minorRadius*) – 1;
iProbes = floor(*fProbes* + 0.5);

To guarantee that texturing a pixel occurs in a bounded time, we clamp *iProbes* to a programmable value *maxProbes*. An application can use a small degree of anisotropy at high frame rates, and then allow more eccentric filters for higher visual quality when motion ceases.

iProbes = min(*iProbes*, *maxProbes*);

When *iProbes* > *fProbes*, because *fProbes* is rounded up, we space probes closer than their radius, rather than blur the image by sampling data off the ends of the ellipse.

When *iProbes* < *fProbes*, either because *fProbes* is rounded down, or because *iProbes* is clamped, the ellipse will be probed at fewer points than desired. Spacing the probes farther apart than their radius, or shortening the line *L*, may cause aliasing artifacts. Instead, we blur the image by increasing *minorRadius* to widen the ellipse, effectively reducing its eccentricity to match *iProbes*. Increasing *minorRadius* increases the level of detail and thus the nominal radius of the probe filter.

if (*iProbes* < *fProbes*)
    minorRadius = 2**majorRadius* / (*iProbes*+1);
levelOfDetail = $\log_2$(*minorRadius*);

Analogous to clamping *minorRadius* and *majorRadius* to 1, we also use a single probe in the smallest 1 x 1 mip-map. This reduces cycles spent displaying a repeated texture in the distance. We don't attempt a similar optimization for the 2 x 2 or 4 x 4 mip-maps. Consider the worst 2 x 2 case, in which a checkerboard is mirror repeated, and an ellipse with a *minorRadius* of 1 is centered at a corner of the texture map. Figure 9 depicts this situation, where the thin lines delineate texels, and the thick lines delineate the (repeated) 2 x 2 mip-map. The circle on the left uses one probe to compute an all-white pixel. The ellipse on the right uses 6 probes to compute the darkest possible pixel of 52% white, 48% shaded. (The white texels apparently inside the ends of the ellipse don't contribute to the pixel's color, as only texel centers are sampled.) Since longer ellipses converge so slowly to an intermediate color, we restrict ourselves to the trivial adjustment:

if (*levelOfDetail* > *texture.maxLevelOfDetail*) {
    levelOfDetail = *texture.maxLevelOfDetail*;
    iProbes = 1;
}

We compute the stepping vector (Δu, Δv), which is the distance between each probe point along the line:

lineLength = 2*(*majorRadius* – *minorRadius*);
Δu = cos(*theta*) * *lineLength* / (*iProbes* – 1);
Δv = sin(*theta*) * *lineLength* / (*iProbes* – 1);

(The stepping vector is irrelevant if *iProbes* is 1.) The sample points are distributed symmetrically about the midpoint (u_m, v_m) of the sampling line *L* in the pattern:

(u_n, v_n) = (u_m, v_m) + *n*/2 * (Δu, Δv)

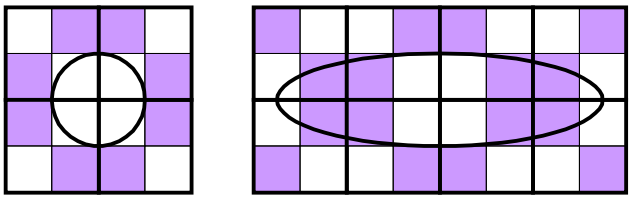

Figure 9: Ellipses sampling a 2 x 2 texture map oscillate around a blend of the two colors as eccentricity increases.

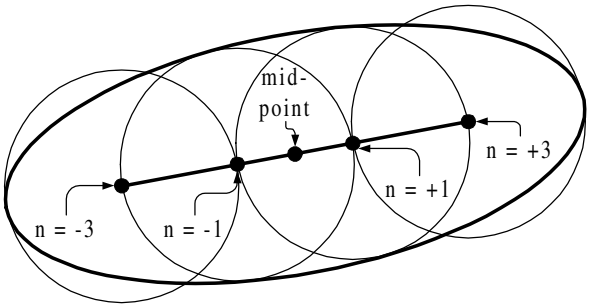

Figure 10: Positioning an even number of probes.

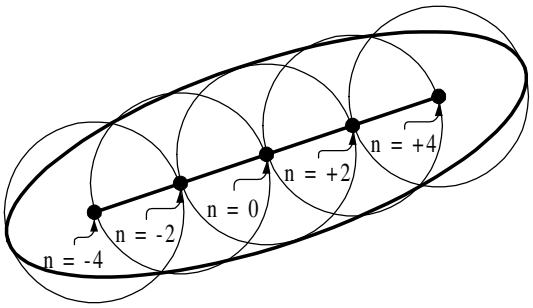

Figure 11: Positioning an odd number of probes.

where *n* = ±1, ±3, ±5, … if *iProbes* is even, as shown in Figure 10, and *n* = 0, ±2, ±4, … if *iProbes* is odd, as shown in Figure 11.

We apply a Gaussian weight to each probe *n* by computing the distance squared of the probe from the center of the pixel filter in screen space, then exponentiating:

d = *n*/2 * sqrt($\Delta u^2 + \Delta v^2$) / *majorRadius*;
$d^2 = n^2/4 * (\Delta u^2 + \Delta v^2) / majorRadius^2$;
$relativeWeight = e^{-\alpha * d^2}$;

Finally, we divide the accumulated probe results by the sum of all the weights applied.

This ideal algorithm uses 6 probes to approximate the filter in Figure 4. The resulting composite filter is shown in Figure 12. It provides a remarkably close match.

