Eurographics Symposium on Rendering 2014
Wojciech Jarosz and Pieter Peers
(Guest Editors)

Volume 33 (2014), Number 4

Bilinear Accelerated Filter Approximation

Paper 1019

(a) Input (b) Exact Lánczos 2 (c) Trilinear interpolation (d) CCTF (e) Our method

Figure 1: Approximations of downsampling a high-resolution input image (a) to 100^2 pixels using a Lánczos 2 filter are compared. The result from exact evaluation is shown in (b) and the approximations by (c) trilinear interpolation, (d) Cardinality-Constrained Texture Filtering (CCTF), and (e) our method all use the same mipmap. Trilinear interpolation appears blurry, whereas our approximation of the Lánczos 2 downsampled image is similar to the exact evaluation of a Lánczos 2 filter and CCTF. However, our method runs twice as fast as CCTF by constructing a filter from hardware accelerated bilinear texture samples.

Abstract

Our method approximates exact texture filtering for arbitrary scales and translations of an image while taking into account the performance characteristics of modern GPUs. Our algorithm is fast because it accesses textures with a high degree of spatial locality. Using bilinear samples guarantees that the texels we read are in a regular pattern and that we use a hardware accelerated path. We control the texel weights by manipulating the u,v parameters of each sample and the blend factor between the samples. Our method is similar in quality to Cardinality-Constrained Texture Filtering [MS13] but runs two times faster.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing

1. Introduction

High-quality texture filtering is important to the appearance of a rendered image because filtering reduces aliasing artifacts when textures are downsampled. Aliasing occurs when a scene contains higher-frequency details than are representable at the resolution of the screen. Antialiasing is often associated with smoothing jagged edges of polygons, but texture sampling provides an even more noticeable source of aliasing. Drawing antialiased edges only improves the profile of objects, whereas improved texture filtering benefits every textured pixel that is drawn.

Resizing an image requires only an isotropic filter. For 3D scenes, perspective projection creates a nonuniform distortion between 2D textures and the image rendered on the screen, so anisotropic filtering is commonly used to filter textures in 3D scenes. However, anisotropic filters are typically generated using multiple isotropic samples [MPFJ99, MS13], so isotropic filters are important in both 2D and 3D rendering. In this paper, we only consider optimizing isotropic filtering, but we show examples of our method applied to anisotropic filtering in Section 4.

Even isotropic filters are difficult to evaluate efficiently at arbitrary scales s and translations t. The ideal low-pass filter, *sinc*, is too expensive to evaluate because it has infinite support, so smaller filters such as the tent, Gaussian, and

Lánczos filters are used. Even filters with small support may sum hundreds of texels under modest scales. For example, a Lánczos 2 filter covers an area of 4×4 pixels, so downsampling by a factor of $s = 5$ sums over $4^2 \times 5^2 = 400$ texels for a single sample.

Precalculated pyramids of downsamplings at different resolutions, called mipmaps [Wil83], keep the cost of evaluating a texture sample constant. The position and scale of a new sample are unlikely to be identical to a precalculated sample in the mipmap, so the data is interpolated by trilinear interpolation. Trilinear interpolation is simple, defines a continuous transition between colors, and never reads more than 8 texels, which results in an interpolant that is fast enough for real-time rendering and that has been the *de facto* standard in computer graphics for the past 30 years.

A natural question arises: is it possible to devise a better approximant than trilinear interpolation? Cardinality-Constrained Texture Filtering (CCTF) [MS13] improves upon trilinear interpolation, but CCTF requires scattered texture reads that are inefficient for cached texture memory. Our method addresses this shortcoming by using bilinear samples to improve memory coherence while using the same memory bandwidth as trilinear interpolation. Combining GPU accelerated bilinear samples allows us to double the performance obtained with CCTF while accurately approximating filters. The result is high-quality renderings, as demonstrated in Figure 1. Like trilinear interpolation, we use two bilinear samples, but we solve for the best u, v coordinates and coefficients of each of the samples for different filter parameters s, t. By optimizing parameters and coefficients of bilinear reads, we achieve a 4×reduction in approximation error compared to trilinear interpolation while reading only eight texels per sample.

2. Related Work

The simplest method for sampling a mipmap is to return the color of the nearest neighbor. This method reads one texel and is fast, but the image is blocky because the nearest neighbor interpolant is discontinuous. Trilinear interpolation [Wil83, BA83] produces continuous changes in color, which looks better than nearest-neighbor interpolation, but it reads eight texels. Increasing the order of the interpolant to tricubic interpolation provides little visual benefit when downsampling an image because increasing smoothness does not necessarily reduce approximation error. However, bicubic interpolation does increase the quality of upsampled images [SH05].

