High Quality Elliptical Texture Filtering on GPU

Pavlos Mavridis* Georgios Papaioannou†

Department of Informatics, Athens University of Economics & Business

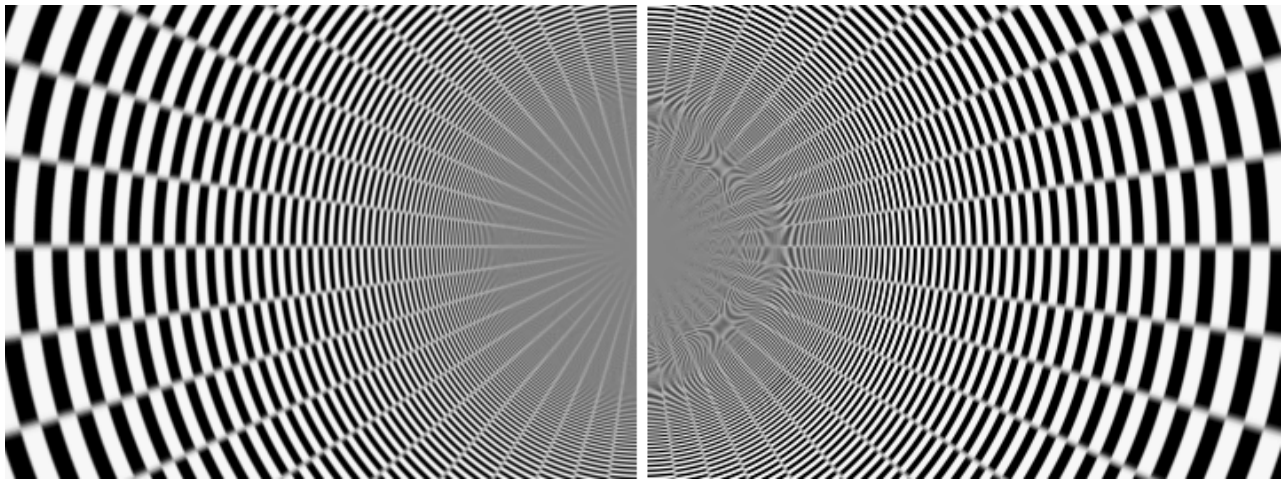

Figure 1: *An infinite tunnel scene demonstrating the improvement of our method over hardware texture filtering. Left: High quality elliptical texture filtering. Right: Hardware 16x anisotropic texture filtering on a NVIDIA GTX460 using the best possible quality settings.*

Abstract

The quality of the available hardware texture filtering, even on state of the art graphics hardware, suffers from several aliasing artifacts, in both spatial and temporal domain. Those artifacts are mostly evident in extreme conditions, such as grazing viewing angles, highly warped texture coordinates, or extreme perspective and become especially annoying when animation is involved.

In this paper we introduce a method to perform high quality texture filtering on GPU, based on the theory behind the Elliptical Weighted Average (EWA) filter. Our method uses the underlying anisotropic filtering hardware of the GPU to construct a filter that closely matches the shape and the properties of the EWA filter, offering vast improvements in the quality of texture mapping while maintaining high performance. Targeting real-time applications, we also introduce a novel spatial and temporal sample distribution scheme that distributes samples in space and time, permitting the human eye to perceive a higher image quality, while using less samples on each frame. Those characteristics make our method practical for use in games and other interactive applications. For cases where quality is more important than speed, like GPU renderers and image manipulation programs, we also present an exact implementation of the EWA filter that smartly uses the underlying bilinear filtering hardware to gain a significant speedup.

CR Categories: I.3.7[Three-Dimensional Graphics and Realism]: Color, shading, shadowing, and texture; I.3.3 [Picture/Image Generation]

Keywords: Texture filtering, Elliptical Weighed Average filter, EWA, MIP mapping, Antialiasing.

*e-mail: pmavridis@aueb.gr

†e-mail:gepap@aueb.gr

1 Introduction

Although the performance and the programmability of graphics hardware has vastly improved in the last decade, real-time rendering still suffers from severe aliasing. Both temporal and spacial aliasing artifacts can be seen on the edges of the polygons, on shadows, on specular highlights and on textured surfaces due to poor texture filtering. Those artifacts, among other issues of course, separate today's real-time rendering, used in games and other interactive applications, from the high quality rendering used in motion picture production. This paper addresses the last issue on the list, the bad texture filtering.

Bad texture filtering is evident as excessive blurring or moiré patterns in the spatial domain and as pixel flickering in the temporal domain, as can be seen in Figure 1 and the accompanying video. The causes of those artifacts are detailed in Section 3.

In this paper we present a high quality texture filtering algorithm that closely approximates the Elliptical Weighted Average (EWA) filter that was proposed by Greene and Heckbert [1986] [Heckbert 1989] and that runs on the programmable shading units of contemporary graphics hardware. EWA is regarded as one of the highest quality texture filtering algorithms and is used as a benchmark to test the quality of other algorithms. It is often used in off-line rendering to eliminate texture aliasing in extreme conditions such as

grazing viewing directions, highly warped texture coordinates, or extreme perspective. We also introduce a spatial and temporal sample distribution scheme that distributes the texture filtering samples both in space and time, resulting in higher quality filtering for a given number of texture samples. Along with the approximating algorithm, we also present an exact implementation of the EWA filter, that takes advantage of the linear and bilinear filtering of the GPU to gain a significant speedup.

