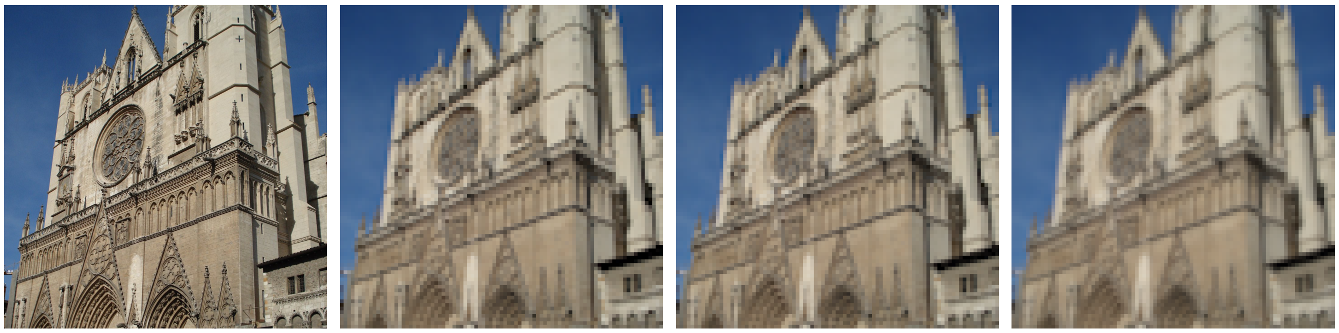
Cardinality-Constrained Texture Filtering

Josiah Manson
Texas A&M University

Scott Schaefer
Texas A&M University



(a) *Input* **(b)** *Exact* **(c)** *8 Texels* **(d)** *Trilinear*

Figure 1: *We show a comparison of Lánczos 2 filter approximations showing (a) the* 1024^2 *input image downsampled to a resolution of* 89^2 *pixels using (b) an exact Lánczos filter, (c) an eight texel approximation using our method, and (d) trilinear interpolation applied to a Lánczos filtered mipmap. Our approximation produces an image that is nearly the same as the exact filtered image while using the same number of texels as trilinear interpolation.*

Abstract

We present a method to create high-quality sampling filters by combining a prescribed number of texels from several resolutions in a mipmap. Our technique provides fine control over the number of texels we read per texture sample so that we can scale quality to match a memory bandwidth budget. Our method also has a fixed cost regardless of the filter we approximate, which makes it feasible to approximate higher-quality filters such as a Lánczos 2 filter in real-time rendering. To find the best set of texels to represent a given sampling filter and what weights to assign those texels, we perform a cardinality-constrained least-squares optimization of the most likely candidate solutions and encode the results of the optimization in a small table that is easily stored on the GPU. We present results that show we accurately reproduce filters using few texel reads and that both quality and speed scale smoothly with available bandwidth. When using four or more texels per sample, our image quality exceeds that of trilinear interpolation.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing

Keywords: texture mapping, image filtering, image resampling, filter approximation, image pyramid, mipmap

Links: DL PDF WEB

ACM Reference Format
Manson, J., Schaefer, S. 2013. Cardinality-Constrained Texture Filtering. ACM Trans. Graph. 32, 4, Article 140 (July 2013), 8 pages. DOI = 10.1145/2461912.2461963 http://doi.acm.org/10.1145/2461912.2461963.

1 Introduction

Artists often apply images, called textures, to the surface of three-dimensional models to add visual interest. However, we must take care when displaying images on a model, because there is not a one-to-one correspondence between the texels (texture elements) and pixels of the display. When a model is in the distance and several texels correspond to each pixel, poor sampling can cause false patterns, called aliasing, to appear. If we interpret drawing textures as sampling a two-dimensional signal, Shannon's sampling theorem [Shannon 1949] implies that we must use a low-pass filter to remove high-frequency data from the image prior to sampling.

There are a variety of low-pass filters, where each filter has its own set of tradeoffs. Some filters remove aliasing at the cost of overblurring the image, while others blur less but allow more aliasing. Filters that are effective at removing aliasing without overblurring sum over a greater number of texels, which makes them expensive to compute. As an extreme example, the sinc filter removes all high frequencies and no low frequencies, but sums over an infinite number of texels. Directly adding all samples that fall under the filter support becomes impractical for distant objects, because we must sum over a number of texels proportional to the squared distance.

Rendering algorithms typically use image pyramids called mipmaps [Williams 1983] to accelerate image filtering. Mipmaps consist of precalculated images downsampled at power-of-two resolutions and can be used to compute filters in constant time, regardless of the scaling factor. We present a method that combines texels in a mipmap to reproduce the results of low-pass filters while only reading a few texels per sample. Our insight is two-fold. Rather than interpolating colors between single points, so that colors are exact at those points but poor everywhere else, we find weights that give good results over all possible sample points. Our second insight is that we can combine texels from any mipmap resolution. Given a sampling filter, the prefilter used to construct the mipmap, and a texel budget, we can solve for which texels to use and the weights that best reproduce the sampling filter.

Memory bandwidth is often a bottleneck in graphics applications, so we attempt to use the bandwidth as efficiently as possible. Our method can also scale the number of texel reads per sample to match the available bandwidth. By carefully choosing which texels to use, we accurately reproduce image filters that are sharp and free of aliasing for all scales, translations, and rotations of an image. Furthermore, we can approximate high-quality filters such as the Lánczos 2 filter in real-time, because the size and complexity of a filter only affects preprocessing time to calculate filter coefficient tables and generate mipmaps. We show an example in Figure 1 where we approximate a Lánczos 2 filter compared to exact evaluation of the filter and trilinear interpolation of the mipmap.

When sampling a texture, we measure the distortion of each pixel into texture space. Isotropic filtering assumes that distortions scale the pixel, whereas anisotropic filtering allows pixels to stretch. When viewing three-dimensional surfaces at oblique angles, anisotropic filtering improves image quality, but reduces to isotropic filtering for perpendicular viewing directions. We focus our attention on improving the quality of isotropic filtering, and we describe how our method applies to anisotropic image filtering at the end of the paper.

2 Related Work

Most real-time rendering algorithms use mipmapping [Williams 1983; Burt and Adelson 1983] to sample textures. Mipmapping reduces aliasing by precalculating downsampled images at several resolutions with a low-pass filter, such as the box, tent, Gaussian, Lánczos [Duchon 1979], or Mitchell-Netravali [Mitchell and Netravali 1988] filters. Because sampling positions do not typically coincide with texel centers, trilinear interpolation is often used to calculate colors between texels. Mipmapping allows sampling algorithms to be independent of scale while using only 33% more memory than the input image.