A few methods improve the quality of isotropically downsampled images. Summed-area tables store the integral of pixel colors instead of a mipmap [Cro84]. These tables take significantly more space to store than the original image because each color component needs 32 bits in order to support texture sizes up to 4096^2 texels while maintaining 8 bits of precision. However, summed-area tables can evaluate the average pixel color within any rectangular region of texels using 4 table lookups, which, because of the quadrupled integer size, is equivalent to reading 16 texels. Constant-time evaluation with different filters can also be done by sampling from a mipmap level that has a resolution only a few factors higher than the sample being approximated [Hec89]. The sampled mipmap needs to be of sufficiently high resolution to capture features of the filter, and Heckbert suggests reading between 9 and 36 texels per sample. Another method that samples from a single mipmap level [HS99] stores a table of weights to approximate affine transforms of a box filter.

Methods that combine texels from multiple mipmap resolutions are more efficient. NIL mapping [FFB88] uses adaptive quadrature to approximate the more-important parts of a filter with high-resolution texels and approximates the remainder of the filter with low-resolution texels. CCTF [MS13] performs a linear cardinality-constrained optimization to determine which set of texel basis functions in a mipmap best reproduce a given filter. Any set of basis functions can be chosen, which means that texel reads are scattered both in position and mipmap resolution. Therefore, CCTF violates the data locality assumptions that are critical for cache performance. Scattered reads increase memory bandwidth because memory is read at the resolution of cache lines rather than texels. In contrast, our method uses cache-friendly bilinear reads, at the cost of precomputing the result of a nonlinear optimization. Although the trilinear interpolant is also composed of two bilinear reads, we produce higher quality images than trilinear interpolation because we perform a nonlinear optimization to find the best u, v parameters and coefficients for the bilinear samples.

Several papers on texture filtering improve the quality of anisotropic filtering [Gla86, GH86, Hec89, SKS96, CS97, HS99, MPFJ99, CS00, CDK04, ZLW06, MP11]. We do not directly consider anisotropic filters because multiple isotropic samples can be used to construct an anisotropic sample [MPFJ99]. A similar method of combining isotropic samples used in GPUs is called $N\times$ anisotropic filtering, which means that up to N isotropic samples are taken along a line to approximate an anisotropic filter. By improving the quality of isotropic samples with our method, we automatically improve the quality of anisotropic filtering, as demonstrated in the results section.

3. Filter Optimization

Sampling an image function $I(x)$ with a spacing s between samples can be modeled by multiplying by the Dirac comb function $\text{Ш}_s(x) = \sum_{t=-\infty}^{\infty} \delta(x - st)$ so that the sampled function is $\text{Ш}_s(x)I(x)$. We can represent a function by its frequency spectrum using the Fourier transform

$$\hat{I}(\omega) = \mathcal{F}\{I(x)\} = \int_{-\infty}^{\infty} I(x) e^{-2\pi i x \omega} \, dx.$$

The Fourier transform of Ш_s is $\mathcal{F}\{\text{Ш}_s(x)\} = \text{Ш}_{1/s}(\omega)$. Because multiplication becomes convolution under the Fourier transform $\mathcal{F}\{f(x)g(x)\} = (\hat{f} * \hat{g})(\omega)$, the frequencies in our sampled function are

$$\mathcal{F}\{\text{Ш}_s(x)I(x)\} = \sum_{t=-\infty}^{\infty} \hat{I}(\omega - \frac{t}{s}).$$

If the spectrum of the sampled function $\hat{I}(\omega)$ contains frequencies outside the range $\omega \in [-\frac{1}{2s}, \frac{1}{2s}]$, then interference between shifted copies of $\hat{I}(\omega)$ can occur. This interference is called aliasing and manifests itself as low-frequency patterns. Removing the unrepresentable frequencies implies sampling the product of $\hat{I}(\omega)$ and $\hat{h}_s(\omega)$ in the Fourier domain, where $\hat{h}_s(\omega)$ has a frequency cutoff of

$$\hat{h}_s(\omega) = \begin{cases} 1, & \text{if } -\frac{1}{2s} \le \omega \le \frac{1}{2s} \\ 0, & \text{otherwise.} \end{cases}$$

This ideal low-pass filter has the form $h_s(x) = \frac{1}{s}h(\frac{x}{s})$, where $h(x) = \frac{\sin(\pi x)}{\pi x}$ is called the *sinc* filter. The inverse Fourier transform of multiplication is convolution, so we need to take point samples from the convolved function $\mathcal{F}^{-1}\{\hat{I}(\omega)\hat{h}(\omega)\} = (I * h)(x)$. To evaluate a sample at offset t, we take the inner product $\langle (I * h_s)(x), \delta(x - st)\rangle = \int_{-\infty}^{\infty} I(x)h_{st}(x)\,dx$, where we define the filter for the sample with scale s and offset t as $h_{st}(x) = h_s(x - st)$. A practical problem is that *sinc* has infinite support, so we typically use low-pass filters with smaller supports. These filters include the box, tent, Gaussian, Mitchell-Netravali, and Lánczos filters, in which each filter offers trade-offs between blurriness, ringing, and size of support.

