Filtering After Shading With Stochastic Texture Filtering
Matt Pharr, Bartłomiej Wronska, Marco Salvi, Marcos Fajardo
Intro – textures and texture filtering
(The way we traditionally teach it)
Textures – essential for high quality rendering

“Physically Based Rendering: From Theory To Implementation”, 2004-2021 M. Pharr, W. Jakob, and G. Humphreys
Texture mapping – why we need filtering?

- Texture – a 1D/2D/3D/4D grid of discrete values
- Values defined only at texel centers
- “Pixel/texel is not a little square!”
- Infinitely small point – Dirac delta
Texture mapping – why we need filtering?

- Texture – a 1D/2D/3D/4D grid of discrete values
- Values defined only at **texel centers**
- “Pixel/texel is not a little square!”
- Infinitely small point – **Dirac delta**
- What happens between?
Texture mapping – why we need filtering?

- Texture – a 1D/2D/3D/4D grid of discrete values
- Values defined only at **texel centers**
- “Pixel/texel is not a little square!”
- Infinitely small point – **Dirac delta**
- What happens between?
- **Filtering and interpolation**
  - Weighted averaging of multiple texture samples
Texture filtering – minification

- Multiple texels might cover a single pixel area
- Potentially thousands (millions?) texels
Texture filtering – minification

• Multiple texels might cover a single pixel area
• Potentially thousands (millions?) texels

Solution – **full filter** – EWA, anisotropic filtering:
• Possibly very slow
• Hundreds+ of texture samples
• Higher quality
Texture filtering – minification

• Multiple texels might cover a single pixel area
• Potentially thousands (millions?) texels

Solution – **full filter** – EWA, anisotropic filtering:
• Possibly very slow
• Hundreds+ of texture samples
• Higher quality

Solution – **prefilter** – precomputed mipmap pyramid:
• Very fast
• Low quality (blurry!)
(Often – hybrid, mipmapping + anisotropic/mip bias)
“Common knowledge”

Almost “axiomatic”

All modern graphics APIs standardize filtering

• Standard filters – (low-quality) bi/trilinear, anisotropic
Almost “axiomatic”

All modern graphics APIs standardize filtering
• Standard filters – (low-quality) bi/trilinear, anisotropic
• All modern GPUs have dedicated filtering hardware
• Very easy and attractive to use without questioning
Motivation
Project beginning: Stochastic neural texture filtering (performance)

Random-Access Neural Compression of Material Textures, Vaidyanathan et al., Siggraph 2023
Project beginning: Stochastic neural texture filtering (performance)

Random-Access Neural Compression of Material Textures, Vaidyanathan et al., Siggraph 2023
Related Work – Long History of Stochastic Filtering

- Percentage closer filtering
- Original UE software rasterizer: texture-space dithered nearest lookups
- Star Trek, 25th Anniversary: dithered bilinear filtering
- Negative LOD biasing (UE + everyone using TAA/DLSS…)
- OpenImageIO: stochastic LOD selection (via Max Liani)
- Dreamworks MoonRay: nearest sampling for minification, bilerp for magnification
- Interactive Path Tracing and Reconstruction of Sparse Volumes, Hofmann, Hasselgren, Clarberg, and Munkberg, i3d 2021: stochastic trilinear
- Random-Access Neural Compression of Material Textures, Vaidyanathan, Salvi, Wronsiki, Akenine-Möller, Ebelin, Lefohn, SIGGRAPH 2023: stochastic trilinear
Related Work – Long History of Stochastic Filtering

- Percentage closer filtering
- Original UE software rasterizer: texture-space dithered nearest lookups
- Star Trek, 25th Anniversary: dithered bilinear filtering
- Negative LOD biasing (UE + everyone using TAA/DLSS...)
- *OpenImageIO*: stochastic LOD selection (via Max Liani)
- *Dreamworks MoonRay*: nearest sampling for minification, bilerp for magnification
- *Interactive Path Tracing and Reconstruction of Sparse Volumes*, Hofmann, Hasselgren, Clarberg, and Munkberg, i3d 2021: stochastic trilinear