3.2. Implementing Simple Feline

Simple Feline implements the above computations, except it uses Texram's approximations to the ellipse axes rather than computing the exact values. We use the longer of the two vectors ($\partial u/\partial x$, $\partial v/\partial x$) and ($\partial u/\partial y$, $\partial v/\partial y$) as the

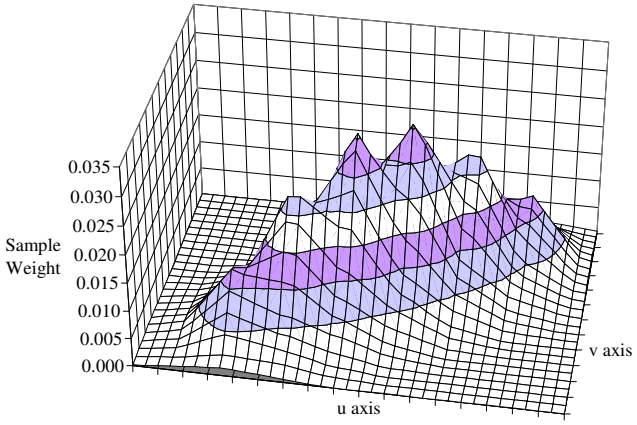

Figure 12: The desired Feline computations use 6 probes for a most excellent approximation to the elliptical filter.

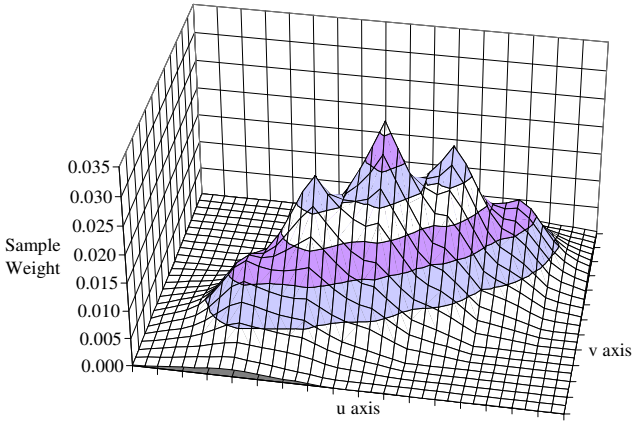

Figure 13: Simple Feline uses 5 probes to nicely approximate the elliptical filter.

major radius, and the shortest of those and the two diagonals ($\partial u/\partial x + \partial u/\partial y$, $\partial v/\partial x + \partial v/\partial y$) and ($\partial u/\partial x - \partial u/\partial y$, $\partial v/\partial x - \partial v/\partial y$) as the minor radius length.

We were surprised that under typical perspective projections, these approximations work essentially as well as the exact values. We discovered that the two vectors ($\partial u/\partial x$, $\partial v/\partial x$) and ($\partial u/\partial y$, $\partial v/\partial y$) are more or less orthogonal under typical perspective distortions. In the images shown below, the angle between the two are in the range 90° ± 30°. The most extreme angles occur with very unequal vector lengths, where the approximations slightly overestimate the minor axis length but accurately estimate the major axis. The simple approximations are tolerably close to the true values under these conditions.

We use a two-part linear approximation for the vector length square root. Without loss of generality, for a vector (a, b) assume that $a, b > 0$ and $a > b$. The following function is within ±1.2% of the true length $\text{sqrt}(a^2 + b^2)$:

if ($b < 3a/8$) return $a + 5b/32$
else return $109a/128 + 35b/64$

We do not compute the stepping vector with trigonometric functions, but instead scale the longer vector directly. Call the longer vector components (*majorU*, *majorV*). Either this vector describes *majorRadius*, or else *iProbes* is one and the stepping vector is irrelevant. By substituting *majorU*/*majorRadius* for cosine, and *majorV*/*majorRadius* for sine, we get:

r = *minorRadius* / *majorRadius*;
i = *oneOverNMinusOneTable*[*iProbes*];
Δu = 2*(*majorU* – *majorU***r*) * *i*;
Δv = 2*(*majorV* – *majorV***r*) * *i*;

Finally, we use a triangularish two-dimensional weight table to avoid computing and exponentiating d^2. We use the smaller of *fProbes* truncated to a couple fractional bits, or *iProbes*, as the weight table's row index, so that each row of weights applies to a small range of ellipses. The column index is floor((abs(*n*)+1)/2). By dividing each of the raw weights in a row by the sum of the weights for that row, the weights in each row sum to 1. Consequently, we need not normalize the final accumulated result. Note that if *iProbes* is odd, the W_0 entry in a row should count half as much as the other entries when computing the sum: it is used once, while the other weights are used twice.

Most of the computations specific to Feline can use group scaled numbers with a precision of 8 bits. That is, the partial derivative with the largest magnitude determines a group exponent for all four derivatives, which are then shifted to yield four 8-bit values. Only the derivative with the largest magnitude is guaranteed to be normalized; any or all of the smaller derivatives may be denormalized. The small errors introduced by limiting precision causes sampling along a line at a slightly different angle, and at intervals that are slightly smaller or larger than desired. These errors are negligible compared to the inaccuracies caused by Simple Feline's gross approximations to the ellipse axes, or by Table Feline's small table size.

Simple Feline computes 4.97 probes, which it rounds to 5 probes, to approximate the filter show in Figure 4. Figure 13 shows the resulting filter. Simple Feline averages fewer probes than the desired computations. But under ordinary perspectives, we couldn't see any visible difference between images created with the desired Feline computations and the Simple Feline computations.