There is surprisingly little literature on how to improve upon mipmapping for isotropic filtering. The attention of researchers has instead focused on how to improve anisotropic texture filtering [Crow 1984; Glassner 1986; Greene and Heckbert 1986; Heckbert 1989; Schilling et al. 1996; Cant and Shrubsole 1997; Hüttner and Straßer 1999; McCormack et al. 1999; Cant and Shrubsole 2000; Chen et al. 2004; Zhouchen Lin and Wan 2006; Mavridis and Papaioannou 2011]. Although these methods are designed to improve anisotropic filtering, some of the methods also improve isotropic filtering. Summed area tables [Crow 1984] accurately calculate axis-aligned box filters, but at the cost of significantly more memory usage. For example, 28 bits instead of 8 bits are required per color channel for a 1024^2 image to avoid loss of precision and increases storage by 250% compared to 33% for a mipmap. Elliptic weighted averaging (EWA) [Greene and Heckbert 1986] samples with a Gaussian filter, and Heckbert uses a mipmap to accelerate EWA [Heckbert 1989] by fetching between 9 and 36 texels from one resolution. Another method stores tables of texel weights for box filters when sampling from a single mipmap level [Hüttner and Straßer 1999].

In contrast, our approach combines a fixed number of prefiltered texels at different resolutions to generate a filter at arbitrary scales. Researchers have explored the idea of reproducing filters by combining texels from images at different resolutions [Burt 1981], but for the purpose of feature detection rather than fast image sampling. Wavelet theory [Mallat 1989] also combines multiresolution basis functions into arbitrary sampling filters, but building a filter from wavelets requires summing a number of basis functions that is logarithmic in the scale of the filter. In order to reproduce sampling filters with a constant number of basis functions, we use scales and translates of the filter functions as our basis.

NIL mapping [Fournier et al. 1988] computes filters through adaptive quadrature, a recursive process of refining a filter in areas of high approximation error. Because NIL mapping stops recursion once a sampling limit is reached, the filter is computed in constant time. Although NIL mapping uses multiple resolutions over the support of the filter, the color of a texture sample depends only on texels from one resolution at any point. Also, NIL mapping computes texel weights directly from the filter function rather than choosing values to minimize approximation error. The resulting algorithm is somewhat slow, difficult to implement on a GPU, and has higher error than necessary.

An alternative approach for constant time filtering is to optimize for the best set of basis functions to reconstruct a filter [Gotsman 1994] rather than optimizing for the coefficients of a fixed basis. In this paper, the authors optimize a set of basis functions to represent rotations and non-uniform scales of a Gaussian filter around a point. The authors do not include translations of the filter in their optimization and do not discuss how they could use a mipmap-like hierarchy of resolutions, which limits the scalability of the method.

3 Multi-resolution Sampling

We wish to sample an image $\hat{I}$ defined over the $[0,1]^2$ domain using a low-pass filter h, such as a box, tent, Gaussian, or Lánczos filter. To compute the color of a pixel with scale $\hat{s}$ and translation $\hat{t} = (\hat{t}_0, \hat{t}_1)$ relative to $\hat{I}$, we transform h to match the position and scale of the sample by $h_{\hat{s},\hat{t}}(x) = 2^{\hat{s}} h(2^{\hat{s}}(x - \hat{t}))$ so that the color of the sample $v_{\hat{s},\hat{t}}$ integrated over points $x = (x_0, x_1)$ is

$$v_{\hat{s},\hat{t}} = \iint_{\mathbb{R}^2} \hat{I}(x) h_{\hat{s},\hat{t}}(x)\, dx. \tag{1}$$

Directly computing $v_{\hat{s},\hat{t}}$ is costly when the support of $h_{\hat{s},\hat{t}}$ is large, so we need to approximate this integral for real-time rendering. In particular, we want the property that the time taken to sample an image is independent of the position and scale of $h_{\hat{s},\hat{t}}$ so that we always fetch a constant number of texels. For scale independence, we store downsampled images in a mipmap image stack I, and compute the texels $I_{\mathcal{S},\mathcal{T}} = v_{\hat{s},\hat{t}}$ using the same filter $h_{\hat{s},\hat{t}}$ that we wish to sample with. Although we could generate the texels $I_{\mathcal{S},\mathcal{T}}$ with a filter other than $h_{\hat{s},\hat{t}}$, using $h_{\hat{s},\hat{t}}$ ensures that we can exactly compute $v_{\hat{s},\hat{t}}$ at texel samples. The set of coordinates E of mipmap samples are the standard cell-centered positions, which have integer coordinates $\mathcal{S}, \mathcal{T}$ that we relate to positions in the mip-volume by $\hat{s} = \mathcal{S}$ and $\hat{t} = 2^{-\mathcal{S}}(\mathcal{T}_0 + \frac{1}{2}, \mathcal{T}_1 + \frac{1}{2})$. We visualize the $\hat{s}, \hat{t}$ coordinate system in Figure 2, with the positions of texels shown as red dots and a hypothetical sampling query for $h_{\hat{s},\hat{t}}$ shown in blue.

We perform an optimization to approximate $v_{\hat{s},\hat{t}}$ by fetching a subset of texels $e \subset E$. Our optimization for the coefficients c_i of the texels e_i has the cardinality constraint is that $|e| = n$, where n is a fixed sample budget. Solving a cardinality-constrained optimization is proven to be NP-hard [Welch 1982], because we must test all possible solutions to find the minimal solution. In higher-dimensional problems, the number of basis functions, and therefore the number of combinations of basis functions, becomes too large to check exhaustively. However, we show how we can efficiently approximate this solution in Section 3.2.

Another constraint is that our filter should reproduce constant functions (i.e. have constant precision) to prevent distracting patterns from appearing in constant and nearly constant regions of an image. A filter has constant precision when $\sum c_i = 1$. We demonstrate the

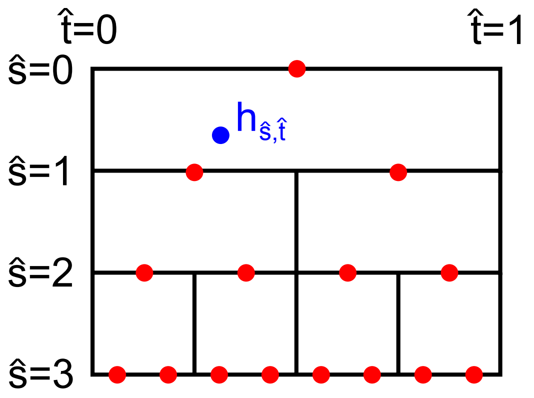


Figure 2: *A two-dimensional depiction of the reference coordinate system. Texels in the mipmap are shown as red dots, and the coordinate of a possible filter $h_{\hat{s},\hat{t}}$ is shown in blue.*

importance of constant precision in Figure 3. Compared to the image downsampled using an exact Lánczos 2 filter, our approximation without constant precision does not reproduce the brightness of the input image. Lack of constant precision also introduces a pattern in the sky where the color should be nearly constant.