We will generalize those, formalize, and propose two families of techniques beyond simple filters.
Related Work – Long History of Stochastic Filtering

• Percentage closer filtering
• Original UE software rasterizer: texture-space dithered nearest lookups
• Star Trek, 25th Anniversary: dithered bilinear filtering
• Negative LOD biasing (UE + everyone using TAA/DLSS...)
• OpenImageIO: stochastic LOD selection (via Max Liani)
• Dreamworks MoonRay: nearest sampling for minification, bilerp for magnification
• Interactive Path Tracing and Reconstruction of Sparse Volumes, Hofmann, Hasselgren, Clarberg, and Munkberg, i3d 2021: stochastic trilinear
• Random-Access Neural Compression of Material Textures, Vaidyanathan, Salvi, Wronski, Akenine-Möller, Ebelin, Lefohn, SIGGRAPH 2023: stochastic trilinear

We will generalize those, formalize, and propose two families of techniques beyond simple filters

But we have a more important problem to solve first...
Stochastic texture filtering: do we have a problem?

Filtering this way can look significantly different...

Random-Access Neural Compression of Material Textures, Vaidyanathan et al., Siggraph 2023
But which way is “correct”?

Bold question: Are we teaching and using texture filtering “incorrectly”?!
Literature review and historical precedents
Precedent: Pre-multiplied alpha

$$\text{filt}(\text{target}) \times \text{filt}(\alpha) + (1 - \text{filt}(\alpha)) \times \text{source}$$

Figure credit: “premultiplied alpha – 2022”, Inigo Quilez
Precedent: Pre-multiplied alpha

"Compositing digital images", Thomas Porter and Tom Duff., SIGGRAPH 1984. Figure credit: “premultiplied alpha – 2022”, Inigo Quilez
Precedent: Percentage Closer Shadow Filtering

\[ \text{visibility} = z < \int \text{depth}(u, v) \, du \, dv \]

Precendent: Percentage Closer Shadow Filtering

\[ \text{visibility} = z < \int \text{depth}(u, v) \, du \, dv \]

\[ \text{visibility} = \int f(u, v) (z < \text{depth}(u, v)) \, du \, dv \]

Jitter \((u,v)\) to sample \(f\)

Rendering Antialiased Shadows With Depth Maps, Reeves et al., SIGGRAPH 1987.
Finally, we hope to be able to generalize and formalize the sample transformation step in percentage closer filtering. We believe that this technique may have important implications to the use of texture maps for other purposes. For example, in bump mapping [Bli78], specular reflections could be computed before filtering, and the results could be filtered and sampled as ordinary textures. In this way, specular highlights from the microfacets of a bumpy surface would be maintained even as the surface were translated back into the far distance.

Precedent: Specular anti-aliasing (minification)

Mipmapping Normal Maps, Michael Toksvig, 2006
LEAN Mapping, Marc Olano and Dan Baker, I3D 2011
Figure credit: Spectacular Specular: LEAN and CLEAN Specular Highlights, Dan Baker, GDC 2011
Still a problem: Metalness vs specular PBR workflow

Specular reflectance = lerp(0.04, filt(color), filt(metalness))
Diffuse reflectance = lerp(filt(color), 0.0, 1-filt(metalness))

Figure credit: *Metallic magic*, Daniel Rose
Motivation - summary

- Texture filtering theory and practice were developed for interpolating just “color”
- ...in early work, not even gamma-corrected!
Motivation - summary

- Texture filtering theory and practice were developed for interpolating just “color”
- ...in early work, not even gamma-corrected!
- Filtering and affine functions commute perfectly – this approach didn’t introduce errors
- **Non-linearity and filtering do not commute and swapping the order results in error**
Motivation - summary

- **Assumption**: textures are authored by artists with ~1-1 pixel-texel ratio
Motivation - summary

• **Assumption**: textures are authored by artists with ~1-1 pixel-texel ratio
• Minifying or magnifying textures before (non-linear) shading **introduces error/bias**
• Different techniques proposed to address specific types of errors
Motivation - summary

• Can we do better in general?
• Let’s try to answer this question from 37y ago!

