Vertex-based Anisotropic Texturing

Marc Olano, Shrijeet Mukherjee, Angus Dorbie [*]
SGI

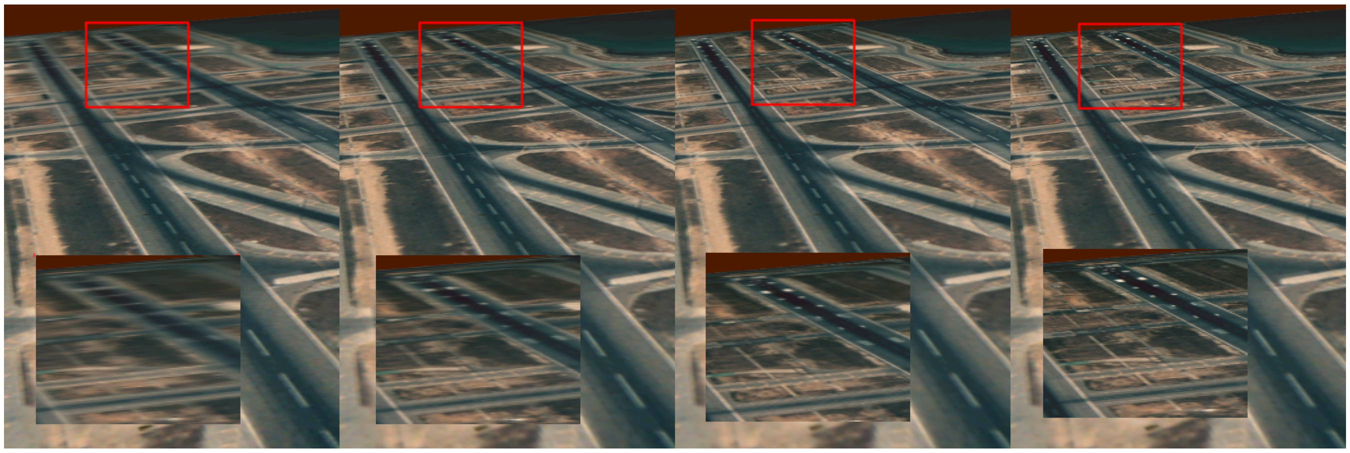
Figure 1: An airport runway texture seen with MIP mapping and anisotropic texture filtering with two, four and eight samples

Abstract

MIP mapping is a common method used by graphics hardware to avoid texture aliasing. In many situations, MIP mapping over-blurs in one direction to prevent aliasing in another. Anisotropic texturing reduces this blurring by allowing differing degrees of filtering in different directions, but is not as common in hardware due to the implementation complexity of current techniques. We present a new algorithm that enables anisotropic texturing on any current MIP map graphics hardware supporting MIP level biasing, available in OpenGL 1.2 or through the *GL_EXT_texture_lod_bias* or *GL_SGIX_texture_lod_bias* OpenGL extensions. The new algorithm computes anisotropic filter footprint parameters per vertex. It constructs the anisotropic filter out of several MIP map texturing passes or multi-texture lookups. Each lookup uses MIP level bias and perturbed texture coordinates to place one probe used to construct the more complex filter profile.

CR categories and subject descriptors: I.3.3 [Computer Graphics]: Picture/Image generation — Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism — Color, shading, shadowing and texture.

Keywords: Graphics Hardware, Interactive Rendering, Multi-Pass Rendering, Anisotropic Texturing, Footprint assembly.

[*]email:{olano,shm,dorbie@sgi.com}

1 INTRODUCTION

Texture mapping is a scene complexity multiplier. A complex image mapped onto simple geometry makes the whole scene look more complex. Graphics hardware generally uses MIP mapping to avoid texture aliasing artifacts [16]. MIP mapping provides a distance and angle variant isotropic low-pass filter. In other words, the MIP filter size is dependent on both the distance and angle at which a texture is seen, but gives the same amount of filtering in all directions across the texture. Proper rendering of stretched textures or textures seen from an angle requires anisotropic filtering.

We present a new anisotropic texture filtering algorithm based on multiple simpler texture accesses. It enables anisotropic texture filtering on existing platforms with hardware MIP mapping only, as well as general anisotropic texturing on platforms with only a limited degree of anisotropy.

