Eurographics Symposium on Rendering 2016
E. Eisemann and E. Fiume
(Guest Editors)

Volume 35 (2016), Number 4

Fast Filtering of Reflection Probes

Josiah Manson and Peter-Pike Sloan

Activision

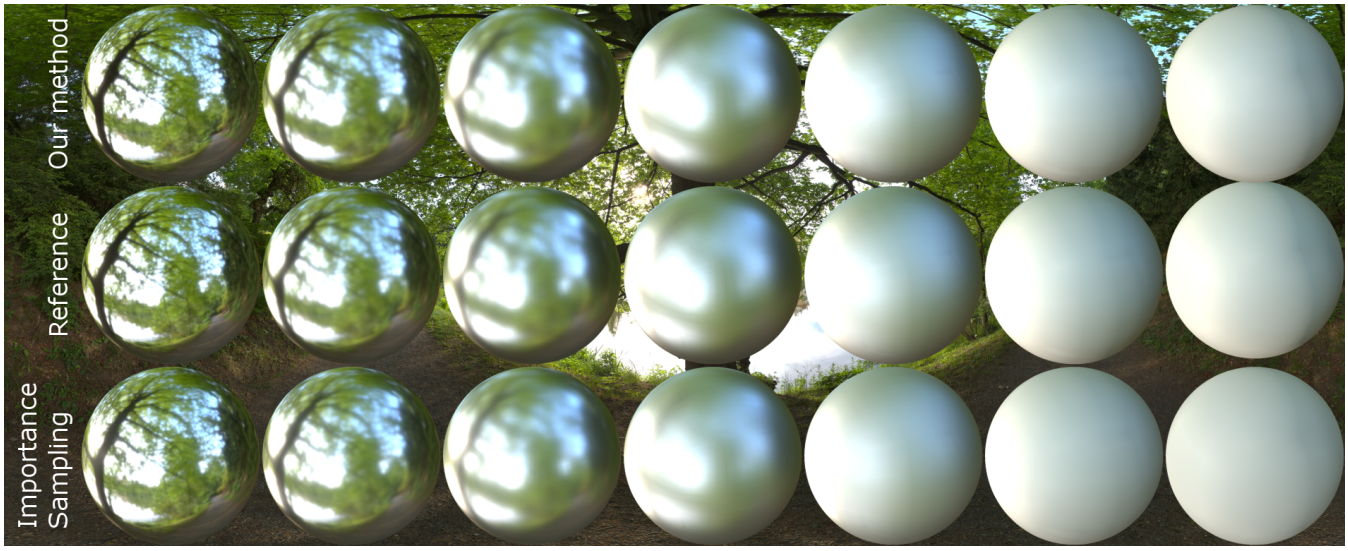

Figure 1: We show the results of convolving a high-dynamic-range image using GGX filters of different roughness for each mip-level of a cubemap. The results are an equal time comparison of our 8x3 sample quadratic method (top) and 32 importance samples with LOD selection (bottom) shown beside analytically evaluated reference images (middle). The L_1 errors are 0.0848 for our method and 0.4201 for importance sampling. ©Peter Sanitra, http://noemotionhdrs.net, Creative Commons License.

Abstract

Game and movie studios are switching to physically based rendering en masse, but physically accurate filter convolution is difficult to do quickly enough to update reflection probes in real-time. Cubemap filtering has also become a bottleneck in the content processing pipeline. We have developed a two-pass filtering algorithm that is specialized for isotropic reflection kernels, is several times faster than existing algorithms, and produces superior results. The first pass uses a quadratic b-spline recurrence that is modified for cubemaps. The second pass uses lookup tables to determine optimal sampling in terms of placement, mipmap level, and coefficients. Filtering a full 128^2 *cubemap on an NVIDIA GeForce GTX 980 takes between 160 µs and 730 µs with our method, depending on the desired quality.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

The reflection properties of materials are crucial for overall scene appearance because the distribution of light reflected from a surface determines not only the appearance of that surface, but reflections also indirectly light other surfaces. Recent movies and games are moving away from ad-hoc lighting models towards physically based rendering (PBR). Physically plausible reflection of light is important even when a cartoon aesthetic is desired because fewer

lights and less time is required for artists to make materials seem like they fit in the environment.

General bidirectional reflection distribution functions (BRDFs) are often too complicated to measure or evaluate, so simplified models are used to represent reflection distributions. A commonly used model is to characterize light as either diffusely or specularly reflected. Diffusely reflected light penetrates the surface, scatters internally, and is emitted equally in all directions. Specularly reflected light reflects directly off the surface like a mirror. Surfaces appear glossy rather than mirror smooth because of small bumps or roughness of the surface. Microfacet reflection models use distributions of surface normals to determine the directions that light is reflected. Several models exist, but we focus on reproducing the results of GGX [WMLT07].

To render images at 30 or 60 frames per second, games further simplify specular reflection models. It is too costly to directly calculate the light field at every point on visible surfaces, so incoming light is computed at a few important locations called reflection probes. Whenever the location of a probe or the environment around it changes, the probe must be recalculated. Glossy reflections are stored in a mipmap such that sharp reflections are stored in higher-resolution mips and blurry reflections are stored in lower resolution mips. While rendering a surface, trilinear interpolation is used to approximate intermediate levels of gloss [AG02, MEW*13].