We can write the constrained optimization for the best set of coefficients c and texels e to approximate $v_{\hat{s},\hat{t}}$ as

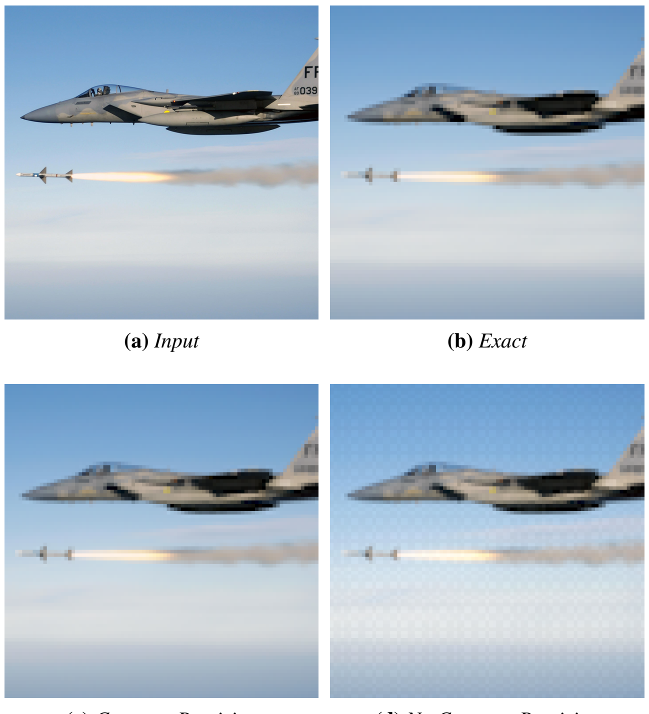
$$\underset{\substack{c,e\subset E \\ \sum c_i=1,|e|=n}}{\operatorname{argmin}} \iint_{\mathbb{R}^2} \left(\hat{I}(x)h_{\hat{s},\hat{t}}(x) - \sum_{i=1}^{n} \hat{I}(x)h_{e_i}(x)c_i\right)^2 dx. \quad (2)$$

This optimization depends on the values of $\hat{I}$, but we wish to precalculate coefficients that are independent of the input image so that we can quickly compute the filter later. Notice that $\hat{I}$ weights the importance of reproducing the shape of the filter at point x. To give the best result when $\hat{I}$ is unknown, we give all x equal weight, which simplifies the minimization to

$$\underset{\substack{c,e\subset E \\ \sum c_i=1,|e|=n}}{\operatorname{argmin}} \iint_{\mathbb{R}^2} \left(h_{\hat{s},\hat{t}}(x) - \sum_{i=1}^{n} h_{e_i}(x)c_i\right)^2 dx. \quad (3)$$

To understand properties of two-dimensional filters, we analyze the optimization in Equation 3 for one-dimensional filters, which are easier to visualize. For a two-dimensional image, trilinear interpolation interpolates over $\hat{t}_0$, $\hat{t}_1$, and $\hat{s}$ to approximate arbitrary filters, while the equivalent one-dimensional process interpolates over $\hat{t}_0$, and $\hat{s}$.

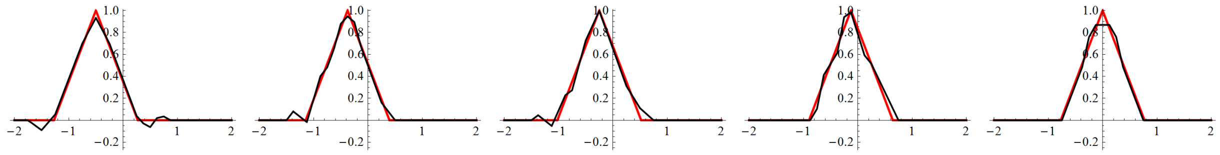
We illustrate how we can accurately approximate filters in Figure 4. We approximate translations of the one-dimensional tent filter shown in red using weighted combinations of the black basis functions. We show h halfway between mipmap levels at translates of $\frac{0}{8}$, $\frac{1}{8}$, $\frac{2}{8}$, $\frac{3}{8}$, and $\frac{4}{8}$ texels. Both our method and linear interpolation over $\hat{t}_0$ and $\hat{s}$ use four basis functions. We show the result of our method in (a) and show the result of linear interpolation in (c), and one can see that our method reproduces the filter well compared to linear interpolation. We show the basis functions times the coefficients used to approximate the filter beneath the approximation in (b) and (d), which shows that our method maintains image sharpness by sampling from higher-resolution basis functions to shape the filter. The different translations in (b) also show how the optimal strategy for approximating a filter depends strongly on the parameters of $h_{\hat{s},\hat{t}}$. On the far left, the best solution is to subtract the sides from a basis function that is wider than $h_{\frac{1}{2},0}$, whereas on the right, the best solution is to add high-resolution basis functions to approximate $h_{\frac{1}{2},\frac{1}{2}}$. For intermediate translations, a combination of both approaches is best.