We must make clear that the purpose of our research is to improve the quality of the available hardware texture filtering using the programmable shading units, and not to present a new texture filtering algorithm suitable for implementation using fixed function hardware. Apart from games, areas that directly benefit from this technique are high quality GPU renderers and GPU image manipulation software.

It should be noted that, although real production scenes do not use checkerboard patterns, aliasing also appears with typical textures but is mostly manifested in the form of pixel flickering when the camera or the objects move. However, in this paper, like in most of the publications in this research area, checkerboard patterns are used to better illustrate the problem in print.

The fastest proposed method runs at more than 50% of the speed of the available hardware anisotropic texture filtering, while offering a vast improvement in the image quality, as it is demonstrated in Figure 1 and the accompanying video.

2 Previous Work

Below we review the most important algorithms in texture filtering. Although most of the algorithms could be implemented in software, programmable shaders or fixed-function hardware, this classification is maintained in order to highlight the primary focus of the original algorithms. Software implementations usually focus on the quality, shader implementations aim to improve the filtering that is available on the underlying hardware and algorithms that target an implementation in hardware take into consideration the amount of logic gates needed to implement the proposed method. This research is directly related to the second category, but we borrow ideas from and extend algorithms in the other two areas.

2.1 Software Implementations

Summed Area Tables(SAT) [Crow 1984] and ripmaps [Larson and Shah 1990] allow the effective computation of anisotropic filtering in the horizontal and vertical directions. Apart from the limitation in the direction of anisotropy, the memory requirements are high, the pixels are assumed square with a quadrilateral footprint and SATs are limited to a box reconstruction filter, resulting in poor antialiasing performance in the general case.

EWA [Heckbert 1989] assumes overlapping circular pixels that project to arbitrarily oriented ellipses in texture space. EWA then computes a direct convolution of the texels inside the ellipse with a gaussian reconstruction filter. Compared to previous direct convolution methods [Feibush et al. 1988] [Perny et al. 1982], it offers similar quality but at much lower cost. The interested reader can refer to [Heckbert 1986] for a complete survey on direct convolution methods. Since our method is based on the theory behind EWA, we present the method in more detail in Section 3.

Ptex [Burley and Lacewell 2008] describes a way to filter per face textures in the context of a REYES [Cook et al. 1987] micropolygon renderer. Such a renderer shades regular micro-polygon grids in object space. The per-face textures of Ptex are aligned with the shading grids, resulting always in horizontal or vertical directions of anisotropy. Fast fully anisotropic filtering is performed on the filter region using separable filters. Unfortunately, this optimization cannot be applied on a renderer that shades pixels in screen space, which is our main target.

Recently a filter with better antialiasing performance than EWA was proposed [Lin et al. 2006], but the running time is much lower for a very marginal improvement in quality.

2.2 Programmable Shader Implementations

Bjorke [2004] describes a method for high-quality texture filtering with arbitrary filter kernels evaluated procedurally in the fragment shader. Similarly, Sigg and Hadwiger [2005] proposes a method to implement fast cubic filtering on GPUs, improving the quality of the hardware filtering. Both methods only consider isotropic projections, completely ignoring the anisotropy of the footprint of the projected pixel.

Novosad [2005] describes a primitive form of anisotropic filtering on GPU, but his method considers square pixels with a quadrilateral footprint, the sampling is performed at regular intervals without any guarantee that those intervals map to texel centers and the convolution is performed using a box filter. All these limitations result in images with several aliasing artifacts.

2.3 Fixed Function Hardware Implementations

TEXRAM [Schilling et al. 1996] approximates the pixel footprint with a parallelogram and uses several isotropic trilinear filtering probes along the major axis to compute a weighted average of the values inside the projected pixel area. The resulting image exhibits several aliasing artifacts, because the probes can be spaced too far apart and they have equal weights.

Feline [McCormack et al. 1999] improves the TEXRAM method by computing a closer approximation of the elliptical pixel footprint. The major axis of the ellipse is computed and several isotropic trilinear probes are taken with gaussian weights. The resulting image quality is improved over TEXRAM and is comparable with EWA filtering, given enough probes. A similar approach is vaguely described in [Akenine-Möller et al. 2008], but square pixels are assumed and no details are given about the weight or the location of those samples.

SPAF [Shin et al. 2001] filters texels in a region that covers a quadrilateral footprint with gaussian weights. The footprint coverage for each texel is computed with sub-texel precision and a gaussian weight is applied based on that coverage. Good quality is achieved for a restricted number of samples, but it assumes square pixels with quadrilateral footprint.

Our algorithm extends the Feline method in order to use the anisotropic probes of the underlying graphics hardware. Compared to an implementation of Feline or any other algorithm in programmable shaders, our algorithm makes better and more extensive utilization of the available hardware, by using the fast but low-quality hardware anisotropic texture filtering to reduce the number of required probes to match the shape of the EWA filter.