1.1 Background

Texture mapping is common on graphics hardware because it provides detail even on simple geometry. Unfortunately, this same detail creates aliasing artifacts when seen in the distance. The surveys by Heckbert and Lansdale provide a good review of many texture filtering techniques that address this problem [7, 10].

MIP mapping is one of the earliest methods, and remains a common hardware solution for aliasing. A MIP map consists of a pyramid of pre-filtered and down-sampled textures. Each MIP level is half the size in both directions of the one before. Texturing hardware computes a texture scale factor per *fragment* (one screen sample that may be composed with others to produce a final pixel). The texture scale indicates which two MIP levels are closest to the desired filter width. The hardware may use one, two, four or eight of the surrounding eight texels to reconstruct the fragment color [15].

While MIP mapping works well for textures seen head on, it over-blurs textures seen at an angle, preventing aliasing in one direction at the expense of detail in the other direction. In essence, the texture is squished more in one direction than another, but MIP

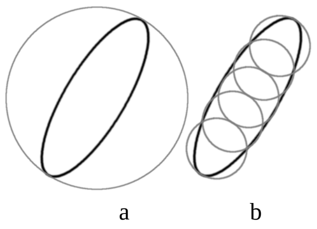

Figure 2: Assembling elliptical filter footprints (black) out of isotropic MIP texture probes (grey). Grey areas outside the black footprint indicate over-blurring, black areas outside the grey footprints indicate potential aliasing. a: ordinary MIP mapping. b: normal footprint assembly sampling pattern. c: under-sampling when probe size is wrong. d: compensating with off-axis probes.

mapping only handles even (isotropic) texture reduction. The solution is to use anisotropic texture filtering. Figure 1 illustrates the problem for an airport runway texture, as well as the results of our algorithm. At this viewing angle, runway markings that should be visible are completely lost by MIP mapping, but preserved by anisotropic texturing.

Summed-area tables [4] and RIP mapping [7, 11] were early methods for anisotropic texturing. Both provide anisotropic filtering when the compression is along one axis of the texture, but perform as poorly as MIP mapping along the texture diagonal.

Heckbert's elliptically weighted average (EWA) method offers a better solution by using an elliptical Gaussian filter in texture space [8]. The ellipse approximates the projection of a circular Gaussian pixel filter from screen space to texture space. The elliptical filter can be constructed by integrating raw texels, or by combining several MIP map samples within the filter footprint (a process called *footprint assembly*).

Variations of footprint assembly to approximate an elliptical filter have proven most popular for attempts at specialized anisotropic texturing hardware [1, 2, 5, 12, 14]. Footprint assembly-based algorithms provide relatively good filtering with only a few MIP map samples. McCormack presents a good comparison of several footprint assembly methods [12]. No commercial implementations exist for most of these. The NVIDIA GeForce is limited to a 2:1 ratio between the major and minor axis lengths. Currently, only the ATI Radeon supports a degree of anisotropy up to 16:1.

Other more complex and accurate techniques can produce better results but are not as well suited to hardware implementation [3] or require many more texture accesses [6].

1.2 Our Contribution

We propose a new anisotropic texturing algorithm that performs footprint assembly using multi-texture lookups and/or multiple passes under application control. The algorithm enables anisotropic texturing on graphics hardware that does not natively support it, and enables a greater degree of anisotropy on hardware with limited anisotropic texturing. The maximum degree of anisotropy is limited only by the number of rendering passes used.

If multi-texture lookups are present [13], they can be used to divide the total number of passes required. If vertex shading is also present [9], it can be used to move most of the (now per-vertex) footprint computations into the graphics hardware.

The overall computational costs are reduced since the elliptical footprint parameters are computed with lower frequency (per vertex instead of per fragment), though the practical effect of this result is depends on the balance of host, vertex and fragment processing power and load.

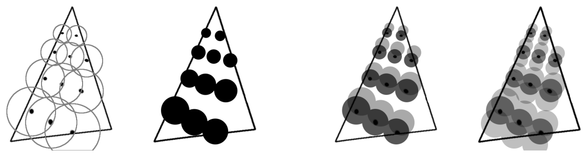

Figure 3: Representation of probe assembly across a triangle in texture space (probe size not accurate). a: ordinary MIP texturing. b: biased center probe. c: blend with second shifted probe. d: blend with third probe.