Filtering algorithms must be fast as well as accurate. Speed requires that only a few samples are read per output texel. Our goal is to use each sample as effectively as possible. Quickly filtering cubemaps is important even during preprocessing. A typical level in a game has on the order of a hundred probes and current filtering methods can take seconds to accurately filter one probe [Man14]. Our algorithm takes less than a millisecond to produce a good approximation, which reduces the build time of a level by minutes and enables real-time update of physically based reflections.

1.1. Related Work

Early work on BRDF approximation focused on metallic surfaces. Specular reflections from metallic surfaces are the product of the surface color and the incoming light. One method utilized this fact to approximate the reflected light from arbitrary BRDFs as a product of two cubemaps that are correspondingly sampled using the surface normal and the reflection vectors [LK02]. The cubemaps depend on the BRDF and are solved for using a regularized linear solver and roughly correspond to the diffuse and specular components of the reflected light. Most recent reflection models assume dielectric materials, and in the case of a metal would simply multiply the specular color by an auxiliary specular color map.

Approaches that are used to filter images in two dimensions are not directly applicable to filtering cubemaps. There are a couple of factors that make it difficult to filter cubemaps. One difficulty is that it is necessary to filter across seams between faces of a cubemap. Corners are challenging because there is no regular parameterization where three faces meet. Another difficulty is that, although the filter kernel is radially symmetric on a sphere, radial symmetry is lost when projecting onto cube faces. The filter is anisotropic on the cube, but most anisotropic filtering methods are designed to filter distorted box filters [HS99] or Gaussians [GH86]. Summed area tables [Cro84, HSC*05] can be used to approximate a filter by rectangular regions, but require high-precision. Summed area tables are adequate for a coarse approximation, but accurately approximating smooth filters with piecewise constant functions converges slowly.

Importance sampling is a common numeric integration technique in which points are sampled with a density that is approximately proportional to the function value. For example, the authors of the GGX reflection model [WMLT07] describe an importance function that can be used with their reflection model. The GGX kernel has a long tail that is sparsely sampled by importance sampling, but the tail contributes enough energy that undersampling manifests as visible noise. Noise is particularly noticeable in environments with high dynamic range, which are common in PBR, and thousands of point samples are needed per texel. Importance sampling is inefficient for texture filtering because raster images are band-limited, which results in the peak of the function being oversampled.

When a large number of samples are taken, cubemaps can be filtered more efficiently by directly reading texels rather than sampling the cubemap as though it is an arbitrary function. Processing texels in scanline order reads each texel once and has a cache-friendly access pattern. It is wasteful to read texels where the filter kernel has very small values, so angular extent filtering [IM05] windows the kernel to some solid angle. A cone enclosing the filter is projected onto cube faces and enclosed in bounding boxes. Texels are then read in scanline order within the bounding boxes. GPU calculations are accelerated by using cubemap lookups for solid angles of texels and a radial filter fall-off table, but dense sampling within angular bounds reads too many texels.

Kautz et al. [KVHS00] showed that it is efficient to downsample an environment map into a mipmap hierarchy using a simple downsampling filter and then draw samples from the mipmap to evaluate a more sophisticated filter. The CPU algorithm they presented used recursive refinements that are impractical for GPU implementation. The hardware accelerated algorithm they presented was simpler, but did not use multiple levels of the mipmap or properly account for projected filter sizes.

Given a fixed number of samples, a mip-hierarchy can also be used with importance sampling [KC08]. The spacing between samples will be inversely proportional to the square root of the importance function, which can be used to sample from a mip-level that was prefiltered with an appropriately wide kernel. This method assumes that a box-filter is used to generate the mipmap and does not consider the effect of the intermediate filter on the final image. In our figures and tables that compare against importance sampling, we are referring to this method.

Cardinality-constrained texture filtering (CCTF) [MS13, MS14] uses non-linear optimization to precalculate a table of texel indices and coefficients. CCTF combines basis functions from different levels of a planar mipmap to approximate filter kernels, but its goal is to create an interpolant that replaces trilinear interpolation with one that is equally fast but higher quality. In contrast, our method combines trilinear samples to approximate the GGX filter, which is defined over a sphere. Filtering a kernel that is defined on a sphere

over a cubemap requires that we correct for distortions that result from mapping a sphere to a cube and requires a different solution for each texel.

The irradiance kernel is the positive lobe of the cosine function on a sphere. Spherical harmonics (SH) are well suited for representing low-frequency functions on a sphere [RH01] and are often used to store irradiance. However, projecting irradiance into a low-order SH basis is prone to ringing for HDR images [Slo08] and high-order basis functions are expensive to evaluate. Rather than using just a cubemap or SH, it is possible to combine both approaches by storing SH coefficients in each texel of a cubemap [RH02]. This combined approach is effective for representing anisotropic BRDFs where two dimensions are used for cubemap lookup and two for SH evaluation for the total of four dimensions of a BRDF. However, we are evaluating a simplified, isotropic reflection model in this paper, so we use a cubemap only.