In order to accelerate texture lookups, we precalculate a mipmap of the texture, which consists of a series of images downsampled at power-of-two resolutions. Colors of texels in the mipmap are calculated as

$$v_{st} = \int_{-\infty}^{\infty} I(x)h_{st}(x)\,dx,$$

so summing a weighted set of mipmap samples $\sum_i c_i v_i$ samples the original image by new filter that is the weighted sum $\sum_i c_i h_i(x)$ of the filters $h_i(x)$ that were used to create the mipmap.

$$\sum_i c_i v_i = \sum_i c_i \int_{-\infty}^{\infty} I(x)h_i(x)\,dx = \int_{-\infty}^{\infty} I(x) \sum_i c_i h_i(x)\,dx$$

By controlling the coefficients c_i, we can make the filter $\sum_i c_i h_i(x)$ closely match any other filter $H(x)$. In particular, we are interested in matching the results from filters with scales and translations that are not precalculated in the mipmap. We find the best coefficients c_i by solving the least-squares minimization

$$\underset{c_i}{\text{argmin}} \int_{-\infty}^{\infty} \left(H(x) - \sum_i c_i h_i(x) \right)^2 dx.$$

Thus far, our derivation has assumed a 1D image, but the same logic applies in 2D. The difference is that, in 2D, we sample the image $I(x,y)$ over a 2D grid by a filter $h_{st_ut_v}(x,y)$. To provide good caching behavior, we want to approximate the filter using a small number of coefficients, and the coefficients should be clustered in space.

GPUs provide hardware acceleration for bilinear interpolated texture reads, which return a weighted combination of a quad of adjacent texels. Bilinear interpolation reads a single mipmap level at scale s and is controlled by the u, v parameters, which range from zero to the resolution of the mipmap image. The bilinear filter $b_{suv}(x,y)$ samples the input image $I(x,y)$ by combining texels weighted by the coefficients

$$\begin{aligned} b_{suv}(x,y) = \quad & (1-u_r)(1-v_r)h_{s,u_f,v_f}(x,y) \\ & + u_r(1-v_r)h_{s,u_f+1,v_f}(x,y) \\ & + (1-u_r)v_r h_{s,u_f,v_f+1}(x,y) \\ & + u_r v_r h_{s,u_f+1,v_f+1}(x,y), \end{aligned} \tag{1}$$

where $(u_f, v_f) = (\lfloor u - 1/2 \rfloor, \lfloor v - 1/2 \rfloor)$ and $(u_r, v_r) = (u - 1/2 - u_f, v - 1/2 - v_f)$. We subtract a value of $1/2$ to maintain the convention that GPUs interpolate between texel centers. In order to use the same memory bandwidth as trilinear interpolation, we optimize for the two bilinear samples that best reproduce a filter.

$$\underset{c,u_i,v_i,s_i}{\text{argmin}} \int_{\mathbb{R}^2} \left(H(x,y) - \sum_{i=1}^{2} c_i b_{s_i u_i v_i}(x,y) \right)^2 dxdy$$

To ensure that the sum of texel weights is one, we define $c_1 = c$, $c_2 = 1 - c$. Most of the parameters can be any real number, but the scales $s_1, s_2 \in \{2^i | i \in \mathbb{N}\}$ discretely choose which image to sample. Thus, there is a combinatorial choice of the resolutions to read from for each sample. We restrict the optimization to sample only from three different resolutions, which yields six combinations, a far-lower number than required for CCTF.

It is not sufficient to find the best fit of $H(x,y)$ for one scale and translation. Instead, we solve for coefficients that simultaneously minimize the error over all translations and scales. The minimization is simplified because the solutions are identical for integer texel translations and dyadic scales, so we solve only over a single texel domain that is tiled over the mipmap volume. We further subdivide the texel domain into subdomains, over which we fit polynomials for the best parameters. Suppose that α, β, and σ are parameters in the range $[0,1)$ that determine the translations α, β and the scale σ of the input filter $H_{\alpha\beta\sigma}$. We then represent each of the real parameters c, u_1, u_2, v_1, v_2 as a polynomial over the domain of α, β, σ with the form

$$c(\alpha,\beta,\sigma) = \sum_{i,j,k} c_{ijk}\alpha^i\beta^j\sigma^k,$$

where, without loss of generality, we show the polynomial expansion of the parameter c. For example, a linear polynomial has four coefficients of the form $c(\alpha,\beta,\sigma) = c_{000} + c_{100}\alpha + c_{010}\beta + c_{001}\sigma$, which means that there are a total