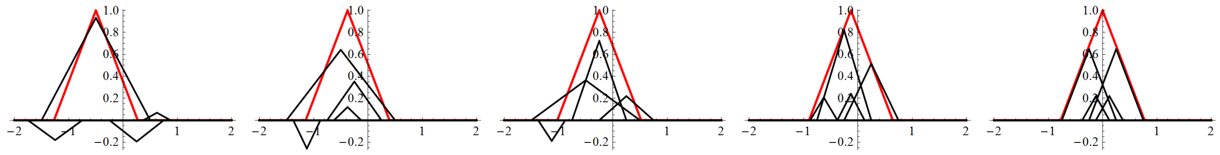
(a) *Input* **(b)** *Exact* **(c)** *Constant Precision* **(d)** *No Constant Precision*

Figure 3: *The difference between enforcing constant precision when downsampling an image and not enforcing constant precision with Lánczos 2 filtering.*

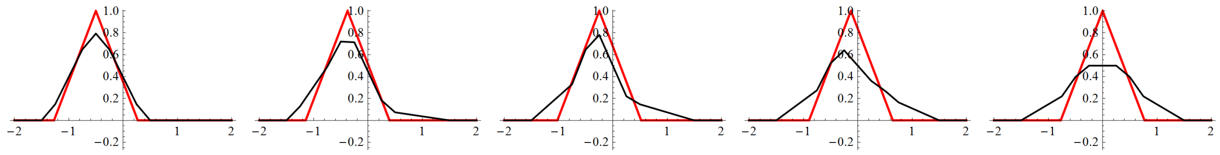
In Figure 5, we show the approximation error of our method in blue when h is a one-dimensional tent filter, compared to the error of bilinear interpolation in black. We evaluated the errors in the graph for translations over the width of a texel $[0, 1]$ at an integer mipmap resolution and plot the error for unique subsets of four texels in green. Not one of these subsets has the lowest error over the entire domain, so finding the optimal solution shown in blue requires that we minimize the error for all of the possible subsets at each point and choose the subset with the least error. In Section 3.1 we show that we can minimize the error over regions instead of for every point. Although the problem is NP-hard, we can exhaustively check all possible combinations for this low-dimensional problem, but have to use a heuristic method described in Section 3.2 for higher dimensions. Linear interpolation and the optimal solution both have zero error at $\frac{1}{2}$, which is when $h_{\hat{s},\hat{t}}$ aligns with a texel center and means that both methods interpolate the texel values. An interesting property of our method is that because h is a tent function in this example, we also have zero error at $\frac{1}{4}$ and $\frac{3}{4}$. This is because tent functions have the recurrence relation that a tent function can be built from three tent functions of twice the resolution. Several of the sets shown in green have zero error at $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$, because fewer than the maximum ($n = 4$) texels are required to give an solution with zero error.



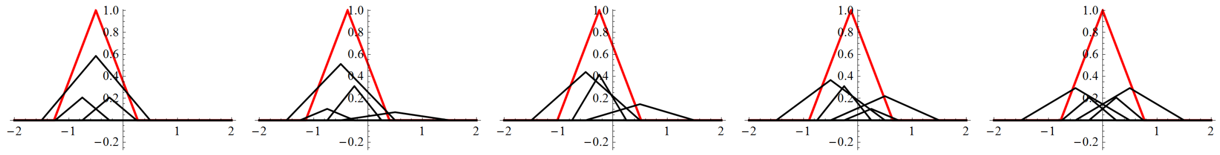
(a) *Optimal approximation*



(b) *Optimal approximation functions used*



(c) *Bilinear interpolation*



(d) *Bilinear interpolation functions used*

Figure 4: *A one-dimensional example of how our optimization improves over bilinear and trilinear interpolation using the same number of basis functions. The filter we approximate is shown in red and the four basis functions or their sums are shown in black. Filters are sampled halfway between integer mipmap resolutions at translates of 0/8, 1/8, 2/8, 3/8, 4/8 of a texel.*

3.1 Polynomial Fitting

For sampling to be practical in a real-time system, it is not possible to use the optimal solution for all possible sampling filters because the best set of texels to use depends strongly on the parameters $\hat{s}$ and $\hat{t}$ of the filter $h_{\hat{s},\hat{t}}$. From Figure 5, one can see that there is no single best subset to use, because the green lines cross along at the bottom of the graph. After dividing the domain into a few pieces, we can choose a subset to fit each piece accurately. Also, texel coefficients have no closed-form solution, and we describe how to fit polynomials to the coefficients of a set of texels in this section. We show the error when fitting linear coefficients over four pieces in Figure 5 as alternating red and orange curves.

We can parameterize cells in Figure 2 by $(\mathbf{t}_0, \mathbf{t}_1, \mathbf{s}) \in [0,1]^3$ so that $\mathbf{s} = \hat{s} - \mathcal{S}$ and $\mathbf{t} = 2^{\mathcal{S}}\hat{t} - \mathcal{T}$, where the integer coordinates of a cell are given by $\mathcal{S} = \lfloor \hat{s} \rfloor$ and $\mathcal{T} = \lfloor 2^{\mathcal{S}}\hat{t} \rfloor$. We cut this domain into $\mathcal{J} \times \mathcal{J} \times \mathcal{K}$ smaller subdomains D that are parameterized by $s = \mathcal{K}\mathbf{s} - \lfloor \mathcal{K}\mathbf{s} \rfloor$ and $t = \mathcal{J}\mathbf{t} - \lfloor \mathcal{J}\mathbf{t} \rfloor$. We fit sets of polynomial coefficients c_{ij} for the power basis $p(s,t)$ to the texel weights for each of the subdomains, where $j = 1 \ldots m$ indexes the power basis function. We have tested using a linear basis $p(s,t) = (1, t_0, t_1, s)$, and a quadratic basis $p(s,t) = (1, t_0, t_1, s, t_0^2, t_1^2, s^2, t_0t_1, t_0s, t_1s)$. Holding the set of texels $e \subset E$ fixed and defining c_i by its polynomial expansion $c_i(s,t) = \sum_j p_j(s,t)c_{ij}$ gives the optimization

$$\underset{\substack{c_{ij} \\ \sum c_i(s,t)=1}}{\operatorname{argmin}} \iint_{\mathbb{R}^2} \iiint_D \left(h_{s,t}(x) - \sum_{i=1}^{n} h_{e_i}(x) c_i(s,t) \right)^2 dt\, ds\, dx. \tag{4}$$

The constant precision constraint creates a linear dependence between coefficients, which allows us to replace one of the coefficients and simplify the minimization. Written in the power basis,

$$\begin{pmatrix} c_{11} \\ c_{12} \\ \vdots \\ c_{1m} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \sum_{i=2}^{m} \begin{pmatrix} c_{i1} \\ c_{i2} \\ \vdots \\ c_{im} \end{pmatrix}. \tag{5}$$

Equation 4 is quadratic in c_{ij}, which we solve as a linear system. Our optimization leaves freedom to choose how many texels to use, how to subdivide the domain, and what order polynomial to use. Each option provides a tradeoff in terms of speed, memory usage, and quality. We discuss the tradeoffs and our choices in Section 4.