3 Elliptical Texture Filtering

In computer graphics a pixel is a point sample. When we refer to the area or the shape of a pixel, we are not referring to the physical properties of the pixels in the output device, but to the shape of the footprint (the non zero areas) of the reconstruction filter that is used to reconstruct the final continuous image from the point samples,

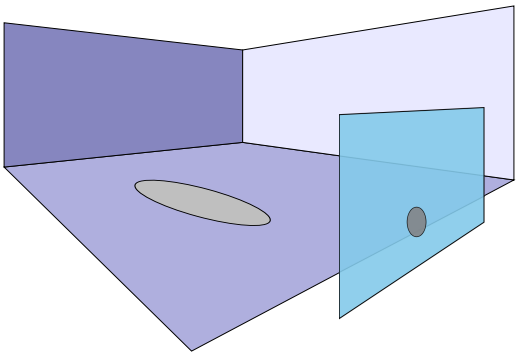

Figure 2: *The projection of a pixel with circular footprint in the world is an ellipse with arbitrary orientation (The transparent surface on the front represents the view plane).*

according to the sampling theorem. This common misconception is further discussed in [Smith 1995]. High quality reconstruction filters, like a truncated sinc or gaussian, have a circular footprint, so a high quality texture filtering method should assume circular overlapping pixels.

The projection of a pixel with circular footprint to texture space is an ellipse with arbitrary orientation, as illustrated in Figure 2. In degenerate cases, like extremely grazing viewing angles, the projection is actually an arbitrary conic section, but for our purposes an ellipse suffices, since at these cases any visible surface detail is lost anyway. A texture filtering algorithm should return a convolution of the texels (texture point samples) inside the projected area of the pixel with the projection of the reconstruction filter in texture space.

The main cause for the artifacts seen in texture filtering is poor approximation of the elliptical region with simpler shapes, like squares or quadrilaterals. This approximation facilitates the computation of the aforementioned convolution sum. The EWA filter directly performs this convolution, resulting in images of very high quality.

The elliptical footprint of a pixel in texture space, when the center of the pixel is translated to $(0, 0)$ in texture space, can be approximated by the following equation [Heckbert 1989]

$$d^2(u, v) = Au^2 + Buv + Cv^2 \tag{1}$$

with

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

The partial derivatives $(\partial u/\partial x, \partial u/\partial y, \partial v/\partial x, \partial v/\partial y)$ represent the rate of change of the texture coordinates relative to changes in screen space. The quantity d^2 denotes the squared distance of the texel (u, v) from the pixel center when projected back in screen space. The algorithm scans the bounding box of the elliptical region in texture space and texels inside the ellipse $(d^2 \leq 1)$ contribute to the convolution sum, with weights proportional to the distance d.

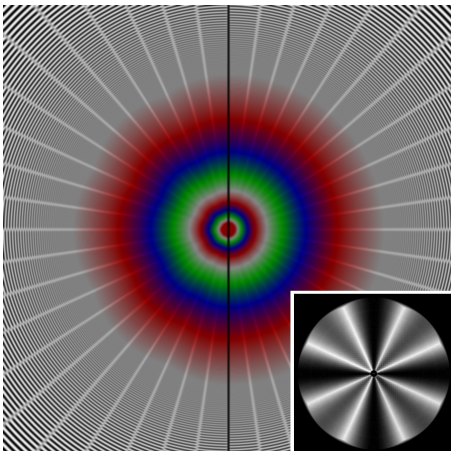

Figure 3: *Visualization of the mip-map lod selection in the infinite tunnel of Figure 1. Left: hardware filtering. Right: software filtering. The hardware performs a piecewise approximation of the optimal calculations, resulting in less detailed lod selection on pixels depending on their angle. Inset:The absolute error of the hardware lod computations, magnified seven times. The error is angle depended and peaks at intervals of 45 degrees.*

The original authors propose the usage of a gaussian filter, but in practice any reconstruction filter can be used.

The run time of the algorithm is directly related to the area of the ellipse in texture space and the number of texels it includes. To reduce the number of the texels in this area, a mip-map pyramid [Williams 1983] is used and sampling is performed from the mip-level level in which the minor ellipse radius is between 1.5 to 3 pixels, depending on the required quality and performance.

Even when using mip-maps, the area of a highly eccentric ellipse can be arbitrarily high, resulting in unacceptably high running times. To avoid this the maximum eccentricity of the ellipse is clamped to a predefined maximum.

Computing the mip-map level and clamping ellipses with high eccentricity requires the computation of the minimum and maximum radius of the ellipse

$$\begin{aligned} r &= \sqrt{(A - C)^2 + B^2} \\ R_{major} &= \sqrt{2/(A + C - r)} \\ R_{minor} &= \sqrt{2/(A + C + r)} \end{aligned} \tag{2}$$

Figure 3 visualizes the lod selection in the case of the ray traced tunnel of Figure 1, and compares it with the one performed by one of the latest graphics architectures (NVIDIA GTX460). We have found that NVIDIA's hardware clamps the ellipses at maximum anisotropy of 16, and selects the mip-map level in which the minor axis has a length of one. By using the same criteria in the software implementation, we have measured the absolute error of the hardware lod calculations. The lod calculations in NVIDIA's hardware perform a piecewise approximation of the optimal calculations, resulting in less detailed lod selection on pixels depending on their angle and the measured error peaks at intervals of 45 degrees. Clearly, the lod selection in the hardware implementation is tweaked for performance and not quality and therefore a high quality implementation cannot rely on the hardware lod selection to perform the texture filtering.