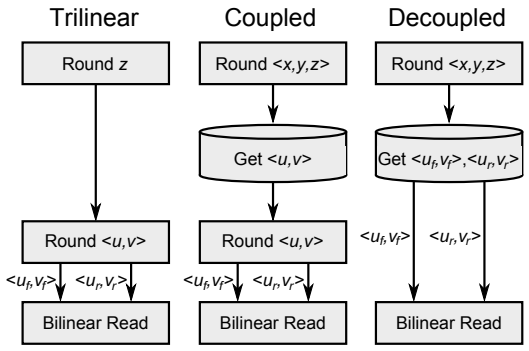


Figure 2: Pipelines of trilinear filtering, our optimized combination of bilinear samples in which texel indices are coupled to the u,v coordinates, and in which texel indices are decoupled from the u,v coordinates. The cylinders denote values read from a precalculated table.

of 20 coefficients to optimize over each subdomain. Likewise, we can fit quadratics to give a total of 50 parameters per subdomain. We write the minimization over all the filter parameters within a subdomain as

$$\int_{[0,1)^3}\int_{\mathbb{R}^2}\left(H_{\alpha\beta\sigma}(x,y)-\sum_{i=1}^{2}c_i b_{s_i u_i v_i}(x,y)\right)^2 dxdy\,d\alpha d\beta d\sigma. \tag{2}$$

The error function is a sixth order polynomial, but there are sufficiently few coefficients for a nonlinear minimization to be tractable, as described in the appendix.

3.1. Decoupled Texel Indices

Bilinear reads that are currently implemented in hardware have the u,v coordinates perform a double duty. The coordinates select both the texel indices u_f,v_f and specify the texel weights, where the u_r,v_r parameters that determine texel weights are in the range $[0,1)$. A small change in the texture-sampling hardware allows a second bilinear-sampling strategy. If we decouple the texel indices from the u,v coordinates, there is more freedom in the choice of texel coefficients. By allowing the u_r,v_r parameters to have values outside of the range $[0,1)$, the coefficients of texels can be evaluated through Equation 1 to have values outside of $[0,1)$, which can reduce approximation error for filters with negative lobes. The texel indices u_f,v_f,s are included in a discrete optimization and are stored in a small table on the GPU, with each index requiring only a few bits.

Although current GPUs couple the texels that are read to the calculation of bilinear coefficients, it would be relatively simple to modify the hardware to decouple the calculations. This decoupling should not degrade performance, and it may even result in a simpler pipeline than a standard bilinear texture read. A block diagram of the trilinear pipeline compared to our optimization for standard bilinear sampling and decoupled bilinear sampling is shown in Figure 2. The cylinders in the diagram represent a lookup into a precomputed table of values. By removing the u,v rounding stage of the pipeline, we can forward all relevant parameters to the stage that evaluates bilinear combinations of texels. Decoupled sampling can be implemented in a GLSL shader by reading individual texels and evaluating their bilinear combination, but it is not as efficient as hardware-acceleration.

	Trilinear	CCTF	Coupled	Decoupled
Lánczos 2	0.2558	0.0645	0.0876	0.0787
Tent	0.1510	0.0357	0.0412	0.0407
Gaussian	0.0705	0.0122	0.0145	0.0145

Table 1: Approximation errors of trilinear interpolation, CCTF, our method with standard bilinear interpolation in which texel indices are coupled to the u,v coordinates, and a modified bilinear sampling in which texel indices are decoupled from the u,v coordinates.

More freedom in choosing texel coefficients decreases approximation error and uses the same texture memory bandwidth. The methods reported in Table 1 all read eight texels per sample, but the choices of texels and coefficients are different between the methods. CCTF has the lowest error because it has the fewest constraints. Likewise, the additional freedom afforded by decoupling texel indices u_f,v_f from bilinear parameters u_r,v_r results in a lower error compared to standard bilinear interpolation. This freedom is especially important for the complicated Lánczos 2 filter, which oscillates and contains negative lobes, whereas the positive tent and Gaussian filters have little to no decrease in approximation error.

The texels that are read by our decoupled bilinear interpolant are now fixed, but our optimization can choose the best sets of texels for each subdomain. This discrete choice results in an optimization akin to the cardinality-constrained optimization in CCTF. We found no useful heuristic for the order in which sets of texels should be checked, but we have far fewer combinations to check than in CCTF. There are $5^2+8^2+10^2=189$ basis functions available to CCTF, from which it chooses 8, giving 34 trillion combinations. We use bilinear samples, so we cover the same set of 189 basis functions with $4^2+7^2+9^2=146$ sample indices, from which choosing two indices results in only 10,585 combinations to check. Although more than the 6 combinations in our coupled sampling, 10,585 combinations are few enough that we check all possible combinations of indices in our optimization.