Finally, we hope to be able to generalize and formalize the sample transformation step in percentage closer filtering. We believe that this technique may have important implications to the use of texture maps for other purposes. For example, in bump mapping [Bli78], specular reflections could be computed before filtering, and the results could be filtered and sampled as ordinary textures. In this way, specular highlights from the microfacets of a bumpy surface would be maintained even as the surface were translated back into the far distance.

Textures and the Rendering Equation
Filtering Before Shading  (Standard Practice Today)

\[ L_0(p, \omega_o) = \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) |\cos \theta_i| \, d\omega_i \]
Filtering Before Shading (Standard Practice Today)

\[ L_o(p, \omega_o) = \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) |\cos \theta_i| \, d\omega_i \]

\[ \text{reflectance} = \int f(u, v) \, \text{tex}_{\text{reflectance}}(u, v) \, du \, dv \]

\[ \text{normal} = \int f(u, v) \, \text{tex}_{\text{normalmap}}(u, v) \, du \, dv \]

HW 16x Aniso  
Reference
Filtering After Shading

\[ L_o(p, \omega_o) = \int f(u, v) \left( \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) |\cos \theta_i| d\omega_i \right) du dv \]
Filtering After Shading

\[ L_o(p, \omega_o) = \int f(u, v) \left( \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) |\cos \theta_i| d\omega_i \right) du dv \]
Filtering After Shading

\[ L_o(p, \omega_o) = \int f(u, v) \left( \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) \cos \theta_i \, d\omega_i \right) \, du \, dv \]

- Use Monte Carlo!
- Sample \((u', v') \sim f(u, v)\)
Filtering After Shading

\[ L_o(p, \omega_o) = \int f(u, v) \left( \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) |\cos \theta_i| \, d\omega_i \right) \, du \, dv \]

- Use Monte Carlo!
- Sample \((u', v') \sim f(u, v)\)
- Estimator: 
  \[ L_o(p, \omega_o) \approx \frac{f(u, v)}{p(u, v)} \left( \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) |\cos \theta_i| \, d\omega_i \right) \, du \, dv \]
  \[ = \int_{S^2} f_r(\omega_i, \omega_o) L_i(p, \omega_i) |\cos \theta_i| \, d\omega_i \]

**reflectance** = \( tex_{\text{reflectance}}(u', v') \)  
**normal** = \( tex_{\text{normalmap}}(u', v') \)

Unfiltered single texel lookups!
Even Single Sample!
Real Time Rendering – Noise

White noise

Single frame
Even Single Sample!
Real Time Rendering – Noise

- DLSS as the robust temporal integrator
- Spatiotemporal Blue Noise reduces the noise appearance
- Makes DLSS job easier, together -> no visible noise in most cases

Single frame

![White noise](image1)
![Spatiotemporal Blue Noise](image2)

DLSS
Two families of methods
Sampling Texture Filters – Discrete, 1D

\[ \text{lookup}(x) = \int f(u - x) t(u) \, du = \sum_{u=-1}^{2} f(u - x) t(u) \]

Chose a sample with probability \( \sim f \)
Filter Reservoir Sampling

- **Importance sampling**: Sample a texel with probability $p \sim f$
- Sample an array of weights or online through weighted reservoir sampling
- Details in the paper
Sampling Texture Filters
Disadvantages of Filter Reservoir Sampling

• Discrete filter sampling – with large filters, can be costly
• Evaluate filter function $K^M$ or $K*M$ times
• Does not support infinite filters (Gaussian, sinc)
Sampling Texture Filters
Disadvantages of Filter Reservoir Sampling

- Discrete filter sampling – with large filters, can be costly
- Evaluate filter function $K^M$ or $K \times M$ times
- Does not support infinite filters (Gaussian, sinc)
- There’s a different way!
- Let’s analyze and understand the “UV jitter + nearest neighbor” prior work.
Magnification

What happens when you take a nearest-neighbor sample?