3.2 Combinatorics and Heuristics

Solving linear systems to find texel weights is reasonably fast, but there are many possible sets of n texels. For each subdomain, we need to find the set of texels $e \subset E$ that has the lowest error when evaluating Equation 4. If we choose n texels out of a pool of $N = |E|$ possible texels, then we need to check the error of $\frac{N!}{(N-n)!n!}$ combinations of texels. Clearly, we need to limit N as much as possible for the problem to be tractable. Our first observation is that we can exclude texels that are not in the support of $h_{s,t}$. Although texels outside of the support of $h_{s,t}$ could theoretically be beneficial, the fact that we use few texels makes it unlikely that they would reduce the approximation error. Our second observation is that we primarily use low-resolution texels to approximate the filter when n is small. We have found that we only use the texels from

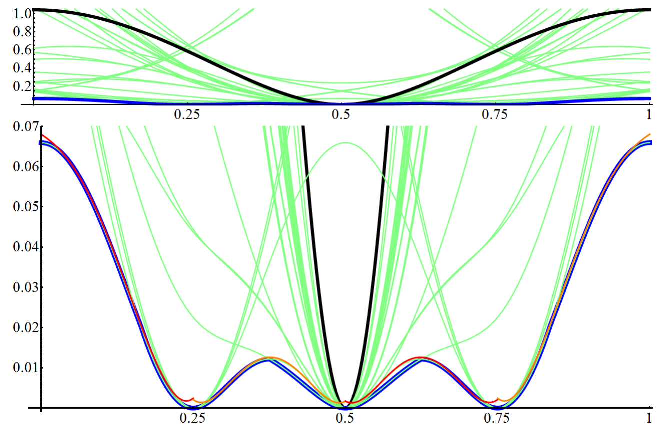


Figure 5: *We show the error of bilinear interpolation in black, different subsets of texels in light green, optimal error in blue, and error of our piecewise polynomial in alternating red and orange. We show the graph zoomed in on the bottom.*

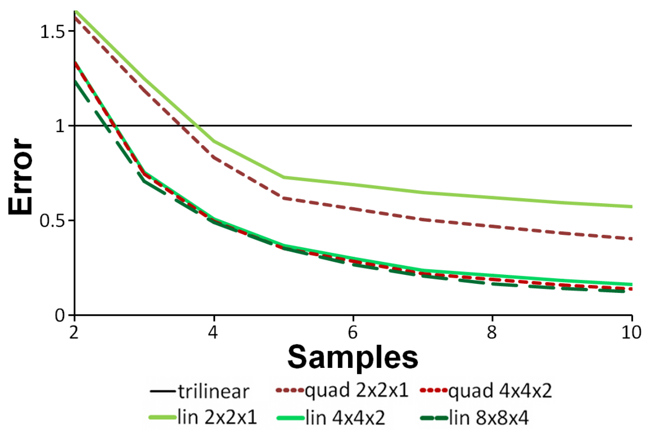


Figure 6: *We show the error of approximating a tent filter using varying numbers of texels with different optimization choices compared against trilinear interpolation. The errors are normalized so that trilinear interpolation has an error of one.*

relative mipmap levels 0, 1, and 2 for a tent filter with $n = 8$, so we exclude other resolutions from our optimization.

Even after restricting E to have fewer texels, a tent filter has 189 texels to choose from. Checking all combinations is not practical, because there are 34 trillion combinations of eight texels. We can check approximately two million combinations in a minute, so exhaustively checking all combinations would take 33 years. We therefore develop a heuristic for determining which sets are most likely to have low error. We define the error of a texel to be the minimal error of the texel by itself in Equation 4. Our heuristic is that a texel basis function that matches $h_{s,t}$ with low error is likely to be in the set of functions that approximates $h_{s,t}$ with minimal error. By extension, sets of basis functions where each function is a good approximation of $h_{s,t}$ are more likely to approximate $h_{s,t}$ well. We therefore check combinations of low-error texels before checking high-error texels.

The single-texel error defines a priority by which we order texels in a list. We try to select the best n-texel subset from among the highest priority texels before progressively widening the search space to include lower priority texels. We terminate our search once we check a desired number of combinations, and, although we can only test a small fraction of the total space for $n = 8$, we often find good solutions quickly. We checked 100 million sets for each of the six unique subdomains (using symmetry) in a $4 \times 4 \times 2$ discretization of an eight sample tent filter. In this test, we found the best set out of the sets checked after 15, 513, 518, 12991, 35960, 534979 trials, and found several other sets with low errors prior to that. All of our best solutions were within the first 1% of the subsets that we checked, which indicates that our heuristic works well and that we find nearly optimal sets.

3.3 Implementation

To implement sampling, we use two tables: an index table and a coefficient table. The index table stores the relative offsets of the n texel indices for each subdomain and the coefficient table stores the coefficients for the texel weights. Index offsets are vectors of three integers indicating the texel's coordinate $(\mathcal{T}_0, \mathcal{T}_1, \mathcal{S})$ relative to the sample. Coefficients of a linear function are four component vectors (one constant coefficient and three linear coefficients).

Sampling $v_{\hat{s},\hat{t}}$ using the filter $h_{\hat{s},\hat{t}}$ consists of the steps:

1. Find subdomain $D \in \mathbb{Z}^3$, texel index $(\mathcal{T}_0, \mathcal{T}_1, \mathcal{S}) \in \mathbb{Z}^3$, and remainder $(t_0, t_1, s) \in [0, 1]^3$.
2. Calculate the offset into the index table and the coefficient table from the subdomain index D.
3. For all n texels:
   (a) Compute the polynomial texel coefficient $c_i(s, t)$.
   (b) Add $c_i(s, t)$ times the texel color I_{e_i} into $v_{s,t}$.

A small complication is that higher-resolution mipmaps are not available for all scales of $h_{\hat{s},\hat{t}}$, so we generate additional tables for low mipmap levels. This is akin to the difference between minification and magnification in GPUs. In our case, we use three mipmap levels when $1 < \hat{s}$, but need to optimize for two mipmap levels when $0 < \hat{s} \leq 1$ and for one level when $\hat{s} \leq 0$. In practice, we do not benefit much from optimizing a single level and revert to the reconstruction filter for $\hat{I}$ when $\hat{s} \leq 0$.

We significantly reduce the number of tables that we store by taking symmetry into account. Tensor-product filters have four-fold rotational symmetry and are symmetric across the diagonal, which means that texel coefficients are uniquely defined over an eighth of the parametric space. If we subdivide the domain into $\mathcal{J} \times \mathcal{J} \times \mathcal{K}$ pieces, symmetries reduce the number of subdomains from $\mathcal{J}^2\mathcal{K}$ to $\mathcal{J}(\mathcal{J} + 2)\mathcal{K}/8$. This space optimization allows us to easily fit precomputed tables into constant memory on a GPU.