4. Results

Our method can be tuned by selecting the resolution at which we discretize the domain. We use the same discretization of the texel domain as in CCTF, so we can directly compare the

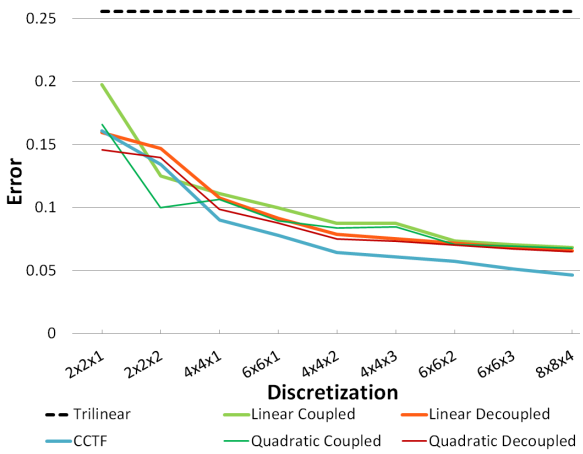


Figure 3: Errors from approximating a Lánczos 2 filter when the domain is cut into different numbers of pieces. Our coupled and decoupled bilinear samplings are compared to trilinear interpolation and CCTF. Errors of our method are shown for both linear and quadratic fitted coefficients. Eight texels are read for all methods.

two methods at different levels of discretization. We show the error of approximating a Lánczos 2 filter in Figure 3, in which our method, trilinear interpolation, and CCTF all read eight texels per sample. Trilinear interpolation performs poorly compared to the other methods, and our method has a slightly higher error than CCTF because we trade approximation error for increased cache coherence and speed. An unexpected result is that we have lower error than CCTF for discretization at very low resolution (2×2×2). In that case, rounding of u,v coordinates by the bilinear sampler allows us to read from different sets of texels within the same subdomain, whereas CCTF must use the same set of texels throughout the subdomain. Our decoupled sampler also has higher error than CCTF, but it has lower error than the coupled sampler for most discretizations.

As the resolution of the discretization or polynomial order increase, the size of the coefficient table increases, which should be kept small to fit in local memory. Figure 3 shows that the added complexity of quadratic polynomials is not worthwhile. Quadratics take 2.5 times more memory and, even in the best case of 2×2×2 discretization (10 coefficients in 2 unique subdomains), linear polynomials in a 4×4×2 discretization have a lower error and take only a little more space (4 coefficients in 6 unique subdomains). Quadratic polynomials for coefficients also take more arithmetic operations to evaluate than linear polynomials.

Subdividing the domain into 4×4×2 pieces and fitting linear polynomials provides the best compromise between table size and accuracy. Reflective symmetries mean that we only need to store the coefficients of six subdomains. It is critical, especially for hardware implementation, that the lookup table is as a small as possible. If sampled texture colors need only 8 bits of precision, it is sufficient to store 16 bits per coefficient. Three possible scales for each bilinear sample take an additional 2 bits per sample. Lookup tables can therefore be stored in $6\times(5\times4\times16+2\times2)/8 = 243$ bytes. In contrast, CCTF requires 4 coefficients for each texel as well as 4, 4, and 2 bits respectively for the u,v, and s offsets, taking a total of $6\times8\times(4\times16+2\times4+2)/8 = 444$ bytes. Our method uses almost half of the memory of CCTF for the same discretization.

	512^2 100×	256^2 400×	128^2 1600×	64^2 6400×
Trilinear: 1 native	277	512	594	650
Trilinear: 2 bilinear	244	320	362	414
Trilinear: 8 single	178	202	210	221
CCTF	54	54	49	45
Our method	100	103	94	87

Table 2: Frames drawn per second when rendering Figure 5 with a tent filter using GLSL pixel shaders. Times are shown for CCTF, our method, and trilinear interpolation using native hardware, two bilinear samples, and eight single texels. Screen resolution and redraws per frame are shown above.

Our test setup to measure frame-rates in Table 2 uses an NVIDIA GeForce GTX 680 to render a single large quad viewed at a 45° angle with a 90° field of view. With these parameters, the top edge of the quad disappears at the horizon, as shown in Figure 5. We tested the performance characteristics of our method, CCTF, and three methods for evaluating trilinear samples: native hardware, combining two bilinear samples, and combining eight single-texel reads. We render into a square window and sample a 512^2 texture such that the texture repeats twice at the bottom edge of the screen. To ensure that only the performance of the fragment shader is measured, we render the scene $100\times\frac{512^2}{W^2}$ times, where W is the width of the window and a constant number of samples are calculated for all window sizes.