3.3. Implementing Table Feline

Extreme perspective distortions may occur when projecting images onto surfaces in a 3D scene. This includes environment mapping (projecting fake reflections of the 3D scene onto shiny surfaces), projecting a film image onto a screen, or painting light transmitted by a stained glass window onto the floor and walls of a room. Under high perspective distortions, mismatches between the approximate and true ellipse parameters cause Simple Feline to blur images excessively. We obtained more accurate approximations of the ellipse parameters by using a two-dimensional 16 x 16 entry table. The table encodes the EWA setup computations of Section 2.1 and the trigono-

metric computations of Section 3.1. The table yields two scaling factors we apply to the longer vector length to yield *minorRadius* and *majorRadius*, and two values we use to scale and rotate the longer vector components to yield (*majorU*, *majorV*).

We first map the two vectors into a canonical representation: the longer vector becomes (1, 0), and the shorter vector is represented as a length between 0 and 1, and a counter-clockwise angle from the longer vector between 0° and 90°. To this end, we compute the relative length of the shorter vector to the longer, and a representation of the angle between them. These two values index the table.

Representing the angle between the two vectors requires some care. We initially used the sine, but were unhappy about arcsine's sensitivity when the sine is near 1: the quantization error involved in indexing a 16-entry table can result in a major axis rotated 15° from the true value. This problem was compounded by the inexact square root length approximation used as the sine's denominator. We improved accuracy by using sine for angles between 0° and 45°, and cosine for angles between 45° and 90°. But we got even better results using the tangent and cotangent, whose inverse functions are less sensitive than sine and cosine, and whose computation avoided the square root approximation.

The first half of the table is indexed by the tangent, which is the cross product divided by the dot product, for angles between 0° and 45°. The second half of the table is indexed by the cotangent, which is the dot product divided by the cross product, for angles between 45° and 90°.

Shorter vectors that are between 270° and 360° (a.k.a. –90° to 0°) from the larger vector are handled through post-processing of the table data. Shorter vectors between 90° and 270° are implicitly rotated 180°, and thus lie between a –90° and 90° degrees. 180° rotation of a vector is equivalent to negating both of its coordinates. Examining the derivation of the desired ellipse parameters in Section 2.1 above, we see that negating either of the two vectors does not change the ellipse parameters.

The following code shows how to convert the $(\partial u/\partial x, \partial v/\partial x)$ and $(\partial u/\partial y, \partial v/\partial y)$ vectors to table indices:

```
xLen = SqrtLengthApprox(∂u/∂x, ∂v/∂x);
yLen = SqrtLengthApprox(∂u/∂y, ∂v/∂y);
if (xLen > yLen) {
    longU = ∂u/∂x;
    longV = ∂v/∂x;
    longLen = xLen;
    ratio = yLen / xLen;
    // if cross product positive, short vector
    // is counterclockwise from long vector
    ccw = True;
} else {
    longU = ∂u/∂y;
    longV = ∂v/∂y;
    longLen = yLen;
    ratio = xLen / yLen;
    // if cross product positive, short vector
    // is clockwise from long vector
    ccw = False;
}
// (∂u/∂x, ∂v/∂x) ✖ (∂u/∂y, ∂v/∂y)
cross = ∂u/∂x * ∂v/∂y – ∂v/∂x * ∂u/∂y;
// (∂u/∂x, ∂v/∂x) • (∂u/∂y, ∂v/∂y)
dot = ∂u/∂x * ∂u/∂y + ∂v/∂x * ∂v/∂y;
ccw = ccw ^ (cross < 0.0) ^ (dot < 0.0);
cross = abs(cross);
dot = abs(dot);
if (cross < dot) { // Compute tangent
    tanCotan = cross / dot;
} else {  // Compute cotangent + 1.0
    tanCotan = dot / cross + 1.0;
}
// Convert into integer indices for a 16 x 16 table.
iRatio = (int) (16 * min(0.999, ratio));
//  tanCotan is between 0 and 2, so uses steps of 1/8.
iTanCotan = (int) (8 * min(1.999, tanCotan));
```

We extract values from the table to compute the vector (*majorU*, *minorU*) describing *majorRadius*, and to compute the unclamped lengths *majorRadius* and *minorRadius*.

```
(uvScale, uvScale90, minorRadiusScale,
    majorRadiusScale) =
    ellipseParamTable[iRatio][iTanCotan];
if (ccw) {
    // Compose major axis from long vector, and
    // long vector rotated 90º counterclockwise.
    majorU = longU*uvScale – longV*uvScale90;
    majorV = longV*uvScale + longU*uvScale90;
} else {
    // Compose major axis from long vector,
    // and long vector rotated 90º clockwise.
    majorU = longU*uvScale + longV*uvScale90;
    majorV = longV*uvScale – longU*uvScale90;
}
// Create major, minor axis radius lengths
minorRadius = longLen * minorRadiusScale;
majorRadius = longLen * majorRadiusScale;
```

From here, Table Feline looks just like Simple Feline.