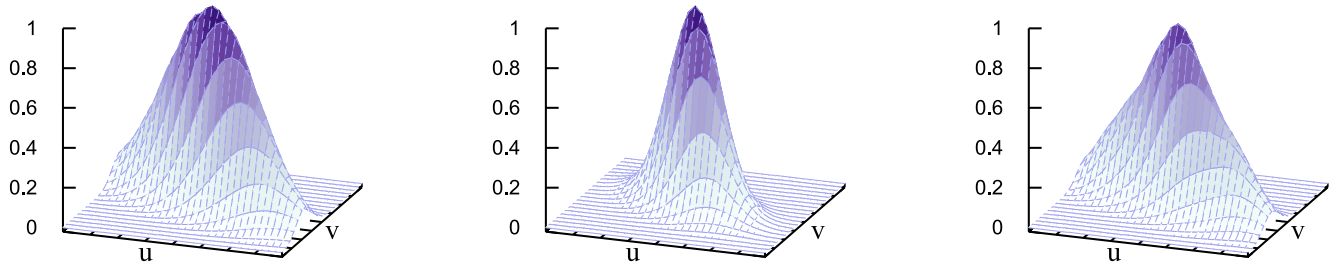

Figure 4: *Left: The ideal EWA filter projected in texture space. Middle: A small elliptical probe used by our method as a building block to approximate EWA. Right: The approximation of the EWA filter using* 3 *of the smaller elliptical probes shown in the middle.*

4 EWA Filter on GPU

Below we propose several methods for the calculation of the EWA filter on GPU. The first two perform a direct convolution of the texels inside the ellipse, while a third one approximates the shape of the ellipse using smaller elliptical shapes. Finally we propose two methods that spatially and temporally distribute the texture samples in order to achieve higher image quality for a given number of texture fetches.

A naive direct implementation of the EWA filter on GPU would read every texel in the bounding box of the elliptical region, and if it were located inside the ellipse, it would be weighted and accumulated. It is clear that the run-time performance of this algorithm is proportional to the number of texture fetches. We use this implementation as a reference to compare the rest.

4.1 Direct Convolution Using Linear Filtering

A much better approach is to use the linear filtering of the graphics hardware to reduce the number of fetches to one half, by smartly fetching two texels at a time using one bilinear fetch. For two neighboring texels C_i and C_{i+1}, the following weighted sum can be replaced by a single bilinear texture fetch operation at position x between the two texel centers:

$$\begin{aligned} w_i C_i + w_{i+1} C_{i+1} &= C_x (w_i + w_{i+1}) \\ x &= i + \frac{w_{i+1}}{w_i + w_{i+1}} \\ 0 \leq \frac{w_{i+1}}{w_i + w_{i+1}} &\leq 1 \end{aligned} \tag{3}$$

The last inequality is always true for reconstruction filters with positive weights, like the gaussian one.

This technique assumes that the cost of one texture fetch with bilinear filtering is less that the cost of two fetches with point sampling plus the time to combine them. Our results confirm that this assumption is true. The results from this method should be identical with the ones from the reference implementation, but slight deviations may occur due to the difference in the precision in which operations are performed by the shader units and by the fixed-function bilinear filtering units.

4.2 Direct Convolution Using Bilinear Filtering

The same method can be extended in two dimensions, by using bilinear filtering to fetch four samples at a time, further increasing the speed of the algorithm. To do this, the weighted sum of four samples that form a square footprint should be replaced by a single bilinear texture fetch inside this square. The

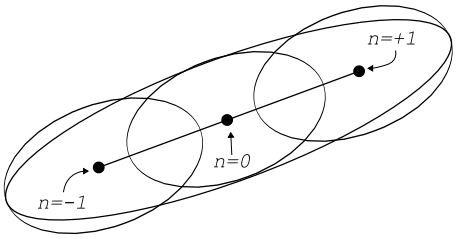

Figure 5: *The basic idea of our method: Approximating a highly eccentric ellipse with many ellipses of low eccentricity. This way we can overcome the limits imposed by the hardware and better filter elliptical footprints with high degree of anisotropy.*

weights $w_{00}, w_{10}, w_{01}, w_{11}$ of the four samples could be seen as the generalized barycentric coordinates of a quad with vertices $C_{00}, C_{01}, C_{10}, C_{11}$ and the weighted sum corresponds to a point C_{xy}. This point is inside the quad when $w_{00}+w_{01}+w_{10}+w_{11} = 1$. In the general case the sum of the weights equals W, so we must normalize the weights. The coordinates of point C_{xy} are then given by the following equation

$$w_{00} C_{00} + w_{01} C_{01} + w_{10} C_{10} + w_{11} C_{11} = W C_{xy}$$

Without loss of generality, we consider C_{ij} to be the edges of the unit square, and substituting we get:

$$\begin{aligned} x &= \frac{w_{01} + w_{11}}{W} \\ y &= \frac{w_{10} + w_{11}}{W} \\ W &= w_{00} + w_{01} + w_{10} + w_{11} \end{aligned}$$

We can now use the coordinate (x, y) to perform a bilinear texture fetch that approximates the weighted sum. Since x and y should fall within the $[0, 1]$ interval, this method is valid only for reconstruction filters with positive weights. The error in the approximation is proportional to the quantity $|w_{00}w_{11} - w_{01}w_{10}|$, because hardware bilinear filtering is equivalent to one horizontal and one vertical linear interpolation, assuming that w_{00} equals w_{01} and w_{10} equals w_{11}. Intuitively, the worst case happens when we have to fetch the samples at one diagonal of the quad, while the samples at the other diagonal have zero weights. In this rare case, the filter effectively widens the ellipse to include all four samples.