Performance measurements show that bilinear samples are significantly faster than the scattered reads used in CCTF. We found that the relative speed of the sampling methods depends on the scene and how well texture reads are cached. As the screen resolution decreases, lower mipmap resolutions are sampled, thereby improving the cache performance of trilinear filtering. At the top of the table, we compare the speed of native trilinear interpolation with a trilinear interpolant constructed from two bilinear samples and from eight reads of single texels. This test shows that a shader has significant overhead when texels are read individually, and therefore bilinear primitives should be used when possible. The bilinear construction of a trilinear sample also provides an upper bound for the speed our method can achieve.

Our method is consistently twice as fast as CCTF, demon-

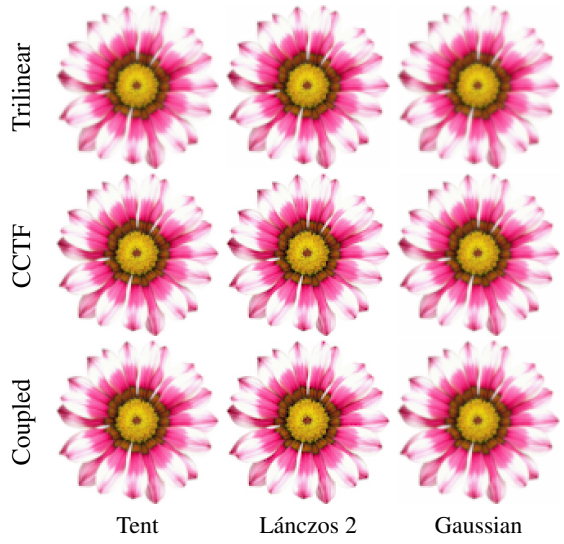


Figure 4: 2D images resampled at 100^2 pixels are approximated by different filters. From left to right, images are sampled using tent, Lánczos 2, and Gaussian filters. From top to bottom, numerical approximations of the filters are calculated by trilinear interpolation, CCTF, and our method using coupled bilinear samples.

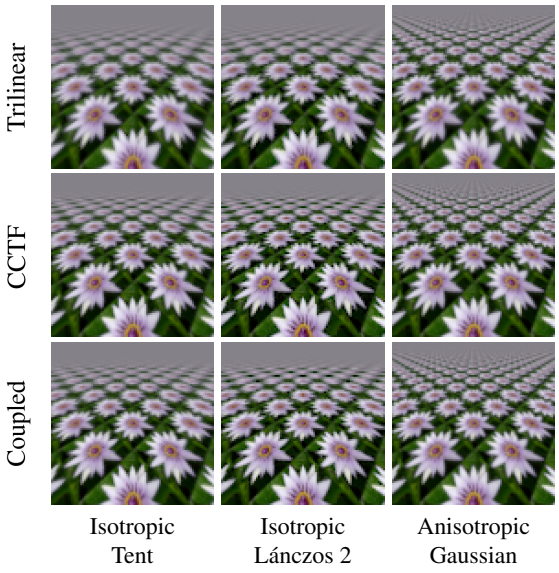


Figure 5: In each row, the effects of approximating filters using trilinear interpolation, CCTF, and our method with coupled bilinear samples are shown for an image displayed on an infinite plane. The first two columns use isotropic samples, and the third column combines several isotropic samples to approximate an anisotropic filter for each sample.

strating that the higher cache coherence and hardware support of bilinear sampling is beneficial. Although we store coefficients as a uniform array in GLSL, we suspect that dependencies between the coefficients and texture reads degrade the performance of our method. As a simple test, we removed this dependence by using constant parameters and found that our method ran twice as fast, matching the speed of *Trilinear: 8 single* in Table 2.

Our method also significantly reduces the error compared to trilinear interpolation for different filters, as shown in Table 1. Filters with simple shapes, such as tent and Gaussian filters, tend to have the lowest error relative to trilinear interpolation, but our method has several times lower error even for the complex Lánczos 2 filter. The direct test of how well our approximation performs compared to other isotropic filter approximations is to scale an image uniformly. Figure 4 shows a matrix of results from uniform scaling of a picture of a flower using isotropic tent, Lánczos 2, and Gaussian filters approximated by trilinear interpolation, CCTF, and our method with coupled bilinear samples. The flower has high-contrast details that are aligned in all radial directions. Although CCTF is slightly sharper than our method, we maintain more textural detail than trilinear interpolation.