3.4. Increasing Efficiency

We investigated how far we could reduce the number of probes by shortening and widening the ellipse, and by spreading probe points farther apart than their radius. We can shorten the ellipse using a *lengthFactor* <= 1:

```
majorRadius = max(majorRadius * lengthFactor,
                  minorRadius);
majorU *= lengthFactor;
majorV *= lengthFactor;
```

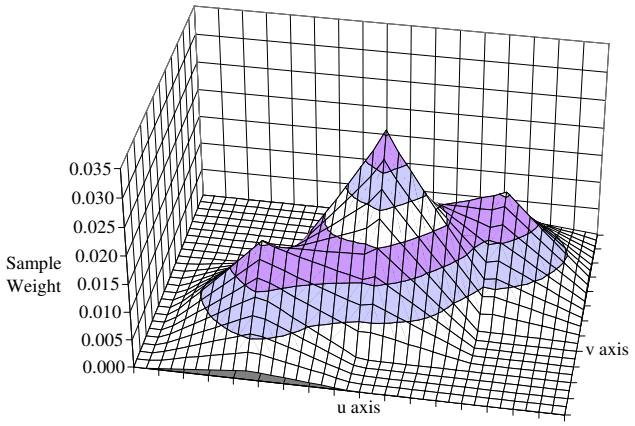

Figure 14: Simple Feline with *blurFactor* 1.31 and *aliasFactor* effectively 1.26

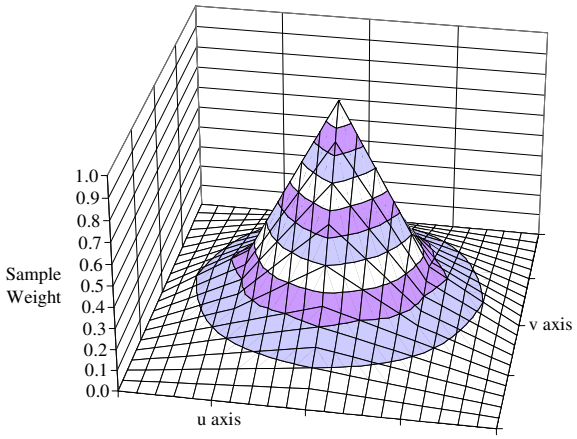

Figure 15: A trilinear filter with a nominal radius of sqrt(2)

The code in Section 3.1 proportionately widens an ellipse more when rounding down a small value of *fProbes* than a large one. For example, if *fProbes* is 1.499, rounding down to 1 scales *minorRadius* up by 25%; if *fProbes* is 4.499, rounding down to 4 scales *minorRadius* up by only 10%. We can instead compute *iProbes* so that for all values of *fProbes*, we widen the ellipse to at most a *blurFactor* times the minor radius. We also allow stretching the distance between probe positions to at most an *aliasFactor* times the probe filter radius:

f = 1 / (*blurFactor* * *aliasFactor*);
iProbes =
ceiling(*f* * 2*(*majorRadius*/*minorRadius*)) – 1;

If *iProbes* is not clamped to *maxProbes*, we can accommodate the reduction of *fProbes* to *iProbes* by some combination of blurring and aliasing within the limits of *blurFactor* and *aliasFactor*. For computational simplicity, we blur (widen the ellipse) by increasing *minorRadius* by up to *blurFactor*:

minorRadius = min(2**majorRadius*/(*iProbes*+1),
minorRadius * *blurFactor*);

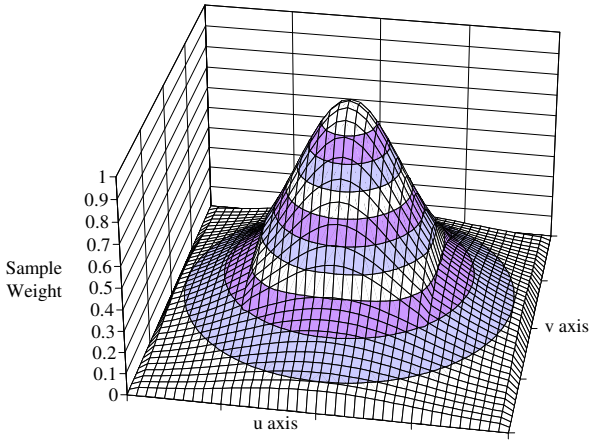

Figure 16: A mip-mapped Gaussian filter with nominal radius of sqrt(2)

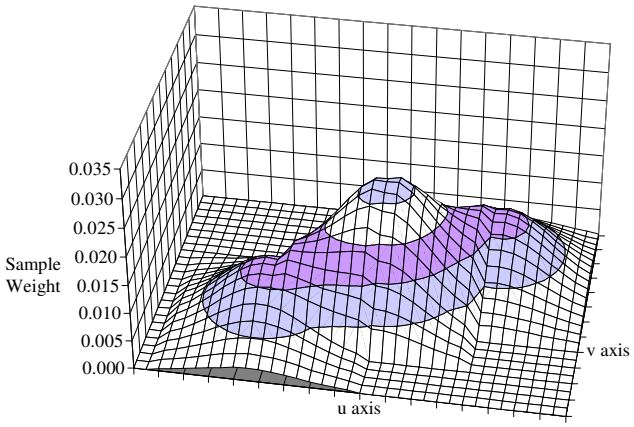

Figure 17: Figure 14 using a Gaussian probe

The computations of Δu and Δv automatically make up any remaining difference between *iProbes* and *fProbes* by increasing probe spacing.

If *iProbes* is clamped, we must exceed either *blurFactor* or *aliasFactor*. In this case, we blur (in excess of *blurFactor*) to the point where the computations of Δu and Δv will increase probe spacing by exactly *aliasFactor*:

minorRadius = 2 * *majorRadius* /
((*iProbes*+1) * *aliasFactor*);

We chose two sets of parameter values empirically. The "high-quality" set (*lengthFactor* .9625, *blurFactor* 1.15625, *aliasFactor* 1.1532) reduces the number of probes by 24% with almost no degradation of image quality, compared to the constant rounding of Section 3.1. The "high-efficiency" set (*lengthFactor* 0.9625, *blurFactor* 1.3125, *aliasFactor* 1.3544) uses the same number of probes as Texram to provide images superior to Texram, though with more artifacts than the high-quality set.