Evaluating the color of a sample consists of a table lookup and n multiply-add operations. The overhead from finding table and texel entries based on symmetry requires $3n + 3$ *if* statements. If our method was implemented in hardware, we could handle the *if* statements more efficiently than is possible in a shader by computing the relatively simple symmetry corrections in parallel and selecting the correct symmetry with a multiplexer. We could also compress the index table significantly by using three bits per index. With this in mind, we anticipate that there would be less overhead from using our method in hardware than there is in software.

4 Results

We graph the approximation error of our method compared to a directly convolved tent filter for integer numbers of samples from 2 to 10 in Figure 6 as measured by Equation 3 and normalized by

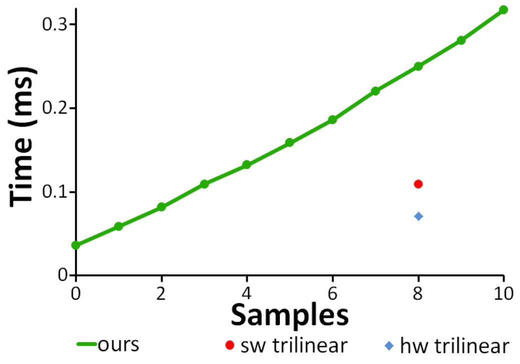


Figure 7: *We show the times to draw Figure 10 at* 512^2 *resolution using our method compared to trilinear interpolation as implemented by the hardware (HW Trilinear) and in a GPU shader (SW Trilinear). The number of texels that the GPU fetches per sample is shown by the horizontal axis.*

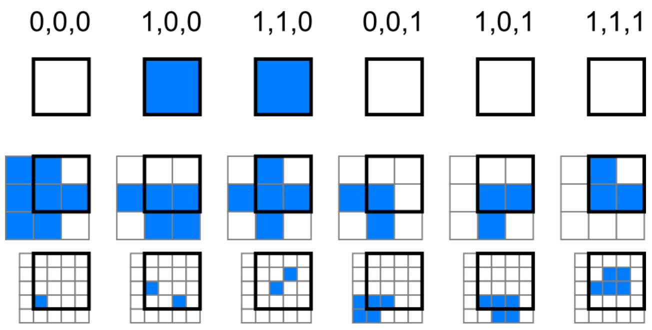


Figure 8: *We show the eight texel access pattern of a 4x4x2 discretization of a tent filter. The unit domain is outlined in black, and each column of images shows the texels used in a subdomain, where texels with nonzero coefficients are blue. There are only six subdomains because of symmetry, and the index of the subdomain is ordered (left to right, bottom to top, low to high resolution).*

the error of trilinear interpolation. The cost of our method depends on the number of subdomains and the order of the polynomials we fit, so we compare the error of: linear polynomials for e_i over $2 \times 2 \times 1$, $4 \times 4 \times 2$, and $8 \times 8 \times 4$ subdivided domains; and quadratic polynomials over $2 \times 2 \times 1$ and $4 \times 4 \times 2$ subdivided domains. The data show that using more than $4 \times 4 \times 2$ subdomains and fitting quadratic polynomials does not significantly reduce error, so we use linear polynomials and a $4 \times 4 \times 2$ discretization of subdomains for all of our examples. Our method can approximate a variety of filters, and we compare the error of our method versus trilinear interpolation of mipmaps sampled with different filters. The errors of our method using eight texels relative to trilinear interpolation of box, tent, Gaussian, and Lánczos 2 filtered mipmaps is 0.569, 0.209, 0.142, and 0.232.

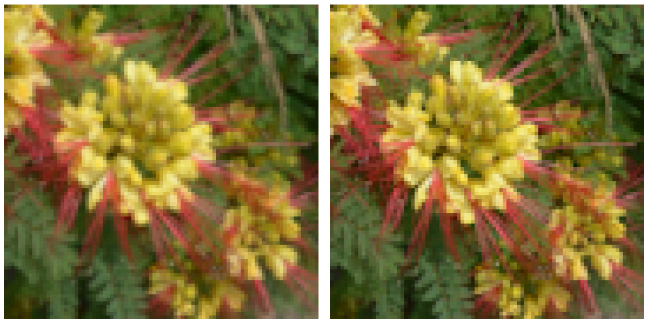
We show the times on an NVidia GeForce GTX 580 in Figure 7. We give two times for trilinear interpolation; one measurement is for the native hardware trilinear interpolation exposed by the shading language, and the second is our shader implementation of trilinear interpolation where we explicitly perform eight texel fetches. Our timing results do not match our prediction that our method should be only slightly slower per texel fetched than trilinear interpolation based on the number of mathematical operations performed. Our most plausible explanation is that we have lower throughput because trilinear interpolation has a more structured and cache-friendly memory access pattern.

We show an example of the access pattern of our method for a tent filter using eight texels in Figure 8. We read from three mipmap levels whereas trilinear interpolation reads from only two levels and it is likely that GPUs optimize for trilinear accesses by using two caches for alternate mipmap levels [Igehy et al. 1998], and that reading from three levels causes cache conflicts. GPUs are also likely to optimize for the 2×2 quads of texels accessed by a trilinear interpolant, whereas our fetches are less regular. Even our software implementation of trilinear interpolation takes $1.5\times$ more time and bandwidth than the native hardware implementation despite fetching the same texels. Our method will also issue irregular reads for adjacent pixels because neighboring pixels in a $4 \times 4 \times 2$ discretization will have a stride of at least one subdomain. GPU profiling tools show that our method fetches more texels than we expect and that time taken is almost directly proportional to the number of texels fetched between our method, our trilinear implementation, and the hardware trilinear interpolant. Our tests are consistent between ATI and NVidia GPUs, and show that a native hardware implementation significantly improves the performance of trilinear interpolation. It is possible that hardware designed for our sampling pattern would achieve similar speedups.

Figure 1 demonstrates that generating mipmaps with a high-quality filter is insufficient to produce sharp images at arbitrary scales when sampled using trilinear interpolation. Trilinear interpolation gives the correct filtered values when evaluated at a texel, but does a poor job between texels, even at the same scale as one of the mipmap images. In contrast, our method minimizes the error over all points. We show an example of an image that we sample between mipmap levels in Figure 1 using a direct convolution of a Lánczos 2 filter as the ground truth, our approximation of the Lánczos 2 filter using eight texels, and trilinear interpolation on mipmaps that are created using a Lánczos 2 filter. The Lánczos filter and our approximation of the Lánczos filter look nearly the same, but trilinear interpolation produces an image that is blurry.