Our new algorithm does require the ability to bias the MIP map level that would otherwise be chosen by the texturing hardware. This feature is available in OpenGL 1.2 or as an OpenGL extension on a variety of hardware platforms, from the NVIDIA GeForce to the SGI Onyx 3000 [13, 15].

The algorithm is described in Section 2, our experimental results are in Section 3, and some final conclusions appear in Section 4.

2 MULTIPASS FOOTPRINT ASSEMBLY

The basis of EWA is a circular Gaussian filter kernel covering one screen fragment. When a surface is viewed at an angle, the projection of this kernel into texture space can be approximated by an elliptical Gaussian (represented by a black contour in Figure 2). The grey circle in Figure 2a shows how the standard MIP footprint avoids aliasing in the direction of most texture variation, but samples a large area of texture outside the desired elliptical footprint. This translates into excess blurring of the texture.

Footprint assembly approximates the elliptical Gaussian by a weighted sum of texture samples (also called probes or taps). Figure 2b shows the 'ideal' probe pattern suggested by the Feline authors [12]. Probes are sized to the ellipse minor (short) axis, spaced along the ellipse major (long) axis, and weighted to approximate the Gaussian filter profile (not shown). The main difference between footprint assembly algorithms is probe placement. Both the ideal algorithm and hardware implementations compute the probe placement per fragment, but designs for hardware use simpler computations to get close to the true ellipse axis without the full eigenvector computation (detailed elsewhere [7, 12]).

For vertex-based anisotropic texturing, we instead compute the probe placement per vertex. The probe size is set using a MIP map level bias. An ordinary textured draw gives a set of large MIP probes centered on each fragment across the geometry (Figure 3a). MIP level bias scales the probe size relative to the size standard MIP mapping would have chosen. With MIP bias, we instead get one smaller probe at each fragment (Figure 3b). To place a probe within the texture area covered by the fragment, we perturb the texture coordinates at each vertex a small amount in the desired direction (Figure 3c, d).

The probes are weighted by blending with either a per-vertex or per-pass alpha. Any set of weights can be computed by working from the alpha for the last sample backwards to the first. Table 1 shows the results for three samples with two choices of final weights. For example, for three equal weight probes, the first is drawn using the alpha associated with the geometry. The second uses an alpha of $1/2$, for an even weight of $1/2$ for both. The third uses an alpha of $1/3$, resulting in $1/3$ weight for the third probe and $1/2 * 2/3 = 1/3$ weight for the other two.

2.1 Per-Vertex Computation

The per-vertex computations introduce two issues that don't appear in per-fragment algorithms. First, a single polygon may span

Probe #	1	2	3	1	2	3
Desired weights	$1/3$	$1/3$	$1/3$	$1/4$	$1/2$	$1/4$
Rendering Alpha	1	$1/2$	$1/3$	1	$2/3$	$1/4$
Pass 1 Contribution	1	0	0	1	0	0
Pass 2 Contribution	$1/2$	$1/2$	0	$1/3$	$2/3$	0
Pass 3 Contribution	$1/3$	$1/3$	$1/3$	$1/4$	$1/2$	$1/4$

Table 1: Two cases for alpha blending three probes, the rendering alpha applies to the new probe and 1-alpha is the multiplier for all old probes.

a range of texture scales and different degrees of anisotropy. We may have a polygon with 2:1 anisotropy at the near vertex and 8:1 or worse at the far vertex. Second, the direction of anisotropy may change. For example, it could line up with the s texture direction at one vertex and t at another.

While the degree of anisotropy varies across a polygon, we are limited to a single choice for the number of texture probes and MIP level bias. Thus, a successful per-vertex algorithm must correctly handle having the wrong number of texture probes at the wrong size (Figure 2c). In our implementation, we choose the number of probes and MIP bias once per-object, not just per-polygon.

The direction of anisotropy may also vary across a polygon. If this change is not smooth, we get meaningless offsets in the middle of the polygon (Figure 4a,b). With a per-vertex algorithm, the probe weights and texture offsets for each texture pass must be consistent across the polygons.