4.3 Ellipse Approximation

In the same spirit as [Schilling et al. 1996] and [McCormack et al. 1999], we propose a method that uses simpler shapes to closely match the shape of the ideal elliptical filter. Instead of using simple

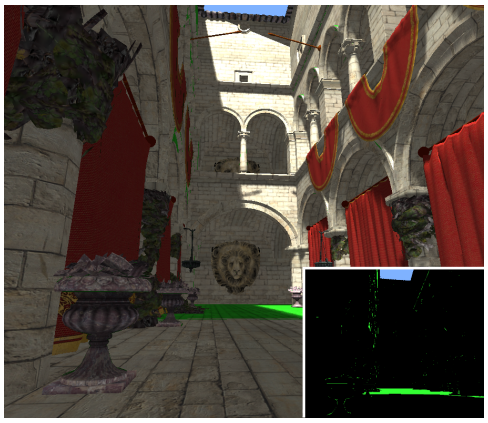

Figure 6: *A typical game environment. Regions with anisotropy higher than* 16 *are highlighted in green (better shown in the inset). In these regions the hardware filtering is prone to excessive blurring or aliasing artifacts, especially when using textures with high frequency components. The spatial distribution filter uses high quality filtering on these regions only.*

trilinear probes like the aforementioned algorithms, we propose the usage of the anisotropic probes provided by the graphics hardware.

We place the probes on a line along the major axis of the ellipse, as shown in Figure 5. The length of the line L and the number of probes N_{probes} are given by the following equations

$$\begin{aligned} N_{probes} &= 2 * (R_{major}/(\alpha * R_{minor})) - 1 \\ L &= 2 * (R_{major} - \alpha * R_{minor}) \end{aligned} \tag{4}$$

where α is the degree of anisotropy of the underlying hardware probes. For $\alpha = 1$ our algorithm is equivalent to Feline. For simplicity, an odd number of probes is considered. Similar to Feline, probes are placed around the midpoint (u_m, v_m) of the filter, as follows

$$\begin{aligned} \theta &= atan(B/(A-C))/2 \\ du &= cos(\theta) * L/(N_{probes} - 1) \\ dv &= sin(\theta) * L/(N_{probes} - 1) \\ (u_n, v_n) &= (u_m, v_m) + n/2 * (du, dv), \quad n = 0, \pm 2, \pm 4 \dots \end{aligned} \tag{5}$$

where (u_n, v_n) is the position of n-th probe. To better match the shape of the EWA filter, the probes are weighted proportionally to their distance from the center of the footprint, according to a gaussian function. The shape of the filter in texture space, compared to an ideal EWA filter is shown in figure 4.

We don't have any strong guarantees about the quality or the shape of those probes, since the OpenGL specification [Segal and Akeley 2006] does not explicitly enforce any particular method for anisotropic filtering. The only hint given by the hardware implementation is that the probes approximate an anisotropic area with maximum anisotropy of N. In the above analysis, we have assumed that the hardware anisotropic probes are elliptical, but in practice, to compensate for their potential imperfections in the shape and to better cover the area of the elliptical filter, we just increase the number of probes depending on the desired quality. Furthermore, using an adaptive number of probes creates an irregular workload per pixel for the graphics hardware scheduler, which should be avoided [Ragan-Kelley 2010]. In practice, setting the number of probes to a constant number gives better performance. In our tests, using

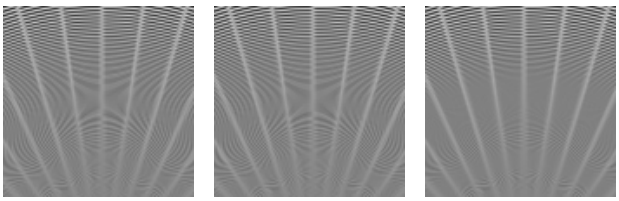

Figure 7: *From left to right: Frame n using a set of* 3 *samples, frame $n+1$ using another set of* 3 *samples, the average of the two successive frames, as perceived by the human eye when rendering at high frame rates. The third image exhibits less aliasing artifacts.*

5 probes eliminated all the visible artifacts, on the NVIDIA hardware. For more than 5 probes, no significant improvement in image quality could be measured.

If the probes fail to cover the ellipse, then aliasing will occur. On the other hand, if the probes exceed the extents of the ellipse (e.g. by placing them beyond half the length of the central line in each direction) then blurring will occur. Our method always avoids the second case, but the first case can still happen in extreme cases, since we have clamped the maximum number of probes. Still, our method always provides an improvement over hardware texture filtering.

4.4 Spatial Sample Distribution

Further examining the infinite tunnel scene of Figure 1, we observe that the regions where the hardware filtering fails are limited. For the majority of the scene the quality of the image is free of any aliasing artifacts. As expected, the problematic regions are regions with high anisotropy, and in particular on the NVIDIA hardware regions with anisotropy greater that 16, which is the advertised maximum anisotropy of the hardware unit. Figure 6 highlights the problematic regions on a normal game scene, to assess the extent of the problem on a typical game environment. We call α_0 the anisotropy threshold after which the hardware filtering of a particular GPU fails.