2. Background

The outgoing radiance $R(v)$ in the viewing direction v can be computed by integrating the reflectance distribution $f(l,v)$ of the incoming light field $L(x)$ over the visible hemisphere H, which is oriented along the surface normal n. For our purposes, $L(x)$ is represented as a cubemap texture.

$$R(v) = \int_H L(l) f(l,v)(n \cdot l)\, dl \tag{1}$$

In the case of diffuse reflection, the reflection function is constant for all directions and is modulated by the diffuse albedo c_{diff}.

$$f_{diff}(l,v) = \frac{c_{diff}}{\pi}$$

After substituting f_{diff} into Equation 1, we can factor the equation for diffusely reflected light as the product of the surface albedo c_{diff} times irradiance

$$\int_H \frac{L(l)(n \cdot l)}{\pi}\, dl. \tag{2}$$

We precalculate and store the irradiance for different view directions v in a cubemap. Evaluating irradiance can also be thought of as convolving the light probe with a filter kernel equal to the positive lobe of the cosine function.

Specular reflections are more complicated and we summarize the results from the paper that introduces GGX [WMLT07]. The general Cook-Torrance microfacet model for specular reflection is

$$f_{spec}(l,v) = \frac{D(h)G(l,v,h)F(v,h)}{4(n \cdot l)(n \cdot v)},$$

where $D(h)$ is the microfacet normal distribution, $G(l,v,h)$ is the geometric masking term, and $F(v,h)$ is the Fresnel term. GGX provides a physically plausible definition for $D(h)$ and $G(l,v,h)$. A simplified model of GGX that is often used in games [Laz11,Laz13, Kar13] splits the integral into two parts.

$$\begin{aligned} R(v) &= \int_H L(l) \frac{D(h)G(l,v,h)F(v,h)}{4(n \cdot v)}\, dl \\ &\approx \left(\int_H L(l)D(h)\, dl \right) \left(\int_H \frac{G(l,v,h)F(v,h)}{4(n \cdot v)}\, dl \right) \end{aligned}$$

Values of the second term can be precalculated once because they are independent of the light field. However, the first integral in the split-sum approximation is the inner product of $D(h)$ with the light field $L(l)$, which changes frequently. This integral can also be thought of as convolving the light field $L(l)$ with a filter, and we investigate how to accelerate evaluating

$$\int_H L(l)D(h)\, dl. \tag{3}$$

Note that $D(h)$ is a function of l because $h = \frac{l+v}{|l+v|}$ represents the microfacet normal that is half-way between l and v. The microfacet normal distribution $D(h)$ is defined as

$$D(h) = \frac{\alpha^2 \chi^+(n \cdot h)}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2},$$

where α represents the roughness of the surface and $\chi^+(n \cdot h)$ is 1 in front of the surface and 0 behind.

When importance sampling Equation 3, we use the microfacet normal distribution as the importance function. The angle of microfacet normals relative to the surface normal are calculated from uniform random numbers $\xi_0, \xi_1 \in [0,1]$ as

$$\theta = 2\pi\xi_0$$

$$\phi = \tan^{-1}\left(\frac{\alpha\sqrt{\xi_1}}{\sqrt{1-\xi_1}} \right).$$

Samples are then weighted by $|l \cdot n|$. The mip-level of an importance sample is found from the ratio of the solid angles SA of the sample and the texel. The solid angle of the sample SA_{sample} is equal to the size of a uniformly distributed sample divided by the importance function. The surface area of a texel is the area of a cube face, 4, weighted by the differential solid angle, $J(x,y,z)$, and divided by the number of texels on the face, $Resolution^2$. The Jacobian of the projection from a $[-1,1]^3$ cube to a unit sphere is $J(x,y,z) = \frac{1}{(x^2+y^2+z^2)^{3/2}}$.

$$SA_{sample} = \frac{4\pi}{N\, D\left(\frac{p+n}{\|p+n\|}\right)}$$

$$SA_{texel} = \frac{4J(p_x,p_y,p_z)}{Resolution^2}$$

$$MipLevel = \frac{1}{2}\log_2\left(\frac{SA_{sample}}{SA_{texel}}\right) \tag{4}$$

3. Overview of Method

Quickly evaluating filtered cubemaps in an interactive environment, requires our evaluation code to be simple and read few texels. Our method is designed to be implemented on a GPU, so we leverage the trilinear cubemap sampling provided by hardware. We decompose filter convolution into two passes and precomputation of a data-independent lookup table.

In the first pass, we quickly generate a rough approximation of the final filtered result as a mipmapped cubemap. Relatively few samples from this intermediate cubemap can then be used to produce a good approximation of the filter. This is analogous to a preconditioner in linear algebra, where a simple transformation is applied to a linear system in order to get faster convergence.

In the second pass, we combine samples from the intermediate cubemap to evaluate texels in the final filtered cubemap. For an efficient GPU implementation, our filter is formulated as a gather operation, where several texels are read for each output texel. It is critical that the sampling pattern changes continuously between output texels to reduce visual artifacts.

Complexity of the method is shifted away from evaluation and into table generation. Locations, mip-levels, and weights of samples are stored as low-order polynomials in a precomputed coefficient table. This polynomial representation allows sampling patterns to smoothly adapt to distortions from spherical projection onto a cubemap. Because table generation is done once as an offline process, we can optimize coefficients using an algorithm that is slow but produces high-quality results.

4. First Pass: Downsampling

The first step of our method is to downsample the input cubemap into a mipmap. The downsampled texels are a weighted combination of input texels, where the distribution of weights act as basis functions $b_i(x)$ that we combine to reproduce a filter in the second pass. Our downsampling algorithm must be fast because time spent in the first pass reduces available time in the second pass. Downsampling should also produce basis functions that have similar characteristics to the target function $B(x)$, so that few samples are required in the second pass to reach acceptably low approximation error.