These concerns lead us to use a different sampling strategy than has been used by per-fragment footprint assembly algorithms. We solve both per-vertex issues by sampling the entire footprint instead of just placing probes along the major direction of anisotropy (Figure 2d/Figure 4c-f). This allows adequate reconstruction of a variety of filter shapes, from circular to highly eccentric, with the same set of probes and weights.

To compute the probe pattern, we establish a simple transformation between a circular Gaussian filter and the desired elliptical filter. Probe positions are transformed from the circular to elliptical domain. Weights come directly from the probe's position in the circular Gaussian. To ensure adequate overlap of samples, we choose a MIP bias based on the distance to other nearby samples:

$$MIPbias = \log_2 \left(\frac{probespacing}{MIPspacing} \right)$$

2.2 Sampling Distribution

This leaves one degree of freedom in the algorithm — the probe sampling pattern. This can be decomposed into two parts, the sampling pattern within a cannonical circular domain and the transformation from circle to ellipse.

One choice of transformation aligns the major axis of the ellipse with the x-axis in the circular domain and the minor axis with the y-axis (Figure 4a, Table 2). This choice is most similar to the axis computations for classic footprint assembly, but it is not well defined (or well behaved) for near-circular footprints. The selection of what axis is longer may change rapidly and unpredictably.

The transformation from screen to texture that originally defined the filter ellipse provides a much more stable coordinate system (Figure 4c), but one that requires a particularly dense sampling to avoid directional artifacts (Figure 4d). Note that using the *Screen* coordinate system is equivalent to implementing screen-based super-sampling in software.

We can get a more stable coordinate system that still tracks the direction of anisotropy by not sorting the major and minor ellipse

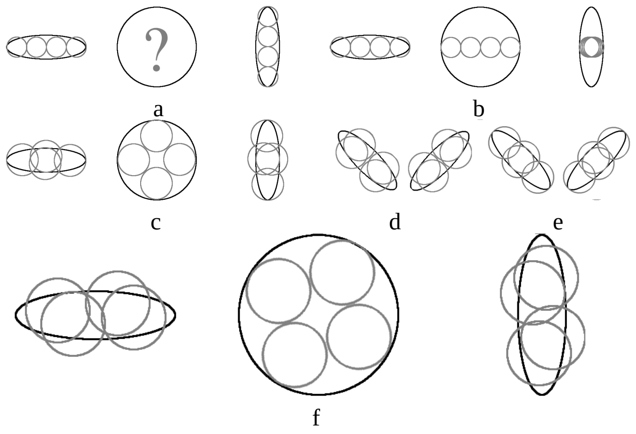

Figure 4: Sampling patterns to assemble elliptical filter footprints (black) from MIP texture probes (grey, drawn here without overlap for clarity). a: along major axis of ellipse, poorly behaved as direction of anisotropy changes. b: avoiding switch in axis, severe aliasing. c: sampling full footprint. d: pattern c as ellipse orientation rotates. e: pattern c rotated to align with direction of anisotropy. f: final pattern.

System	Ellipse	Screen	Max
Axis 1	$Acos\theta$	$\partial s/\partial x$	$max(\partial s/\partial x, \partial s/\partial y)$
	$Asin\theta$	$\partial t/\partial x$	$max(\partial t/\partial x, \partial t/\partial y)$
Axis 2	$-Bsin\theta$	$\partial s/\partial y$	$min(\partial s/\partial x, \partial s/\partial y)$
	$Bcos\theta$	$\partial t/\partial y$	$min(\partial t/\partial x, \partial t/\partial y)$

Table 2: Coordinate systems. Each system has two axes and each axis has a s and t components. Partials are elements of the pixel-to-texture Jacobian, A and B are the lengths of the major and minor ellipse axes, and θ is the angle subtended by the major axis

axes. This is prone to serious aliasing if used with a line of samples (Figure 4b), but behaves much more reasonably given a symmetric sampling pattern (Figure 4c,e).

Our final selection for sampling pattern uses a staggered arrangement (Figure 4f). This avoids the doubling-up that occurs in the center of the axis-aligned pattern. We also choose an approximate coordinate system that is somewhat easier to compute and behaves well as it is interpolated across the polygons (*Max* in Table 2).