After this observation, we perform high quality filtering in regions above the threshold α_0, and for the rest of the scene we use hardware filtering. The anisotropy of a pixel is accurately measured using Equations 2. To eliminate any visible seams between the two regions, areas with anisotropy between $\alpha_0 - 1$ and α_0 use a blend of hardware and software filtering. This approach creates exactly two different workloads for the hardware, one high and one low. In the majority of cases the two different sampling methods are used in spatially coherent pixel clusters within a frame. Therefore, compared to a completely adaptive sample selection, our tests indicate that this case is handled more efficiently from the GPU hardware scheduler and results in a sizable performance gain.

4.5 Temporal Sample Distribution

In order to further improve the run-time performance of our filtering algorithm in games and other real-time applications that display many rendered images at high frame rates, we propose a temporal sample distribution scheme, where texture filtering samples are distributed among successive frames.

In particular, the n anisotropic samples (probes) of Equation 5 are distributed in two successive frames. The first frame uses samples $0, \pm 4, \pm 8 \dots$ and the next one $0, \pm 2, \pm 6 \dots$. The sample at the center of the filter (sample 0) is included in both frames to minimize the variance between the two. When the frames are displayed in quick succession, the human eye perceives the average of the

	reference	linear	bilinear	approximate	spatial	temporal	hardware 16x
MTexels/sec	93	127	149	658	774	1012	1701
Relative Perf.	5.5%	7.4%	8.8%	38.7%	45.5%	59.5%	100%
TEX lookup	adaptive	adaptive	adaptive	5	1 or 5	3	1
Perceptual diff	0	0	0	2	31	-	12029
Mean Sq. Error	0	0.007	6.4	30.0	55.2	-	126.3

Table 1: *Comparison between the proposed methods and the hardware 16x anisotropic filtering, using the highest possible quality settings on a NVIDIA GTX460. The direct convolution methods (reference, linear, bilinear) use a maximum anisotropy of* 32 *and the length of the minor radius of the projected ellipse at the selected mip-map level is one.*

two frames, as shown in Figure 7. To further improve the temporal blending between the frames, the set of samples used for filtering could vary depending on the pixel position, but our tests showed that the technique works surprisingly well even without these improvements.

For this technique to work, the application should maintain a stable and vertical sync locked frame rate of $60Hz$ or more. The usage of vertical sync is mandatory, otherwise the rate of the rendered frames will not match the rate they are displayed and perceived by the human eye and the method naturally fails. This is a shortcoming, but we should note that in the era of high performing graphics hardware, rendering without vertical sync makes little sense, since it introduces visible tearing. Obviously the method can be extended to distribute samples in more than two frames when the refresh rate of the application and the output device is high enough.

Using this method, the quality of texture filtering is enhanced considerably for static or distant objects, but fast moving objects receive less samples. This is hardly objectionable, since in that case potential aliasing artifacts are difficult to notice. In our tests, the usage of temporal sample distribution always improved the perceived image quality for a given number of samples.

Since the temporal jittering is performed along the axis of anisotropy in texture space, pixels with an isotropic footprint will not be affected by this method. Alternatively, the jittering could be performed in screen space, by jittering the pixel center between frames (by adjusting the projection matrix), but this method will affect all the pixels and is much more objectionable in moving objects and low framerates. Compared to temporal sample reprojection methods [Herzog et al. 2010], our method is much simpler, does not require additional memory and always provides a constant speedup. The obvious shortcoming is that the application should maintain a high frame rate.

5 Results

All the images presented in this paper are rendered using a gaussian reconstruction filter with a standard deviation of 0.5 and a width of one pixel. Adjusting the standard deviation of the filter affects the sharpness or the smoothness of the final image. A smaller standard deviation creates a sharper image, but may lead to aliasing artifacts when textures with high frequency components are used. In our tests, we have found that a standard deviation of 0.5 gives a nice tradeoff between aliasing and sharpness.

The spatial and temporal sample distribution schemes can be used to accelerate both the direct convolution methods and the ellipse approximation method. Since the first methods target the maximum possible quality, we only present results when distributing the samples of the ellipse approximation method.

Table 1 presents comprehensive performance and quality measurements of the methods proposed in this paper and compares them with hardware texture filtering. Performance was measured on the ray-traced tunnel scene of Figure 1 on a GTX 460 with a resolution of 900^2. The quality comparisons are based on the reference (naive) EWA implementation. We present both a perceptual error metric [Yee 2004] and the mean square error of the rendered frames. Quality metrics for the temporal sample distribution scheme have been omitted, because the perceived image quality involves multiple images.

To better assess the extent of the improvement our methods offers over hardware filtering, the readers are highly advised to see the accompanying video. Figure 8 shows a closeup of the tunnel scene, rendered with the methods proposed here. We observe that the direct convolution methods (the last three images) clearly give the best results. Compared to the other methods, less aliasing artifacts are visible and more detail is preserved at the leftmost part of the image, the center of the tunnel. At this area, as the tunnel tends to infinity, pixels tend to have infinite anisotropy, posing a serious challenge for texture filtering algorithms. The ellipse approximation method has some aliasing artifacts and the details on the left are blurred out, but still provides a tremendous improvement over the hardware filtering. The spatial and temporal filters give similar results. Finally, we can see that the hardware filtering exhibits several aliasing artifacts.

The quality of the algorithms when measured by the MSE metric, scales almost linearly with the performance. In other words, increasing the computation time decreases proportionally the measured error. On the other hand, using the perceptual metric indicates a huge gain on the perceived quality for a moderate performance drop.