The standard recursive box filter is a special case of b-splines. Specifically, the box filter is a 0-order b-spline, which has the recurrence $(1/2\ 1/2)$ [DB01]. The 2D recurrence needed for image filtering is the tensor-product of the 1D recurrence. Box filters are simple and fast, but combining samples from the resulting piecewise constant basis does not converge quickly in the second pass. The next b-spline that can be used on a power-of-two grid is the quadratic b-spline basis, which has the recurrence $(1/8\ 3/8\ 3/8\ 1/8)$. We tested mipmap filtering with up to a quartic basis and found that quadratic b-splines give the best trade-off between speed and quality.

Because quadratic b-splines are a 4×4 tensor product, we can efficiently evaluate the 16 texel footprint on a GPU using 4 bilinear samples (a similar optimization applies to cubic filters [SH05]). Consider the 1D case, of evaluating $1/8 s_1 + 3/8 s_2 + 3/8 s_3 + 1/8 s_4$.

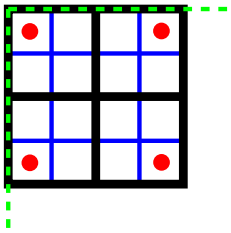

Figure 2: We show a graphical depiction of quadratic b-spline downsampling. The black outlined cells show a quad of texels at the corner of a cubemap face, such that the texel centers are at the vertices of the dual grid shown in blue. The edges of the cubemap face are drawn as dashed green lines, and the bilinear sample coordinates are drawn as red dots. The red dots do not cross the face boundary.

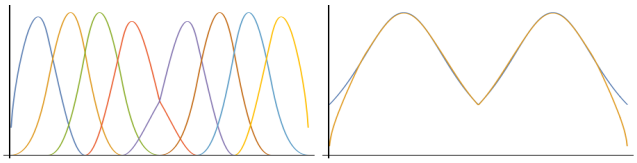

Figure 3: A 1D example of weight modification of quadratic b-splines. On the left, we show basis functions weighted by the "Jacobian", and on the right we show the "Jacobian" in blue with our approximation overlaid in orange.

Given the function $lerp(a,b,t) = (1-t)a+tb$, we can rewrite the expression as $1/2(lerp(s_1,s_2,3/4)+lerp(s_3,s_4,1/4)$. We do not need to do anything special to filter across cubemap faces because the bilinear samples, shown as red dots in Figure 2, lie within the same face. The GPU will perform the necessary coordinate transformations to read from and blend across cubemap edges.

Quadratic b-splines are smooth, but the projection of a constant function over a sphere onto a cubemap is not smooth between faces. To get a basis that better approximates a constant over a sphere, we weight each of the samples in the b-spline recurrence by the Jacobian, $J(x,y,z)$. Weighting samples by the Jacobian has the desired effect of giving less weight to corners and edges, and produces functions that are smooth everywhere except for at edges.

We demonstrate the idea of weighting by the Jacobian on a 1D function that has a similar shape to $J(x,y,z)$ in Figure 3. The basis functions have different shapes depending on their position and the middle functions have a non-smooth bend, which allows us to approximate non-smooth objective function well. The boundary does not match because we did not use a periodic boundary condition when calculating basis functions for this demonstration figure.

In practice, we modified the weighting function to not quite multiply by $J(x,y,z)$, because the distortion to the basis functions was too strong. We found that we got lower error by blending the Jacobian with a constant function $1/2(1+J(x,y,z))$. Individually weighting each of the 16 texels in the cubemap recurrence is not possible when using bilinear texture reads. We weight the 4 bilinear samples instead, which is faster but results in subtle non-smoothness over the function. We tried offsetting the positions and

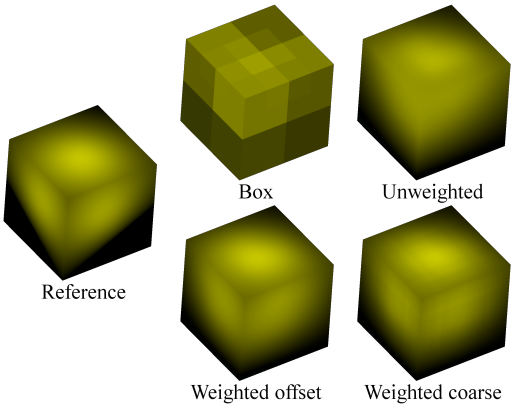

Figure 4: Different basis functions reproducing the GGX kernel of a rough surface using our 8x3 sample quadratic fit. A good set of basis functions can significantly reduce error, especially when combining a low number of samples.

Figure 5: A depiction of the weights of the three frames in a t-projection of a cubemap. The weights of the x, y, z axial frames are shown in the r, g, b color channels respectively.

weights of the bilinear parameters to better approximate the 16 texel weights, but calculating the bilinear offsets was too expensive relative to the resulting benefit.

We compare the reconstruction of a large filter kernel using different basis functions in Figure 4. The discontinuous box function is not suitable for reproducing a continuous function. The unweighted quadratic basis performs better than the box basis, but cannot accurately reproduce the reduced values at the edges of the cube. Both of the weighted quadratic bases reproduce the edges more accurately and the basis functions with coarse weights look almost the same as the more accurately weighted functions.