3 RESULTS

We have tested this algorithm on SGI Onyx 3000, SGI Onyx2, SGI Octane2 and NVIDIA GeForce graphics subsystems. The Onyx and Octane2 do not support anisotropic rendering in the rasterizer, whereas the GeForce could have fixed 2:1 anisotropy (which we did not use for these tests).

Our tests on the SGI Onyx 3000 and Onyx2 used the *SGIS_multisample* extension. *SGIS_multisample* allows probe combination using separate screen super-samples instead of alpha blending. This is faster and retains full precision, but limited our tests to a maximum of eight probes. We used from one to eight probes for our tests on all systems. Higher numbers of probes are possible using alpha blending.

Both the geometry and pixel fill rendering requirements are multiplied by the number of passes used. All of our tests modeled the typical case of a large textured surface with low geometric complexity. Despite the object covering a large fraction of the screen, our eight rendering passes did not have a significant impact on the rendering frame rate.

We evaluated various coordinate systems and sample locations

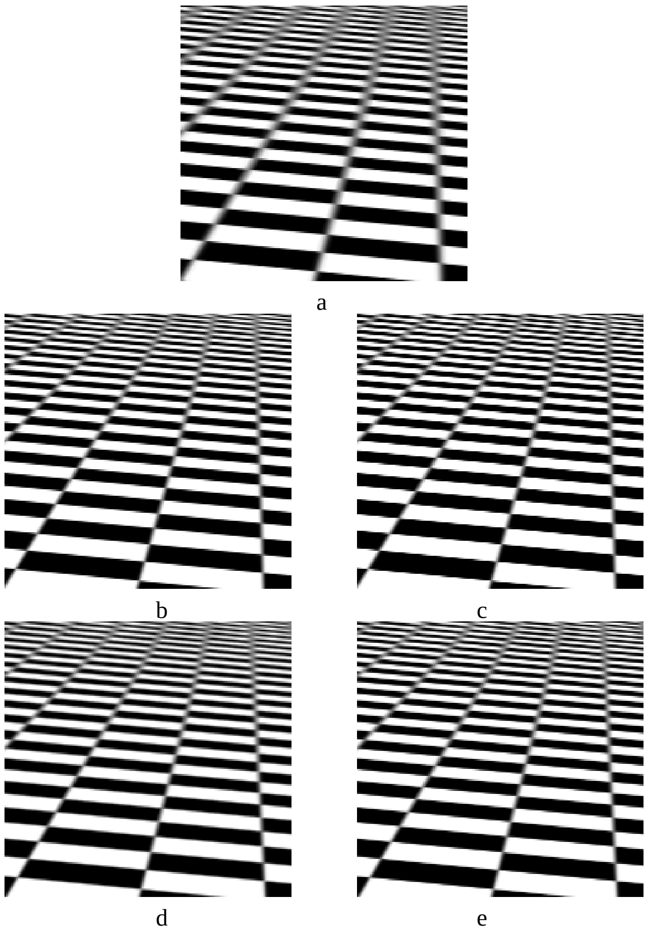

Figure 5: Effect of sampling pattern and MIP bias. a: normal MIP mapping, notice blurring of foreground edges and background. b: correctly sized probes along axis of anisotropy. c: same size probes (now too small) for pattern covering full footprint. d: larger probes along axis of anisotropy. e: larger (now correctly sized) probes covering full footprint

within them. Table 2 defines three of the more interesting coordinate systems. The *Screen* coordinate system is equivalent to supersampling the MIP mapped texture. It exhibited aliasing artifacts we attributed to directional effects shown in Figure 4d. The *Ellipse* coordinate system was superior from some directions, but had instabilities caused by by the singularity in Figure 4a. These singularities could be seen sweeping evenly across the polygons as the model rotated. The *Max* coordinate system produced good results. It is better behaved than the *Ellipse* system but is still oriented along the direction of maximum anisotropy.

We also experimented with different probe placements for each coordinate system. Our tests included probes along the major axis, jittered off of the major axis, at the end of both axes, at the corners of an axis-aligned square, and on a jittered grid within the circular footprint. Of these combinations, the best overall visual quality was achieved with the axis-aligned square pattern and *Max* coordinate system. Snapshots of some of these results are shown in Figure 5.