The high-efficiency *aliasFactor* allows probes to be spaced quite far apart, which introduces aliasing artifacts. Figure 14 shows how Simple Feline approximates the filter of Figure 4 with the high-efficiency parameters. Figure 15

shows a detailed picture of a single trilinear probe. Note how especially at high weights, a trilinear filter is not circularly symmetric: the isocontour lines extend substantially farther along the *u* and *v* axes than along the diagonals, and the peak in the center is sharp.

We obtained slightly better images by changing the probe filter on each of the two adjacent mip-map levels from a bilinear filter to a Gaussian filter truncated to a 2 x 2 square. We then linearly combine the two Gaussian results using the fractional bits of the level of detail. Figure 16 shows a mip-mapped Gaussian with a nominal radius of sqrt(2) texels. The circular symmetry and lack of a sharp central peak result in the smoother composite filter shown in Figure 17, and in images with fewer aliasing artifacts. The Gaussian also makes single-probe magnifications look better. However, note that the Gaussian also slightly reduces the sharpness of images.

The Gaussian is the epitome of a separable filter, and so a hardware trilinear filter tree is easily adapted to implement Gaussian weightings [9]. A trilinear filter tree uses the fractional bits of *u* and *v* directly as filter weights. The Gaussian requires four copies of a small one-dimensional table to map the fractional bits of *u* and *v* before using them as weights.

4. Comparisons with Previous Work

Figure 18 through Figure 22 show various algorithms generating a pattern of curved lines. Figure 23 through Figure 26 show a floor of bricks, and Figure 27 through Figure 30 show magnified texture-mapped text. These images *should not be viewed with Adobe Acrobat*, which uses a reduction filter that introduces artifacts. Please print them on a high-quality ink-jet printer.

Texram images use the original algorithm in [10]; correcting the errors described in Section 2.3 above results in many more probes *and* degrades visual quality! Aliasing artifacts mostly remain, and images significantly blur due to the equal weighting of probes along a long line.

Simple Feline images use parameters as described in Section 3.4 above, and a mip-mapped Gaussian for the probe filter.

Mip-mapped EWA samples from a mip-map level where the minor radius is between 1.5 and 3 texels; this

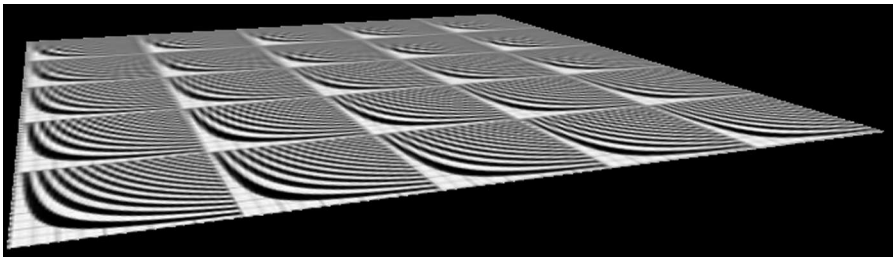

Figure 18: Trilinear paints curved lines with blurring.

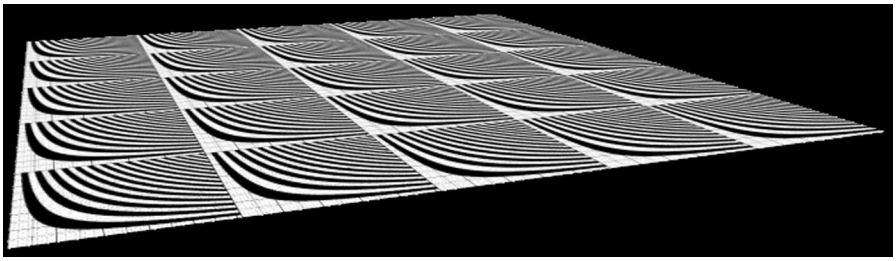

Figure 19: Texram paints curved lines with strong Moiré artifacts.

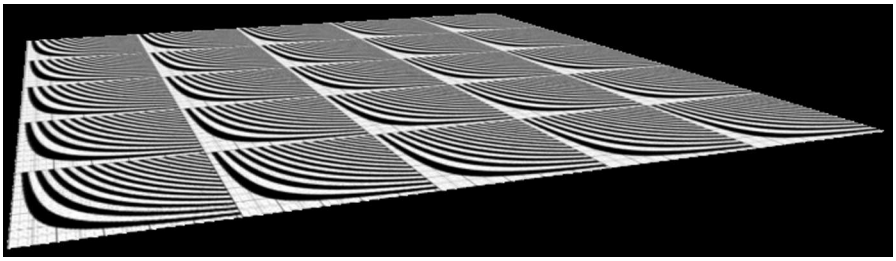

Figure 20: High-efficiency Simple Feline paints curved lines with fewer artifacts.

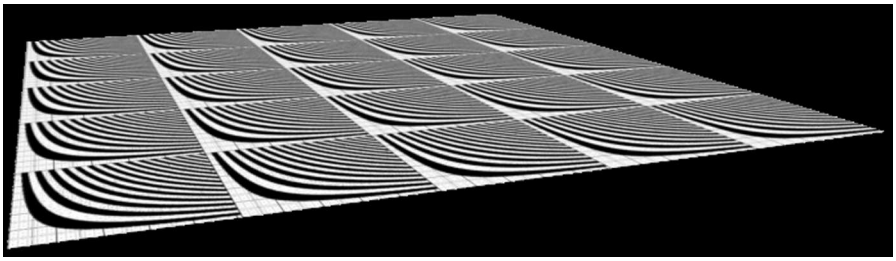

Figure 21: High-quality Simple Feline paints curved lines with few artifacts.