The linear filtering implementation gives almost the same results as the reference one, with very small measured error, which indicates that on the latest graphics hardware the texture filtering is performed at high precision. In should be noted that older hardware may use lower precision for such calculations. The bilinear filtering implementation introduces some errors, because the gaussian weights are approximate, but the resulting image is perceptually the same as the reference implementation.

The performance of the *spatial distribution filter* obviously depends on the ability of the GPU scheduler to handle irregular workloads and on the actual amount of regions with high anisotropy on a particular scene. Typical game scenes will have fewer highly anisotropic areas that the infinite tunnel we are making the tests with, and we expect the performance of the method to be even better.

Using the approximated hardware lod computations instead of the accurate ones did not result in any significant speedup in the case of the direct convolution methods (only 3%), something expected since the performance of these methods is mostly limited by the texture fetches. The other methods always implicitly use the hardware mip-map selection method.

Interestingly we have not seen any major performance gain by com-

bining the spatial and temporal filter. We have found that, for a single texture per pixel, after 1012Mtexels/sec (around 1250fps on 900^2 pixels) the performance of these filters is limited by ALU computations and not texel fetches. Our proof of concept implementation does not make any effort to optimize the computational part of the shader, but concentrates mainly at reducing the required texel fetches. To get better performance, further optimizations should try to reduce the computations performed by the shader. The results of course will vary depending on the ALU/TEX units ratio of a particular GPU.

5.1 Integration with Game Engines

The proposed texture filtering algorithms were implemented completely in GLSL v4 as a drop-in replacement to the build-in texture lookup functions. Integration with game engines and other applications is trivial, since only the calls to the corresponding texture functions should be changed, and also the current frame number should be exposed inside the shaders in the case of temporal filtering. Game engines have the option to globally replace all the texturing function calls with the enhanced ones, or at content creation time the artists could selectively apply the enhanced texture filtering on certain problematic surfaces. The second option can lead to better run-time performance, but at the cost of increased authoring time. We believe that the performance is high enough for the methods to be used globally in games that have enough performance headroom. Ideally, this could be accomplished transparently in the graphics drivers with a user selectable option.

5.2 Discussion

We observe that even the reference *EWA* implementation exhibits some aliasing artifacts. The elliptical region of the *EWA* filter is defined by a contour of the gaussian kernel, but since a true gaussian (or sinc) has infinite extend, direct EWA only uses a windowed filter, resulting in some aliasing artifacts. Extending the window size and the standard deviation of the filter will reduce the aliasing by introducing some blur.

Instead of using a better texture filtering method, another option to reduce the aliasing artifacts in textures is to use more shading samples per pixel (supersampling). Although supersampling is easier to implement, it is essentially a brute force solution to the problem, with significantly higher fill rate and pixel shading requirements. It is much more efficient to address the aliasing issue separately for the textures, the polygon edges and the other problematic areas, than to take more shading samples per pixel.

Methods like Variance Shadow Maps [Donnelly and Lauritzen 2006] and Lean Mapping [Olano and Baker 2010] use traditional anisotropic texture filtering to efficiently antialias shadows and specular highlights respectively. Our method can be used in those cases too, to offer better antialiasing performance.

Furthermore, alias-free texture filtering is essential for high quality GPU renderers and of course for the GPU image manipulation programs that have emerged recently. For this category of applications, the proposed direct convolution methods are more suitable because of the superior antialiasing performance. For applications demanding high run-time performance, like games, the approximate EWA filtering with our proposed optimizations is more suitable.