Our method can be tuned to use different numbers of texels for fine-grained control over the memory bandwidth and quality of texture sampling. We show an example where we compare trilinear interpolation, our method with four texels, our method with eight texels, and exact evaluation of the Lánczos 2 filter on a two-dimensional image in Figure 9. This image contains high-frequency details aligned in all directions: horizontally, vertically, and diagonally. When using both four and eight texels, our method produces similar results to the exact filter, whereas trilinear interpolation of the Lánczos 2 filtered mipmap produces a blurry image. Figures 6, 9, and 11 provide quantitative and qualitative evidence that our method smoothly adapts image quality to available memory bandwidth.

We compare the visual quality of trilinear interpolation to our method for a three-dimensional scene using a Lánczos 2 filter in Figure 10. Again, trilinear interpolation produces an image that is blurry, whereas our approximation is sharper. In Figure 11 we show a checkerboard pattern on an infinite plane using a tent filter when fetching four, six, and eight texels to demonstrate aliasing. The texture has ten checkers on a side, so that there is not an even power of two checkers to texels; hence, a poor filter cannot easily hide aliasing patterns. When using eight texels, the same number of texels fetched in trilinear interpolation, our filter looks sharp and clear. The results when from using six texels are almost indistinguishable from eight texels, despite using 75% of the bandwidth. When using only four texels, the image in the distance appears slightly noisier, and edges in the foreground appear somewhat rougher.


(a) *Trilinear* **(b)** *4 Texels*

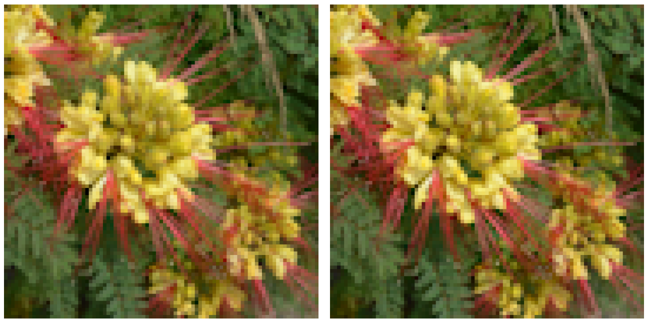

(c) *8 Texels* **(d)** *Exact*

Figure 9: *We show images downsampled using (a) trilinear interpolation, our approximation of the Lánczos 2 filter using (b) 4 and (c) 8 texels, then (d) exact evaluation of the Lánczos 2 filter.*

A possible concern is that flickering or popping artifacts will occur in animated scenes because of the piecewise nature of our method. In our tests, we have seen no obvious flickering. Although coefficients change discontinuously across subdomain boundaries, the filter we are approximating changes continuously, and our approximation error is typically low enough that no artifacts are detectable. For the particularly challenging scene of rotating the checker pattern in Figure 11, we could see a single transition line in the distance when the method first samples from a very coarse resolution mipmap for a Lánczos 2 filter with $n = 4$. However, this artifact was data dependent as we did not see the problem at $n = 4$ for other images. When $n > 5$, we did not see any artifacts in any images under animation for the Lánczos 2 filter and when we used a tent filter, we found that we could not see the transition line with $n = 4$ because a tent filter is blurrier than a Lánczos 2 filter.

Our optimized texture samples can also be used to improve the results of anisotropic texture filters that combine isotropic samples. Hardware anisotropic filtering uses the model of Feline [McCormack et al. 1999], where anisotropic filters are approximated by summing smaller isotropic filters. Feline approximates stretched Gaussians and uses trilinear interpolation to cheaply approximate isotropic samples. By replacing trilinear interpolation with our approximation of the isotropic Gaussians, we generate higher-quality anisotropic filters while using the same texture bandwidth. We compare the results of Feline and our improved anisotropic sampling in Figure 12. Using more isotropic samples in Feline can improve filtering in the direction of stretch, but increasing the quality in the perpendicular direction requires better isotropic samples.

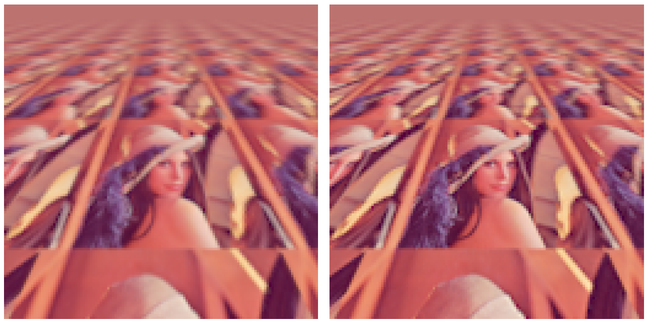

(a) *Trilinear* **(b)** *Our Method*

Figure 10: *We show (a) trilinear interpolation of a Lánczos 2 filtered mipmap compared against (b) our approximation of the Lánczos 2 filter using 8 texels.*

5 Conclusions and Future Work

We believe that our method is of practical value because memory bandwidth is often a bottleneck in graphics applications. A limitation of our method is that GPUs have been designed to optimize for trilinear interpolation and do not perform well on less structured reads. This leaves several interpretations for the role of our method. One is that our method is more suitable for offline rasterizers and ray-tracers with more flexible pipelines. Another possibility is that hardware designs will change to better support random access or even the access pattern of our method. Our paper can also be viewed as a stepping stone. We have shown that better filtering is possible by optimizing which texels and coefficients to use under the simple assumption that cost is proportional to number of texels fetched. It may be possible to incorporate the current texel fetch behavior of GPUs in our optimization. For example, we could optimize for reading quads of texels using bilinear interpolation.

Our paper focuses on improving the quality of isotropic texture filtering. When displaying two-dimensional images such as in Figures 1, 3, and 9, or when viewing a surface straight-on, anisotropic filtering does not apply. It is possible to improve anisotropic filtering by replacing isotropic probes used in current hardware with our method as in Figure 12, but we could also directly apply the principle of optimizing for the best set of texels and their coefficients to anisotropic texture filtering. This could improve sampling quality relative to the number of texels used by reducing the number of redundant texel reads. The challenge of extending our method to anisotropic filters is that the dimensionality of the optimization increases from three to five dimensions because we must include stretch and orientation of the filter, which, in turn, increases the complexity of the optimization. The idea of simultaneously optimizing basis functions and their coefficients for filter reproduction [Gotsman 1994] has potential for producing even better results when combined with our idea of optimizing for which texels to use from different resolutions; although the simultaneous optimization may be complex to solve.