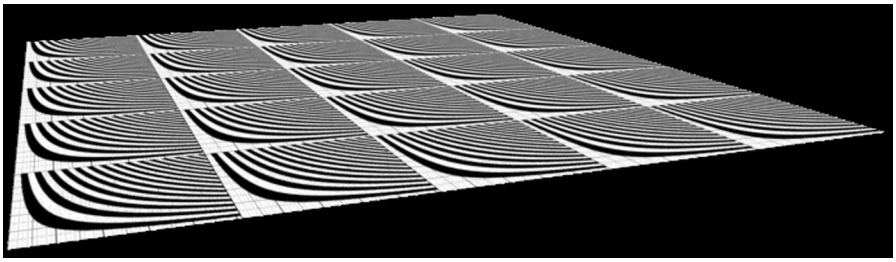

Figure 22: Mip-mapped EWA paints curved lines with few artifacts.

looks identical to a radius between 2 and 4, but samples about half as many texels.

Trilinear, Texram, and Feline images use a radius 3 Lanczos filter to create mip-maps. EWA images use a box filter: the Lanczos filter causes "bluriness banding" artifacts when EWA jumps from using a large ellipse in one mip-map to using a small ellipse in the next higher mip-map.

Simple Feline with high-quality parameters generates images comparable to EWA, but with slightly stronger Moiré patterns. The only exception occurs if a box filter is used to create mip-maps for textures like checkerboards. Because the base texture and all its mip-maps then contain illegally high frequencies that Feline's relatively narrow filter cannot remove, Feline displays much stronger Moiré artifacts than EWA. Using a better filter, such as the Lanczos, to create the mip-maps makes Feline display *fewer* artifacts than EWA—Feline is more likely to use filtered mip-mapped data, rather than the unfiltered base texture. Unfortunately, images showing such artifacts are severely misrendered by all printers we've tested.

Both sets of Feline images are much sharper, and exhibit far fewer Moiré artifacts, than those generated by trilinear filtering. Though not shown here, we note that Texram, high-efficiency Feline, and even to some degree high-quality Feline are subject to "probe banding" on repeated textures. Some images show a visible line where the number of probes increases from one value to another. (Setting all parameters to 1.0 removes the banding from Feline images, but requires many more probes.)

Texram images sometimes seem a little sharper than Feline images, but then, aliased images always seem sharper than antialiased images. Repeated texture patterns amplify Texram's aliasing problems to create strong Moiré patterns, as shown in the curved lines and bricks images. These patterns are even more disturbing in moving images, where they shimmer across the surface. Texram's aliasing is more subtle in non-repeated textures such as text. Comparing the high-efficiency Feline images to Texram is especially interesting: both use the same number of probes,

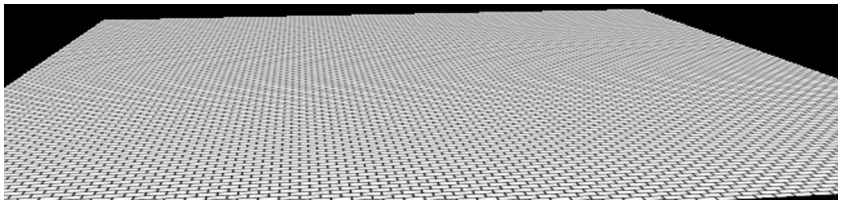

Figure 23: Texram paints bricks with herringbone artifacts.

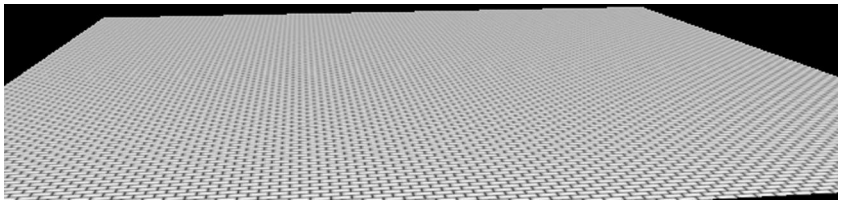

Figure 24: High-efficiency Simple Feline paints bricks with fewer artifacts.

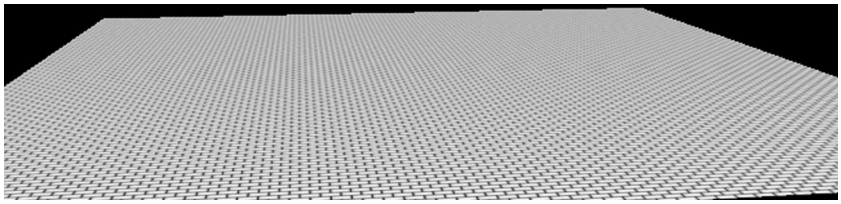

Figure 25: High-quality Simple Feline paints bricks with few artifacts.

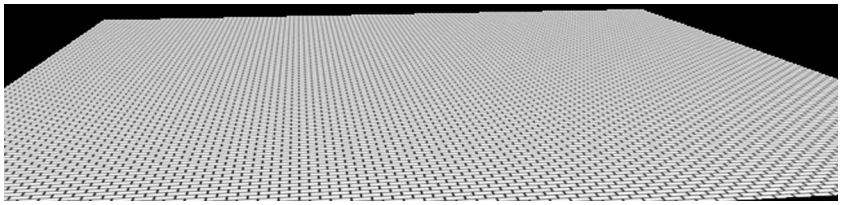

Figure 26: Mip-mapped EWA paints bricks with fewest artifacts.