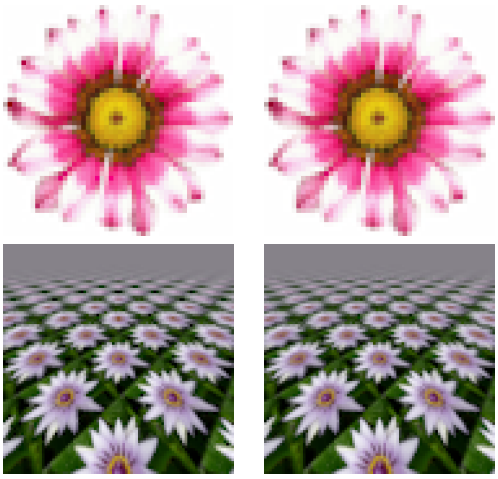
Texture filtering is also often used in highly anisotropic 3D scenes. Images produced by trilinear interpolation, CCTF, and our method are compared in Figure 5. Each column corresponds to approximations of isotropic tent, isotropic Lánczos 2, and anisotropic Gaussian filters, respectively. We approximate the anisotropic Gaussian from isotropic samples using the Feline algorithm [MPFJ99]. The quality of our method is similar to that of CCTF despite having less freedom in choosing locations and coefficients of texels. The results obtained with a Lánczos 2 filter appear sharper than with a tent filter, but trilinear interpolation blurs the image. Thus, other methods resolve more detail when approximating a tent filter than trilinear interpolation resolves when using a Lánczos 2 filter. Anisotropic filtering shows more detail in distant objects for all approximation methods, but trilinear filtering renders the blurriest image at all distances. Figure 6 demonstrates that we can produce a sharper image of a 3D scene by using our improved texture filter rather than trilinear interpolation. Most of the detail in the scene is contained in textures on the relatively flat geometry of the flags and walls, not in the overall geometry of the scene.

We show a comparison between coupled and decoupled sampling in Figure 7 for 2D and 3D scenes. In the 2D images, some pixels are noticeably darker with coupled sampling than with decoupled sampling. These pixels are easiest to spot in the pink part of the flower at the 6-7 o'clock position. In the 3D images, the difference between coupled and decoupled sampling is most easily seen between the flowers where they begin to become blurry on the horizon.

(a) Trilinear Lánczos 2

(b) Coupled Lánczos 2

Figure 6: Texture sampling in the 3D Sponza scene using (a) trilinear interpolation and (b) our sampling method with coupled bilinear samples. It is easiest to see the difference in sharpness in the emblem on the blue flag at the bottom left of the images.



Coupled Lánczos 2 Decoupled Lánczos 2

Figure 7: A comparison between coupled and decoupled bilinear sampling used to approximate a Lánczos 2 filter. The coupled sampling creates artifacts of isolated darker pixels.

5. Conclusion

It is possible to improve the quality of texture sampling over trilinear interpolation significantly while still maintaining the same level of cache coherence when samples are constructed from bilinear texture reads. Our method is slower than trilinear filtering when run on current hardware, but our method is twice as fast as CCTF. We designed the simple scene in Figure 5 so that we could differentiate between the speeds of texture filtering for various methods. In practice, scenes will typically be much more complex, and texture filtering may not be a performance bottleneck. In such cases, there would be only a small performance penalty for the improved filtering of our method.

Decoupling the texels that are used in the bilinear sample from the bilinear weights can improve quality even further if such changes are made in GPUs. We expect that hardware support will increase the speed of our method to match that of trilinear interpolation. In contrast, implementing CCTF in hardware would not provide as much of a performance benefit as our method does, because the speed of CCTF is limited by poor cache coherence, whereas we have the same coherence as trilinear interpolation.

Knowing the exact cache structure of texture memory on a video card may allow us to improve the performance of texture filtering even more. It makes sense to optimize filter approximation when cost is measured by the number of cache lines that are read. Not knowing the cache structure can create high cost in reading a texel while providing little benefit. Currently, a bilinear sample may require up to four cache lines, and a whole cache line may be read for a single texel with a coefficient that is nearly zero. Conversely, texels are often unused although they are prefetched by the caching subsystem and could decrease approximation error at little additional cost. It may also be possible to improve performance by factoring the cost of decompressing block-compressed texture formats such at DXTC.

Acknowledgements

Acknowledgements will be added after the anonymous review process.

References

[BA83] BURT P., ADELSON E.: The laplacian pyramid as a compact image code. *IEEE Transactions on Communications 31*, 4 (1983), 532–540. 2

[CDK04] CHEN B., DACHILLE F., KAUFMAN A. E.: Footprint area sampled texturing. *IEEE Transactions on Visualization and Computer Graphics 10*, 2 (2004), 230–240. 2

[Cro84] CROW F. C.: Summed-area tables for texture mapping. In *Proceedings of ACM SIGGRAPH* (1984), pp. 207–212. 2

[CS97] CANT R., SHRUBSOLE P.: Texture potential mapping: A way to provide antialiased texture without blurring. In *Visualization and Modelling* (1997), pp. 223–240. 2

[CS00] CANT R., SHRUBSOLE P.: Texture potential mip mapping, a new high-quality texture antialiasing algorithm. *ACM Transactions on Graphics 19*, 3 (2000), 164–184. 2