5. Second Pass: Filter Approximation

The second pass of our algorithm combines multiple trilinear samples from the mipmapped cubemap to approximate the ideally filtered results. We define sample offsets in a polar frame that changes smoothly everywhere except for point discontinuities at the poles. The position of a sample is defined by its offset with respect to the coordinate system $\hat{x}, \hat{y}, \hat{z}$ of the output texel. Given the direction $\hat{z} = \hat{n}$ toward the texel and a polar axis $\hat{a}$, we define the tangent frame as $\hat{x} = normalize(\hat{a} \times \hat{z})$, $\hat{y} = normalize(\hat{a} - \hat{z}(\hat{a} \cdot \hat{z}))$. This tangent field has singularities at $\pm\hat{a}$, which we would like to avoid. Each tangent field is weighted such that faces that do not intersect the pole have a value of one and large portions of the faces that intersect the pole are zero and change continuously.

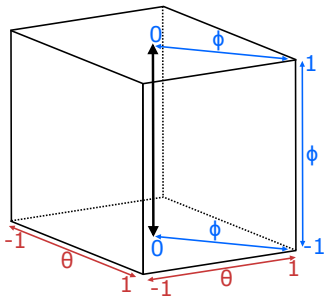

Figure 6: The parameterization of the cube. The coordinates change linearly along each line. There are discontinuities in θ at vertical edges of the cube, but ϕ changes continuously from 0 to -1 to 1 and back to 0 when tracing from bottom to top.

We remove point discontinuities by overlapping three polar parameterizations that are oriented along the Cartesian axes. Each parameterization has a weight that is one at the equator and zero near the poles as shown in Figure 5. Without loss of generality, assume that a texel in the cubemap has coordinates $\hat{n} = (n_x, n_y, n_z)$ and that the pole is along the z-axis. We define the weight of the frame as $W_{frame}(\hat{n}) = saturate(4\, max(abs(n_x), abs(n_y)) - 3)$. One can think of a cube face as being divided into 8×8 tiles. The center 36 tiles have contributions from 2 frames, and the remaining 28 edge tiles have contributions from 3 frames. Because of the regular tiling between cases, most thread groups on a GPU will have coherent control flow.

Each frame has N independently controlled samples, and each sample has 5 parameters: the offset within the frame (x_i, y_i, z_i), the mip-level l_i, and the weight w_i of the sample. The position of the sample p is defined within the frame coordinates as $p = x_i\hat{x} + y_i\hat{y} + z_i\hat{z}$. The weights are encoded prior to multiplication by the frame weight, and after all of the samples are taken, the weight is normalized to ensure reproduction of constant colors. The texel color is calculated as

$$\frac{\sum_{frame=0}^{2} W_{frame}(n) \sum_{i=0}^{N-1} w_i\, sample(p, l_i)}{\sum_{frame=0}^{2} W_{frame}(n) \sum_{i=0}^{N-1} w_i}.$$

The sampling function applies a mip-level offset that is a function of the sampling position and adjusts for the Jacobian of the sphere to cube mapping. This offset is simply the contribution of the Jacobian factored out of Equation 4.

In addition to increasing the number of samples, we can reduce error by having the sampling parameters change as a function of the output texel's position. Better spatial adaptation more accurately accounts for the nonuniform projection of a filter onto the cubemap. We add spatial adaptation by evaluating each of the sample parameters as a low-order polynomial of the parameters θ, ϕ defined over the cube as shown in Figure 6. Going counter-clockwise around the pole, θ changes linearly from -1 to 1 across each face, and ϕ changes linearly from -1 at the bottom of a side to 1 at the top.

The ϕ coordinate reflects at the top and bottom edges of the cube, such that ϕ is 0 at the poles and changes linearly to connect contin-

uously at the edges. This reflection of ϕ is meant to represent the fact that projective warping is symmetric across edges. The top and bottom faces have very little weight, so we use this reflection trick rather than adding more degrees of freedom to properly handle the top and bottom faces.

Continuity of the preimages themselves is unimportant for continuity of the image. Rather, we must ensure that the preimages change continuously between neighboring texels for the filtered image to be continuous. The parameterization from -1 to 1 for each face is naturally continuous for symmetric functions, such as even powered monomials. Odd powered monomials are not symmetric, but a polynomial is continuous when coefficients of the odd powers sum to zero. In the case of a quadratic, the continuity constraint means that linear functions are unused.

6. Precomputation: Coefficient Table Optimization

The polynomial parameters for sample placement are stored in a table that is calculated offline. The contents of the table are independent of image data, so the table only needs to be calculated once. We therefore focus on reducing the perceived error as much as possible, at the cost of a lengthy optimization.

Integrals for the diffuse (Equation 2) and specular (Equation 3) filters are defined over a hemisphere, but we operate in the domain of cubemaps. For each texel, we use $B(x)$ to represent the projection of the filter associated with the texel onto the cubemap. In the case of specular filters, the roughness of the glossy reflection varies with the mipmap resolution. The projection of $D(h)$ onto the cubemap produces a filter that has anisotropic distortion and wraps across faces, which has a different optimal solution for each texel.