Figure 27: Trilinear paints blurry text.

Figure 28: Texram paints text with stairstepping.

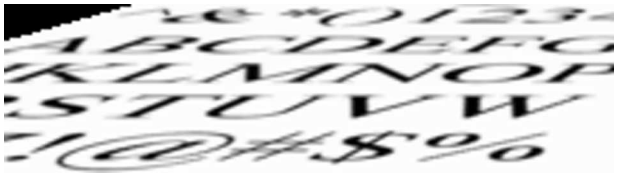

Figure 29: High-efficiency Simple Feline paints smooth text.

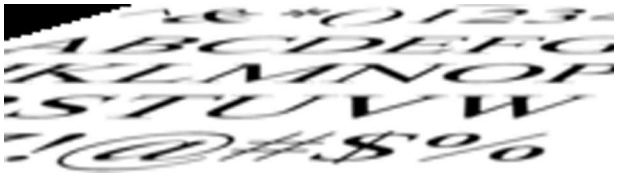

Figure 30: Mip-mapped EWA paints smooth text.

Figure 31: Simple Feline blurs highly distorted perspective projections.

Figure 32: Table Feline is much sharper for highly distorted perspective projections.

Figure 33: Simple Feline uses a few probes that are too wide (light blue is 2 probes, dark blue is 4 probes).

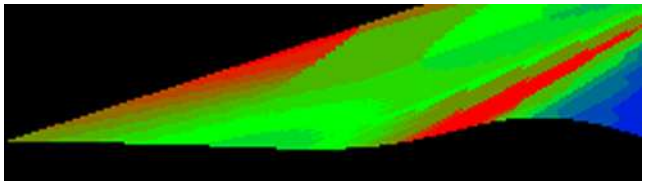

Figure 34: Table Feline uses many more probes (pure green is 8 probes, pure red is 16 probes).

but the Feline images exhibit far fewer artifacts. Experiments show that Feline's quality is due to the use of a Gaussian probe filter, the Gaussian weighting of probe results, the widening of the ellipse, and the end-to-end coverage of the ellipse.

Under modest perspective distortions that feel "normal," Simple Feline and Table Feline have equivalent visual quality, as the Simple Feline approximations result in an *fProbes* that is only slightly smaller than desired. However, under extreme distortions, Simple Feline underestimates the major axis and overestimates the minor axis. This results in blurring, as shown in Figure 31. Table Feline's approximations are much more accurate, resulting in a sharper image, as shown in Figure 32. For such scenes, though, Table Feline requires many more probes. Figure 33 shows that Simple Feline uses two to four probes per pixel to paint Figure 31, while Figure 34 shows that Table Feline uses as many as 16 probes to paint Figure 32.

Higher visual quality comes at increased computational cost for setup and sampling. Fortunately, Feline's extra setup is easy to hide in pipe stages, which exacts a chip real estate cost but not a performance cost. In today's ASIC technology, even Table Feline's setup logic is quite acceptable. The ellipse parameter table requires 1024 bytes, and 8 x 8 multipliers are small. Since much of Feline's setup can be performed in parallel with the perspective divide pipeline, it increases pipeline length over Texram by only a few stages. Feline's setup costs are substantially smaller than mip-mapped EWA's.

The increased texel fetching inherent to anisotropic texture mapping increases the cycles required to texture a pixel. This cost is impossible to eliminate, and difficult to keep small without sacrificing visual quality.

Both Feline and Texram access eight texels each probe. Since it is easy to compute each probe's location, and efficient trilinear filtering is well understood, we assume both algorithms can perform one probe per cycle. Any higher performance requires duplicating large portions (100k to 200k gates) of the texture mapping logic. Fortunately, probes overlap substantially (especially in the smaller of the two mip-maps), and a texel cache [7][9] eliminates most redundant memory fetches. Thus, the demand upon memory system bandwidth does not scale directly with the number of probes per textured pixel.

Mip-mapped EWA fetches each texel at most once per pixel and samples a substantially larger area. To provide a lower bound on cycles/pixel, we instrumented "Optimistic EWA," which naively assumes we can sample 8 texels/cycle on all but the last cycle for each pixel. This assumption is quite aggressive. Unlike probe-based schemes, we don't know how to design hardware for EWA that quickly traverses an ellipse with perfect efficiency. Our "Realistic EWA" assumes that hardware traverses the ellipse using a 4 x 2 texel "stamp" for *u*-major ellipses, and a 2 x 4 stamp for *v*-major ellipses. Each cycle several of the stamp's texels usually lie outside the ellipse, averaging about three outside for highly eccentric ellipses, and over four outside for nearly circular ellipses. This substantially reduces the efficiency of an EWA implementation.