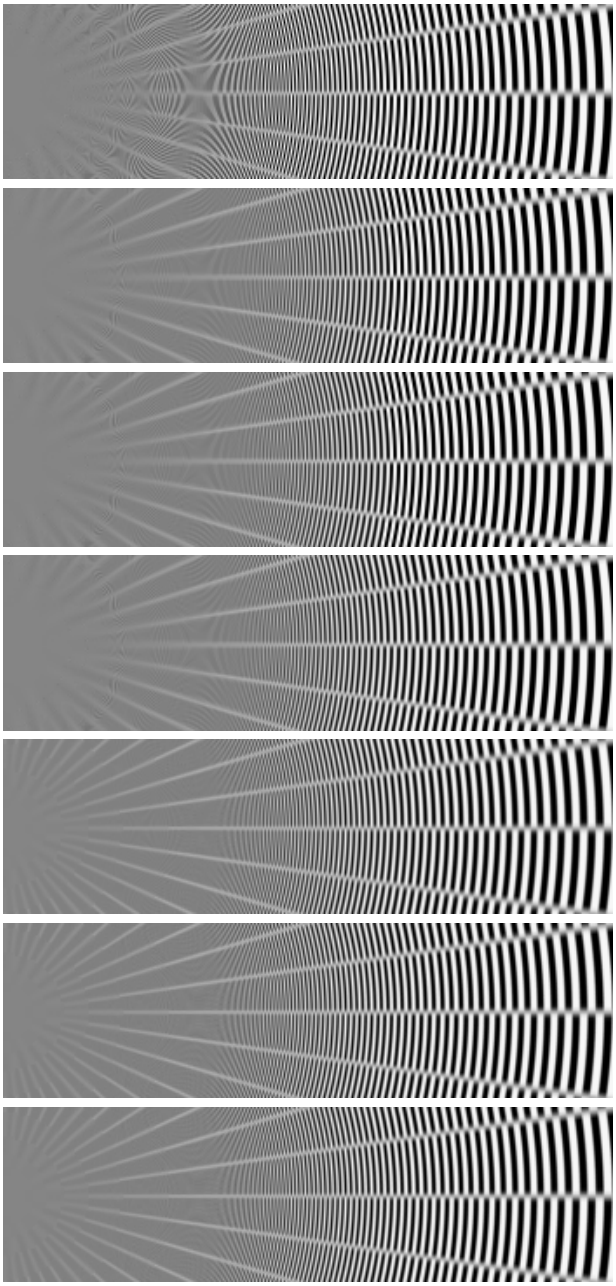

Figure 8: *Closeup of the infinite tunnel scene rendered using the proposed methods. From top to bottom: 16x hardware anisotropic, temporal filter (average of two frames), spatial filter, approximated ewa, bilinear ewa, linear ewa, reference ewa. All of the proposed methods offer an improvement over the hardware filtering, removing most of the aliasing artifacts. The direct convolution methods (the last three images) offer the best antialiasing quality, but with a performance hit.*

6 Conclusion

In this paper we have demonstrated that high quality elliptical texture filtering is feasible in real-time on contemporary GPUs, using

either accurate direct convolution approaches or a much faster approximation of the EWA filter. We have also presented a spatial and a temporal sample distribution scheme, the first distributing more samples in areas with high anisotropy, the latter distributing samples between frames, improving the perceived quality for a given number of samples.

We believe that high quality alias-free texture filtering is the first step for the real-time rendering to match the quality of prerendered graphics. In the future we indent to address in a practical way the aliasing problems in other aspects of the real-time rendering, like shadows.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. The sponza atrium model was kindly provided by Crytek. The raytracing code for the infinite tunnel examples was originally written by Inigo Quilez as part of the ShaderToy. The current research was supported by the Athens University of Economics and Business Research Fund (BRFP3-2010-11).

References

AKENINE-MÖLLER, T., HAINES, E., AND HOFFMAN, N. 2008. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA.

BJORKE, K. 2004. High-quality filtering. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addison Wesley, 391–415.

BURLEY, B., AND LACEWELL, D. 2008. Ptex: Per-face texture mapping for production rendering. In *Eurographics Symposium on Rendering 2008*, 1155–1164.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The reyes image rendering architecture. *SIGGRAPH Comput. Graph. 21*, 4, 95–102.

CROW, F. C. 1984. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 207–212.

DONNELLY, W., AND LAURITZEN, A. 2006. Variance shadow maps. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 161–165.

FEIBUSH, E. A., LEVOY, M., AND COOK, R. L. 1988. Synthetic texturing using digital filters. 275–282.

GREENE, N., AND HECKBERT, P. S. 1986. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Comput. Graph. Appl. 6*, 6, 21–27.

HECKBERT, P. S. 1986. Survey of texture mapping. *IEEE Comput. Graph. Appl. 6*, 11, 56–67.

HECKBERT, P. S. 1989. Fundamentals of texture mapping and image warping. Tech. Rep. UCB/CSD-89-516, EECS Department, University of California, Berkeley, Jun.

HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2010. Spatio-temporal upsampling on the GPU. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, 91–98.

LARSON, R., AND SHAH, M. 1990. *Method for generating addresses to textured graphics primitives stored in rip maps*. United States Patent 5222205.

LIN, Z., WANG, L., AND WAN, L. 2006. First order approximation for texture filtering. In *Pacific Graphics 2006*.

MCCORMACK, J., PERRY, R., FARKAS, K. I., AND JOUPPI, N. P. 1999. Feline: fast elliptical lines for anisotropic texture mapping. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 243–250.

NOVOSAD, J. 2005. Advanced high-quality filtering. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison Wesley, 417–435.

OLANO, M., AND BAKER, D. 2010. LEAN mapping. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, 181–188.

PERNY, D., GANGNET, M., AND COUEIGNOUX, P. 1982. Perspective mapping of planar textures. *SIGGRAPH Comput. Graph. 16*, 1, 70–100.

RAGAN-KELLEY, J. 2010. Keeping many cores busy: Scheduling the graphics pipeline. In *SIGGRAPH '10: ACM SIGGRAPH 2010 Courses*, ACM, New York, NY, USA.

SCHILLING, A., KNITTEL, G., AND STRASSER, W. 1996. Texram: A smart memory for texturing. *IEEE Computer Graphics and Applications 16*, 32–41.

SEGAL, M., AND AKELEY, K. 2006. The OpenGL graphics system: A specification: Version 4.1. Tech. rep., Silicon Graphics Inc., December.

SHIN, H.-C., LEE, J.-A., AND KIM, L.-S. 2001. SPAF: sub-texel precision anisotropic filtering. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM, New York, NY, USA, 99–108.

SIGG, C., AND HADWIGER, M. 2005. Fast third-order texture filtering. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison Wesley, 313–329.

SMITH, A. R. 1995. A pixel is not a little square, a pixel is not a little square, a pixel is not a little square! (and a voxel is not a little cube). Tech. rep., Technical Memo 6, Microsoft Research.

WILLIAMS, L. 1983. Pyramidal parametrics. *SIGGRAPH Comput. Graph. 17*, 3, 1–11.

YEE, H. 2004. A perceptual metric for production testing. *journal of graphics, gpu, and game tools 9*, 4, 33–40.