Every texel, indexed by i, in the intermediate mipmap has a contribution $b_i(x)$ from the input texture. The color of a filtered texel $\int L(x)B(x)\,dx$ is the integral of the preimage of the filter $B(x)$ times the light field $L(x)$. In particular, $B(x)$ is the projection of the GGX or irradiance kernels. The color of a texel in the intermediate mipmap is equal to $\int L(x)b_i(x)$ and we would like to find a sum of texels in the mipmap weighted by coefficients c_i that approximate the color of the texel filtered by $B(x)$.

$$\int L(x)B(x)\,dx \approx \sum_i c_i \int L(x)b_i(x)\,dx$$

Rearranging the equation, we find that the light field $L(x)$ is a common factor.

$$\int L(x)\left|B(x) - \sum_i c_i b_i(x)\right|\,dx \approx 0$$

The intensity $L(x)$ of the light at a point x weights the importance of approximating the preimage $B(x)$ at x. We therefore remove $L(x)$ from the equation to find an approximation that is not specialized to a particular light field [MS13].

$$\int \left|B(x) - \sum_i c_i b_i(x)\right|\,dx \approx 0$$

Instead of directly solving for the coefficients c_i, we solve for parameters of trilinear samples. Each trilinear sample contributes a fraction of its weight to up to eight texel coefficients.

There are several ways to measure the error between a pair of preimages, of which, the L_2 measure is most common. Minimizing L_2 error will approximate the peak of a function well, but at expense of approximating the tail poorly. Unfortunately, the tail of the GGX filter collects a significant amount of energy spread over a large area. We found that L_1 is a more perceptually relevant measure because small errors that accumulate over a large area are equally important to the output texel color as large errors accumulated over a small area. Minimizing an L_1 objective function is significantly more expensive than L_2, but spending the extra time for improved quality is worthwhile.

For each texel in the output image, we measure error by comparing the preimage of our approximation to the reference preimage. To calculate the preimage of our approximation, we initialize a mipmap of weights to be zero. Each of the N trilinear samples adds its contribution into 8 texels in the mipmap. Then, we use the recurrence relation to calculate the weights in successively higher resolutions of the mipmap until all of the weights are in the highest resolution of the mipmap. This can be thought of as the inverse of the mipmap generation described in Section 4.

Sampling patterns are initialized with importance sampling [KC08], which we then optimize using the BFGS algorithm [LN89] with golden section line searches and numerically approximated gradients. BFGS works reasonably well despite the L_1 error metric having discontinuous gradients [LO08]. The interaction between samples from the three axial frames is complex even in the case of constant sampling parameters, and requires optimization over many texels. Rather than measuring the error at every texel in the output cubemap, we use a sparse sampling so that every mip-level takes approximately the same amount of time to compute despite the number of texels being exponential in level.

7. Results

7.1. Analysis of Time and Error

We show results from filtering four example HDR scenes in Figure 7 with different lighting conditions. When we specify that Nx3 samples are used by our method, we mean that each of the 3 coordinate frames may read up to N trilinear samples from the mipmapped cubemap that is generated by the first pass ($^{39}/_{16}$ N samples on average). We compare the filtered images calculated with 32 importance samples and our 8x3 sample quadratic method because both methods take an equal time of 200 μs to compute. The first scene is taken during the day with a clear sky so that the sun is bright and directly visible, whereas the sun is setting and dimmer in the second scene. The third scene shows the interior of a building, and the the fourth image is of a city at night. Because it is difficult to see differences between adjacent images, we show component-wise, absolute differences from the reference and write the L_1 error next to the difference image. The importance sampled images tend to be too hazy, and highlights are often slightly off-center.

The characteristics of the filters are more apparent in synthetic point-response functions. Cubemaps are not spherically symmetric, so we show the response from a texel in the center of a face, in the middle of an edge, and at a vertex in Figure 8. Given equal computation time, the importance sampled filter has noticeably lower

Figure 7: Equal time comparison of our quadratic method with 8x3 samples (left) vs. 32 importance samples (right), and reference images (center) of mip-levels 1 and 3. L_1 errors are shown next to difference images that are one minus the absolute value of the difference between neighboring images (white means no error). Values are scaled to fill the dynamic range, and the same scale is applied to our method and importance sampling. ©Peter Sanitra, http://noemotionhdrs.net, Creative Commons License.

quality. Because the importance samples are asymmetric, the point response function is noticeably off-center.

Our goal was to make an algorithm that quickly approximates cubemap filters with sufficient accuracy for real-time applications, which makes it important to understand the trade-off between speed and accuracy. We report the computation time and errors of importance sampling compared to our method using different numbers of samples in Table 1. We give the times for filtering all six faces of a 128^2 source cubemap into the $128^2 - 2^2$ mip levels of the destination cubemap. All timings were measured on an NVIDIA GeForce GTX 980 using DirectX time-stamp queries.

Our algorithm relies on precalculated tables for its speed, which can take a few days to generate on a workstation. To be fair, we also precalculate tables for importance sampling. These tables circumvent random number generation and several transcendental functions that would otherwise be required to importance sample a GGX filter, while generating identical results. For each importance sample, we pack the microfacet normal l relative to the local frame

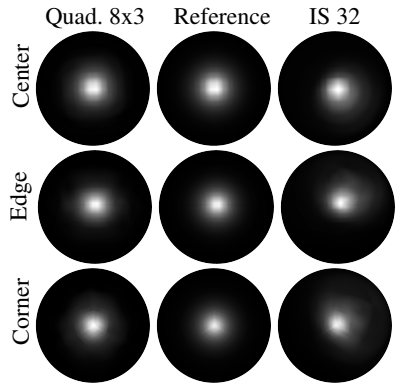

Figure 8: Result of filtering images with a single non-zero input texel located in the center, on the edge, and in the corner of a cubemap face. The third mip-level is shown for each input with the reference, IS 32, and Quad 8x3 filters.