Figure 35 shows how many cycles/pixel each algorithm uses for different viewing angles of one exemplary

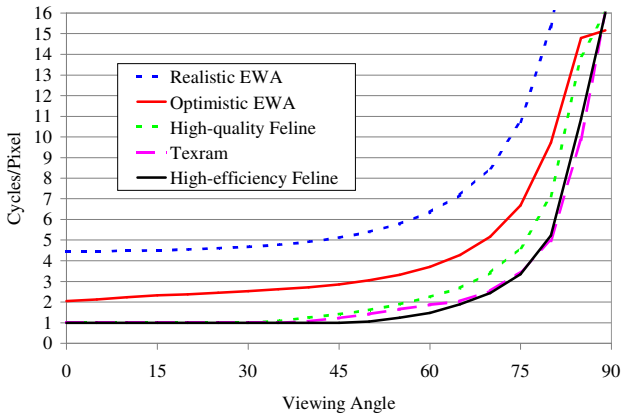

Figure 35: Performance at increasingly oblique viewing angles.

surface. At 0°, the surface normal is parallel to the viewing angle, and mip-mapped EWA samples the same size circle for each pixel. We made this circle's area the same as would be obtained by averaging results from randomly distributed viewing distances. This graph should be interpreted like EPA gas mileage numbers: it is useful for relative comparisons, but mileage will vary depending upon position on the screen, perspective distortion, etc.

Finally, note that if a scene uses multiple textures per surface, anisotropic texture mapping performance doesn't always slow down by these cycles/pixel ratios. For example, illumination maps tend to be small, so are usually magnified [7], which takes a single probe. They also tend to be blurry (that is, contain mostly low frequencies), so even when minified, an application might limit illumination mapping to one or two probes per pixel.

5. Conclusions

Feline provides nearly the visual quality of EWA, but with much simpler setup and texel visiting logic, and many fewer cycles per textured pixel. Feline provides better image quality than Texram, especially for repeated textures, even when limited to use the same number of probes. Feline requires somewhat more setup and texel weighting logic than Texram, but this cost is small compared to the increase in visual quality. Feline can be built on top of an existing trilinear filter implementation; for better results, the trilinear filter tree can also permit mip-mapped Gaussian filtering at little cost. Since several aspects of Feline are parameterized, Feline can gracefully degrade image quality in order to keep frame rates high during movement. This degradation might accentuate aliasing for irregular textures, in order to preserve image sharpness, and accentuate blurring for repeated regular textures, in order to avoid Moiré artifacts.

Table Feline proves visually superior to Simple Feline only for large perspective distortions occur when projecting images onto surfaces in a 3D scene. It requires an ellipse table and a few pipe stages over Simple Feline.

In the Sep/Oct 1998 issue of *IEEE Computer Graphics and Applications*, Jim Blinn wrote in his column that "No one will ever figure out how to quickly render legible antialiased text in perspective. Textures in perspective will always be either too fuzzy or too jaggy. No one will ever build texture-mapping hardware that uses a 4x4 interpolation kernel or anisotropic filtering." Feline is simple enough to implement, yet of high enough visual quality, to prove him at least partially wrong.

6. Acknowledgements

Thanks to Paul Heckbert for answering questions and for providing us with the EWA source code, and to Gunter Knittel for answering questions about Texram. Thanks to Mark Manasse for suggesting the use of two trigonometric functions to reduce angular errors in inverse trig functions.

References

[1] Anthony C. Barkans. High Quality Rendering Using the Talisman Architecture. *Proceedings of the 1997 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 79-88. ACM, August 1997. ISBN 0-89791-961-0.

[2] Frank C. Crow. Summed-Area Tables for Texture Mapping. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH 84 Conference Proceedings)*, volume 18, pages 207-212. ACM, July 1984.

[3] Alain Fournier & Eugene Fiume. Constant-Time Filtering with Space-Variant Kernels. In Richard J. Beach, editor, *Computer Graphics (SIGGRAPH 88 Conference Proceedings)*, volume 22, pages 229-238. ACM SIGGRAPH, Addison-Wesley, August 1988. ISBN 0-89791-275-6.

[4] Ned Greene & Paul Heckbert. Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter. *IEEE Computer Graphics and Applications*, 6(6):21-27, June 1986.

[5] Paul S. Heckbert. *Texture Mapping Polygons in Perspective*, Technical Memo #13, NY Inst. Tech. Computer Graphics Lab, April 1983.

[6] Paul S. Heckbert. *Fundamentals of Texture Mapping and Image Warping* (Masters Thesis), Report No. UCB/CSD 89/516, Computer Science Division, University of California, Berkeley, June 1989.

[7] Homan Igehy, Matthew Eldridge, Kekoa Proudfoot. Prefetching in a Texture Cache Architecture. *Proceedings of the 1998 EUROGRAPHICS/SIGGRAPH Workshop on Graphics Hardware*, pp. 133-142. ACM, August 1998. ISBN 0-89791-1-58113-097-x.

[8] Robert C. Landsdale. *Texture Mapping and Resampling for Computer Graphics* (Masters Thesis), Department of Electrical Engineering, University of Toronto, Toronto, Canada, January 1991, available at ftp://dgp.toronto.edu/ pub/lansd/.

[9] Joel McCormack, Robert McNamara, Chris Gianos, Larry Seiler, Norman Jouppi, Ken Correll, Todd Dutton & John Zurawski. *Neon: A (Big) (Fast) Single-Chip 3D Workstation Graphics Accelerator*, WRL Research Report 98/1, Revised July 1999, available at www.research.digital.com/wrl/techreports/ pubslist.html.

[10] Andreas Schilling, Gunter Knittel & Wolfgang Strasser. Texram: A Smart Memory for Texturing. *IEEE Computer Graphics and Applications*, 16(3): 32-41, May 1996. ISSN 0272-1716.

[11] Lance Williams. Pyramidal Parametrics. In Peter Tanner, editor, *Computer Graphics (SIGGRAPH 83*

Conference Proceedings), volume 17, pages 1-11. ACM, July 1983. ISBN 0-89791-109-1.

[12] George Wolberg. *Digital Image Warping*, IEEE Computer Society Press, Washington, DC, 1990. ISBN 0-8186-8944-7.