Acknowledgements

This work was supported by NSF CAREER award IIS 1148976 and the NSF Graduate Research Fellowship Program.

References

BURT, P., AND ADELSON, E. 1983. The laplacian pyramid as

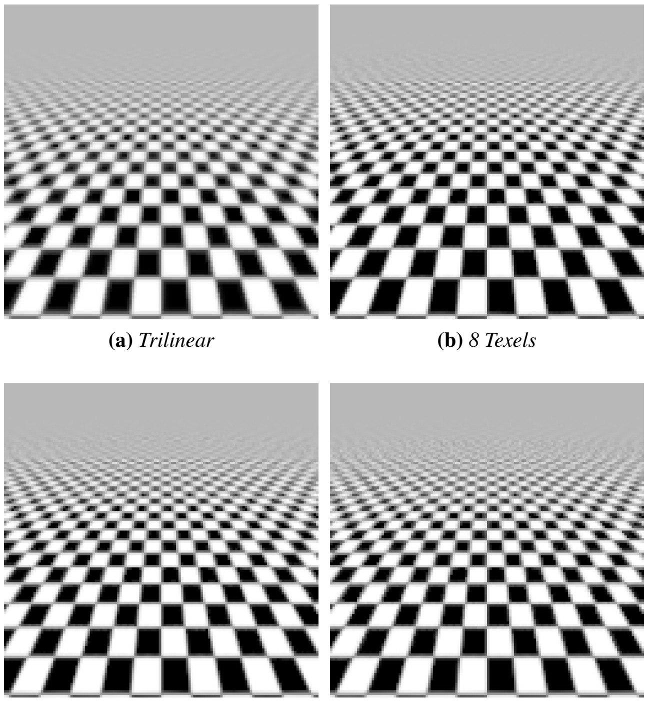

(a) *Trilinear* **(b)** *8 Texels* **(c)** *6 Texels* **(d)** *4 Texels*

Figure 11: *This figure demonstrates aliasing of a checker pattern with ten squares on a side repeated over an infinite plane. We show the results of (a) trilinear interpolation and our approximation of the tent filter using (b) 8, (c) 6, and (d) 4 texels.*

a compact image code. *IEEE Transactions on Communications 31*, 4, 532–540.

BURT, P. 1981. Fast filter transform for image processing. *Computer Graphics and Image Processing 16*, 1, 20–51.

CANT, R., AND SHRUBSOLE, P. 1997. Texture potential mapping: A way to provide antialiased texture without blurring. In *Visualization and Modelling*, 223–240.

CANT, R., AND SHRUBSOLE, P. 2000. Texture potential mip mapping, a new high-quality texture antialiasing algorithm. *ACM Transactions on Graphics 19*, 3, 164–184.

CHEN, B., DACHILLE, F., AND KAUFMAN, A. E. 2004. Footprint area sampled texturing. *IEEE Transactions on Visualization and Computer Graphics 10*, 2, 230–240.

CROW, F. C. 1984. Summed-area tables for texture mapping. In *SIGGRAPH*, 207–212.

DUCHON, C. 1979. Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology 18*, 8, 1016–1022.

FOURNIER, A., FIUME, E., AND BUILDING, S. F. 1988. Constant-time filtering with space-variant kernels. In *SIGGRAPH*, 229–238.

GLASSNER, A. 1986. Adaptive precision in texture mapping. In *SIGGRAPH*, 297–306.

GOTSMAN, C. 1994. Constant-time filtering by singular value decomposition. *Computer Graphics Forum 13*, 2, 153–163.

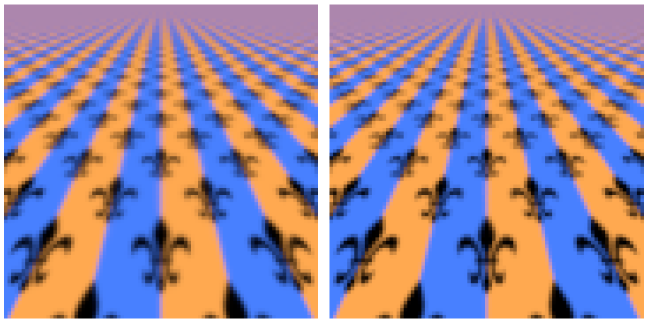

(a) *Feline Trilinear* **(b)** *Feline 8 Texels*

Figure 12: *We compare (a) the Feline algorithm using trilinear probes, and (b) Feline using our method to more accurately reproduce isotropic Gaussian probes.*

GREENE, N., AND HECKBERT, P. 1986. Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications 6*, 6, 21–27.

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

HÜTTNER, T., AND STRASSER, W. 1999. Fast footprint mipmapping. In *Proceedings of the SIGGRAPH/EUROGRAPHICS workshop on graphics hardware*, 35–44.

IGEHY, H., ELDRIDGE, M., AND PROUDFOOT, K. 1998. Prefetching in a texture cache architecture. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware*, 133–143.

MALLAT, S. 1989. A theory for multiresolution signal decomposition: the wavelet representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 11*, 7, 674–693.

MAVRIDIS, P., AND PAPAIOANNOU, G. 2011. High quality elliptical texture filtering on gpu. In *Symposium on Interactive 3D Graphics and Games*, 23–30.

MCCORMACK, J., PERRY, R. N., FARKAS, K. I., AND JOUPPI, N. P. 1999. Feline: Fast elliptical lines for anisotropic texture mapping. In *SIGGRAPH*, 243–250.

MITCHELL, D. P., AND NETRAVALI, A. N. 1988. Reconstruction filters in computer-graphics. *ACM Computer Graphics 22*, 221–228.

SCHILLING, A., KNITTEL, G., AND STRASSER, W. 1996. Texram: a smart memory for texturing. *Computer Graphics and Applications, IEEE 16*, 3, 32–41.

SHANNON, C. 1949. Communication in the presence of noise. *Proceedings of the IRE 37*, 1, 10–21.

WELCH, W. J. 1982. Algorithmic complexity: three NP-hard problems in computational statistics. *Journal of Statistical Computation and Simulation 15*, 1, 17–25.

WILLIAMS, L. 1983. Pyramidal parametrics. In *SIGGRAPH*, 1–11.

ZHOUCHEN LIN, L. W., AND WAN, L. 2006. First order approximation for texture filtering. In *Pacific Graphics Poster*.