Ours	Time	Error	IS	Time	Error
Const. 8x3	162 μs	0.1111	IS 32	203 μs	0.4201
Const. 16x3	263 μs	0.0815	IS 64	374 μs	0.2976
Const. 32x3	575 μs	0.0613	IS 128	772 μs	0.2233
Quad. 8x3	224 μs	0.0848	IS 1K	6.18 ms	0.0887
Quad. 16x3	480 μs	0.0656	IS 2K	12.2 ms	0.0658
Quad. 32x3	726 μs	0.0506	IS 4K	24.2 ms	0.0500

Table 1: Median time over 101 trials to GGX filter all mips of a 128^2 cubemap using importance sampling and our method. Average L_1 fitting errors exclude level zero for fair comparison.

and the solid angle of the sample into a *float4*. A further optimization one could perform for any filtering algorithm, but that we did not, is to copy the highest resolution level rather than filtering it. This would mean the highest resolution mip represents a perfect specular reflection and would reduce the processing time of the second filtering stage by a factor of approximately four.

The errors in Table 1 measure the L_1 error of the filter approximation itself. This is in contrast to the data-dependent errors reported in Figure 8. We exclude level 0 from the average error to not unfairly penalize importance sampling. Because we sample from a 128^2 into a 128^2 image using bilinear samples, naïve imporance sampling results in an additional convolution by the 128^2 resolution bilinear reconstruction filter, which prevents convergence for very small GGX filter kernels as shown in Table 2. Our method optimizes for reproduction of the preimage that results from trilinear filtering, so automatically accounts for this extra convolution. Note that importance sampling does not suffer from this aliasing problem if the input image is of sufficiently higher resolution than the output images.

A very rough specular reflection is nearly identical to a diffuse reflection. In typical image processing, a Fourier transform can be used to quickly convolve an image by a large kernel in the frequency domain. Likewise, spherical harmonics can be used to convolve images over a sphere, but are only practical to use for very large kernels such as the clamped cosine used to evaluate irradiance. We compare the speed and quality of our method against spherical harmonics to evaluate irradiance in Table 3. We modified a code generator to only generate the non-zero bands used to convolve with a clamped cosine kernel [Slo13]. These results show that spherical harmonics have significantly lower error given equal time. Although it is more efficient to convolve large kernels in the frequency domain, spherical harmonics are unusable for small kernels, and it may still be desirable to use our sampling algorithm for all filters to reduce code complexity.

	Quad. 32x3	IS 4096
Level 0	0.0495	0.2484
Level 1	0.0461	0.0502
Level 2	0.0516	0.0375
Level 3	0.0410	0.0517
Level 4	0.0394	0.0521
Level 5	0.0502	0.0512
Level 6	0.0758	0.0575

Table 2: Average fitting errors for the GGX filter evaluated using 4096 importance samples and our quadratic 32x3 filter.

Ours	Time	Error	SH	Time	Error
Const. 8x3	131 μs	0.1460	SH 3	87 μs	0.1227
Const. 16x3	211 μs	0.1186	SH 5	149 μs	0.0593
Const. 32x3	397 μs	0.0933	SH 7	246 μs	0.0358
Quad. 8x3	224 μs	0.1181	SH 9	376 μs	0.0243
Quad. 16x3	419 μs	0.0996	SH 11	585 μs	0.0177
Quad. 32x3	651 μs	0.0793			

Table 3: Fitting error for irradiance with our new method and with spherical harmonics.

7.2. Implementation details

The first stage of mipmap generation has a dependency chain where each level depends on the next higher resolution level. The 64^2 mip is processed in 10μs and the subsequent levels are each processed in 5μs for a total of 40μs. Few texels are processed for the lower resolutions, and the shader is quite simple, so the GPU is mostly idle for the smaller mip-levels. Although it is not possible to reduce latency, the first stage can share the GPU as an asynchronous compute job, or we could process batches of cubemaps to increase throughput.

Unlike the first stage, the second stage has no dependencies between mip-levels, so we can saturate the GPU by processing all mip-levels and faces of the cubemap in one dispatch. We organized the computation so that each thread processes one output texel in a nested loop over the three frames and the N samples. Grouping threads as squares of texels maximizes control flow coherency so that little work is wasted.

If throughput is more important than latency, it is possible to process batches of cubemaps to amortize the cost of table lookups, calculation of local frames, and polynomial evaluations. Amortizing computations that are independent of the value of input texels is possible with other methods besides ours. For example, the com-

putations that evaluate spherical harmonic basis functions can be shared. However, the most expensive part of evaluating a spherical harmonic on a GPU is the reduction of summing the contribution from each texel, so the benefit of amortization is less clear.

8. Conclusion

We have presented a method for quickly evaluating filters over cubemaps. If a cubemap is already used to store glossy reflections for an object, it is worth considering also storing irradiance in one of the low-resolution mips. Our method can optimize for any kernel, including the irradiance kernel, so our method is a unified approach for specular and diffuse lighting.

Our method works well for situations where it is okay to sacrifice some accuracy for fast evaluation. Specifically, this method has been deployed in the content pipelines of two shipping games and is being integrated into others. The main drawback of our method is that precalculating coefficient tables is slow and our optimization is easily trapped in local minima. As the number of samples increases, the polynomial order for coefficients will start to limit accuracy, so it will be necessary to use higher-order polynomials which entails larger tables, more precalculation, and slower shader evaluation. Scaling to large sample counts to get a very accurate filter is therefore impractical.