[FFB88] FOURNIER A., FIUME E., BUILDING S. F.: Constant-time filtering with space-variant kernels. In *Proceedings of ACM SIGGRAPH* (1988), pp. 229–238. 2

[GH86] GREENE N., HECKBERT P.: Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications 6*, 6 (1986), 21–27. 2

[Gla86] GLASSNER A.: Adaptive precision in texture mapping. In *Proceedings of ACM SIGGRAPH* (1986), pp. 297–306. 2

[Hec89] HECKBERT P.: *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, University of California, Berkeley, 1989. 2

[HS99] HÜTTNER T., STRASSER W.: Fast footprint mipmapping. In *Proceedings of the ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (1999), pp. 35–44. 2

[MP11] MAVRIDIS P., PAPAIOANNOU G.: High quality elliptical texture filtering on gpu. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2011), pp. 23–30. 2

[MPFJ99] MCCORMACK J., PERRY R. N., FARKAS K. I., JOUPPI N. P.: Feline: Fast elliptical lines for anisotropic texture mapping. In *Proceedings of ACM SIGGRAPH* (1999), pp. 243–250. 1, 2, 6

[MS13] MANSON J., SCHAEFER S.: Cardinality-constrained texture filtering. In *Proceedings of ACM SIGGRAPH* (2013), pp. 140:1–140:8. 1, 2

[SH05] SIGG C., HADWIGER M.: Fast third-order texture filtering. In *GPU Gems 2*, Pharr M., (Ed.). Addison-Wesley, 2005, pp. 313–329. 2

[SKS96] SCHILLING A., KNITTEL G., STRASSER W.: Texram: a smart memory for texturing. *Computer Graphics and Applications, IEEE 16*, 3 (1996), 32–41. 2

[Wil83] WILLIAMS L.: Pyramidal parametrics. In *Proceedings of ACM SIGGRAPH* (1983), pp. 1–11. 2

[ZLW06] ZHOUCHEN LIN L. W., WAN L.: First order approximation for texture filtering. In *Pacific Graphics Poster* (2006). 2

Appendix

There are two equivalent interpretations of Equation 2. One interpretation is that we sample the mipmap in two stages. We first evaluate two bilinear samples, then take a linear combination of those samples. An alternate interpretation is that the two sets of bilinear weights from Equation 1 define an array of eight texel coefficients. This second interpretation allows us to write the minimization in matrix form as though it were a linear problem as

$$\underset{X}{\operatorname{argmin}} \quad (AX-B)^2, \tag{3}$$

where the only difference from a linear problem is that the vector X is a function of the parameters $c, u_1, u_2, v_1, v_2, s_1, s_2$. This reformulation allows us to use the Levenberg-Marquardt algorithm to find a minimal solution.

We have a closed-form expression for the integral over x, y, but numerically evaluate the integral over α, β, σ by summing the errors of discrete parameter values. In the framework of Levenberg-Marquardt, the sum of the error from numerical integration is calculated by concatenating error vectors from each of the parameter values. The difficulty is that although Equations 2 and 3 are both quadratic, integrals over products of functions in Equation 2 do not directly give an error vector. Rather, they give a single value that is the summed square of an error vector. Thus, we must decompose the quadratic given in terms of $\hat{A}, \hat{B}, \hat{C}$ into a sum of squares

$$X^T \hat{A} X + \hat{B} X + \hat{C} = \sum_i (A_i X - B_i)^2,$$

where

$$X^T \hat{A} X = \int_{\mathbb{R}^2} \left(\sum_{i=1}^{2} c_i b_{s_i u_i v_i}(x,y) \right)^2 dx\, dy$$

$$\hat{B} X = -2 \int_{\mathbb{R}^2} H_{\alpha\beta\sigma}(x,y) \sum_{i=1}^{2} c_i b_{s_i u_i v_i}(x,y)\, dx\, dy$$

$$\hat{C} = \int_{\mathbb{R}^2} \left(H_{\alpha\beta\sigma}(x,y) \right)^2 dx\, dy.$$

The hardware bilinear sampler selects different sets of texels u_f, v_f depending on the u, v parameters. Therefore, X, $\hat{A}$ and $\hat{B}$ must contain entries for each texel that can be read. Only eight of the entries in X are non-zero for a particular u, v, and the zero entries will not contribute to the sum of the error vector, but the Levenberg-Marquardt algorithm requires the error vector to change continuously. Although limiting X to use eight coefficients by changing the indexing of X would result in a continuous change in the total error, the error vector itself would change discontinuously. In practice, we need to bound the size of X, so we allow only u, v coordinates in the range $[-1, 2]$ by adding a large error when u, v approach or leave the valid range.