4 CONCLUSIONS AND FUTURE WORK

We have presented a new anisotropic texturing algorithm that works on a variety of hardware with limited or no native anisotropic texturing support. The algorithm composes multiple perturbed MIP map renderings to build an anisotropic filter kernel.

The algorithm requires the ability to bias the MIP level used in texturing, but this is a common extension to ordinary MIP map hardware found on a wide variety of platforms.

In the future, we would like to try using lower degree anisotropic samples as the probes instead of MIP samples. For example, the NVIDIA GeForce has native support for 2:1 anisotropy. We should be able to achieve convergence to the desired footprint with fewer probes when those probes are already oriented elliptical samples, amplifying the limited built-in anisotropy to something much better.

Our selection of coordinate systems and sampling patterns was somewhat ad-hoc. We achieved excellent results, but getting the best results for a given number of samples demands a more thorough examination of the choices. It is also possible that better results could be achieved by specifying different per-vertex weights and probe offsets within the circular kernel.

Many of the artifacts of per-vertex computation could be avoided through adaptive tessellation of the objects.

For much more accurate anisotropic texturing or other filter shapes, it should be possible to do per-vertex control point transformation for NIL mapping [6]. This is not as practical as footprint assembly for real-time use since it would require a much larger number of passes.

The algorithm could also be exploited for depth of field effects, blurring the texture more in front of and behind the focal plane.

Finally, as a tool for better performance, real-time scene management software could control the number of probes used for anisotropic texturing on a per object basis. Distant or less important objects may use fewer samples. This gives application control over the choice between fidelity and performance.

References

[1] ATI. Radeon charisma engine and pixel tapestry architecture. ATI White Paper, 2000.

[2] BARKANS, A. C. High-quality rendering using the talisman architecture. *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware* (August 1997), 79–88.

[3] CANT, R. J., AND SHRUBSOLE, P. A. Texture potential mip mapping, a new high-quality texture antialiasing algorithm. *ACM Transactions on Graphics 19*, 3 (July 2000), 164–184. ISSN 0730-0301.

[4] CROW, F. C. Summed-area tables for texture mapping. *Computer Graphics (Proceedings of SIGGRAPH 84) 18*, 3 (July 1984), 207–212.

[5] EVERITT, C. Anisotropic texture filtering in OpenGL. NVIDIA White Paper, September 2000.

[6] FOURNIER, A., AND FIUME, E. Constant-time filtering with space-variant kernels. *Computer Graphics (Proceedings of SIGGRAPH 88) 22*, 4 (August 1988), 229–238.

[7] HECKBERT, P. S. Survey of texture mapping. *IEEE Computer Graphics & Applications 6*, 11 (November 1986), 56–67.

[8] HECKBERT, P. S. Fundamentals of texture mapping and image warping. Master's thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, June 1989.

[9] KILGARD, M. J. NV_vertex_program extension specification. NVIDIA, 2000.

[10] LANSDALE, R. C. Texture mapping and resampling for computer graphics. Master's thesis, Department of Computer Science, University of Toronto, January 1991.

[11] LARSON, R. D., AND SHAH, M. S. Method for generating addresses to textured graphics primitives stored in RIP maps. US Patent 05222205, 1993.

[12] MCCORMACK, J., PERRY, R., FARKAS, K. I., AND JOUPPI, N. P. Feline: Fast elliptical lines for anisotropic texture mapping. *Proceedings of SIGGRAPH 99* (August 1999), 243–250.

[13] OPENGL ARB. Extension specification documents. http://www.opengl.org/, 2000.

[14] SCHILLING, A., KNITTEL, G., AND STRASSER, W. Texram: A smart memory for texturing. *IEEE Computer Graphics & Applications 16*, 3 (May 1996), 32–41. ISSN 0272-1716.

[15] SEGAL, M., AKELEY, K., FRAZIER, C., AND LEECH, J. *The OpenGL Graphics System: A Specification (Version 1.2.1)*. Silicon Graphics, Inc., 1999.

[16] WILLIAMS, L. Pyramidal parametrics. *Computer Graphics (Proceedings of SIGGRAPH 83) 17*, 3 (July 1983), 1–11.