Combining texture samples with our method produces high-quality results when convolving small filter kernels such as GGX reflections with medium to high-gloss, whereas spherical harmonics are effective for convolving by large kernels such as the irradiance kernel. An interesting avenue of research would be to combine the two methods for medium to low-gloss. Spherical harmonics could be used to capture the large-scale features of the function and texture sampling could then be used to reduce the residual.

A common property of the data captured in a cubemap is that there are often only a few texels that are exceptionally bright. For example, the sun or light bulbs are small spots of intense light that are prone to revealing sampling artifacts. An interesting extension would be to remove those bright sources from the image and process them analytically. The remaining image would have relatively low dynamic range and could be processed with the method described in this paper.

Acknowledgements

Thanks to Dimitar Lazarov, Padraic Hennessy, Paul Malin, Bruce Wilkie, and Michał Iwanicki for discussions and help. Thanks to Peter Sanitra for HDR images from http://noemotionhdrs.net and permission to modify them.

References

[AG02] ASHIKHMIN M., GHOSH A.: Simple blurry reflections with environment maps. *J. Graph. Tools 7*, 4 (2002), 3–8. 2

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

[DB01] DE BOOR C.: *A practical guide to splines; rev. ed.* Applied mathematical sciences. Springer, Berlin, 2001. 4

[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

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

[HSC*05] HENSLEY J., SCHEUERMANN T., COOMBE G., SINGH M., LASTRA A.: Fast summed-area table generation and its applications. *Computer Graphics Forum 24* (2005), 547–555. 2

[IM05] ISIDORO J. R., MITCHELL J. L.: Angular extent filtering with edge fixup for seamless cubemap filtering. In *ACM SIGGRAPH 2005 Sketches* (2005). 2

[Kar13] KARIS B.: Real shading in unreal engine 4. In *SIGGRAPH Course Notes: Physically Based Shading in Theory and Practice* (2013). URL: http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf. 3

[KC08] KŘIVÁNEK J., COLBERT M.: Real-time shading with filtered importance sampling. *Computer Graphics Forum 27*, 4 (2008), 1147–1154. 2, 6

[KVHS00] KAUTZ J., VÁZQUEZ P.-P., HEIDRICH W., SEIDEL H.-P.: A unified approach to prefiltered environment maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques* (2000), pp. 185–196. 2

[Laz11] LAZAROV D.: Physically-based lighting in call of duty: Black ops. In *SIGGRAPH Course Notes: Advances in Real-Time Rendering in 3D Graphics and Games* (2011). URL: http://advances.realtimerendering.com/s2011/. 3

[Laz13] LAZAROV D.: Getting more physical in call of duty: Black ops ii. In *SIGGRAPH Course Notes: Physically Based Shading in Theory and Practice* (2013). URL: http://blog.selfshadow.com/publications/s2013-shading-course/lazarov/s2013_pbs_black_ops_2_notes.pdf. 3

[LK02] LATTA L., KOLB A.: Homomorphic factorization of brdf-based lighting computation. *ACM Trans. Graph. 21*, 3 (2002), 509–516. 2

[LN89] LIU D. C., NOCEDAL J.: On the limited memory BFGS method for large scale optimization. *Math. Program. 45*, 3 (1989), 503–528. 6

[LO08] LEWIS A. S., OVERTON M. L.: Nonsmooth optimization via bfgs. *Optimization Online* (2008). 6

[Man14] MANESKU D.: CMFT - cubemap filtering tool, 2014. URL: https://github.com/dariomanesku/cmft. 2

[MEW*13] MCGUIRE M., EVANGELAKOS D., WILCOX J., DONOW S., MARA M.: *Plausible Blinn-Phong Reflection of Standard Cube MIP-Maps*. Tech. Rep. CSTR201301, 47 Lab Campus Drive, Williamstown, MA 01267, USA, September 2013. 2

[MS13] MANSON J., SCHAEFER S.: Cardinality-constrained texture filtering. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH) 32*, 4 (2013), 140:1–140:8. 2, 6

[MS14] MANSON J., SCHAEFER S.: Bilinear accelerated filter approximation. *Computer Graphics Forum (Proceedings of EGSR) 33*, 4 (2014), 33–40. 2

[RH01] RAMAMOORTHI R., HANRAHAN P.: An efficient representation for irradiance environment maps. In *Proceedings of ACM SIGGRAPH* (2001), pp. 497–500. 3

[RH02] RAMAMOORTHI R., HANRAHAN P.: Frequency Space Environment Map Rendering. *SIGGRAPH, ACM Transactions on Graphics 21*, 3 (2002), 517–526. 3

[SH05] SIGG C., HADWIGER M.: Fast third-order texture filtering. In *GPU Gems 2*, Pharr M., (Ed.). Addison-Wesley, 2005, pp. 313–329. 4

[Slo08] SLOAN P.: Stupid spherical harmonics (SH) tricks, 2008. 3

[Slo13] SLOAN P.: Efficient spherical harmonic evaluation. *Journal of Computer Graphics Techniques 2*, 2 (2013), 84–83. 8

[WMLT07] WALTER B., MARSCHNER S. R., LI H., TORRANCE K. E.: Microfacet models for refraction through rough surfaces. In *Proceedings of Eurographics Conference on Rendering Techniques* (2007), pp. 195–206. 2, 3