Nearest neighbor = box filter
Magnification
Filter Importance Sampling through UV jittering

Uniform UV jitter + nearest neighbor = ?
Magnification
Filter Importance Sampling through UV jittering

Uniform UV jitter + nearest neighbor = tent kernel!
The same as linear interpolation

Jitter PDF \times \text{Box Kernel Convolution}
Magnification
Filter Importance Sampling through UV jittering

Linear/tent UV jitter + nearest neighbor box = quadratic B-Spline
Quadratic UV jitter + nearest neighbor box = \textbf{cubic B-Spline}
Magnification
Filter Importance Sampling through UV jittering

• For B-Spline filters, this additional box is desirable!
• Can sample other, including infinite spatial support filters
• Jitter UVs according to PDF deconvolved with a box
Magnification
Filter Importance Sampling through UV jittering

- For B-Spline filters, this additional box is desirable!
- Can sample other, including infinite spatial support filters
- Jitter UVs according to PDF deconvolved with a box
- For many other filters -> can still be advantageous (prevent Gaussian under-sampling)

\[
\begin{align*}
\sigma = 0.3 & \quad & \sigma = 0.5 & \quad & \sigma = 0.8 \\
\text{Discrete} \ 4 \times 4 & \quad & \text{Discrete} \ 4 \times 4 & \quad & \text{Discrete} \ 4 \times 4 \\
\text{FIS} & \quad & \text{FIS} & \quad & \text{FIS}
\end{align*}
\]
Stochastic Filtering families compared

- Main difference: discrete vs continuous domain
- In some cases, FRS is the only option (arbitrary discrete kernels, positivization)
- Otherwise, we recommend FIS – simpler implementation, see provided source code
Minification
Stochastic Filtering After Shading

- Anisotropic filtering or elliptically weighted average
- Many pixels, non-uniform mapping for jittering
- There's a simpler, **already-used** method!
Minification
Stochastic Filtering After Shading

- Common practice – jitter the projection matrix for anti-aliasing reconstruction filter
- Used offline (e.g., MoonRay) and real-time (TAA, DLSS)
Minification
Stochastic Filtering After Shading

• Common practice – jitter the projection matrix for anti-aliasing reconstruction filter
• Used offline (e.g., MoonRay) and real-time (TAA, DLSS)
• Projects to trapezoid, minification supersampling -> filtering after shading!
Minification
Stochastic Filtering After Shading

• Common practice – jitter the projection matrix for anti-aliasing reconstruction filter
• Used offline (e.g., MoonRay) and real-time (TAA, DLSS)
• Projects to trapezoid, minification supersampling -> filtering after shading!
• Add magnification/translation UV jitter -> **unified minification and magnification**
Appearance change and possible aliasing
Magnification specular appearance change

Filtering before shading

Filtering after shading
Appearance change explained

Filtering before shading:
Interpolated surface and normals
Appearance change explained

Filtering before shading:
Interpolated surface and normals

Filtering after shading:
Two adjacent geometric facets and normals
Filtering lighting does not produce surface curvature

Filtering before shading

Filtering after shading

Note: it’s neither “good” or “bad”, depends on the intent / assumption
But it changes the appearance – artists need to be aware!
Worse example – magnification aliasing

Filtering before shading

Filtering after shading
Non-linearity introduced aliasing

- Any non-linearity always introduces new, higher signal frequencies ("harmonics")
- When applied to discrete signals... **those frequencies alias immediately**
Non-linearity introduced aliasing

• Any non-linearity always introduces new, higher signal frequencies (“harmonics”)
• When applied to discrete signals... **those frequencies alias immediately**
• **Magnification**: screen Nyquist higher than texture Nyquist
• For formal analysis, [see the paper supplement](#)
Results
Appearance preservation – real time

Hybrid: Use a mipmap, but higher resolution (reduce cache trashing, make it easier for DLSS)
Appearance Preservation – offline, volumetric textures

(a) Trilinear
(b) Stochastic trilinear
(c) Trilinear, MIP mapped
(d) Stochastic minification
Offline - Improved Image Quality & Performance
No additional noise!

Figure 1: A section of the Disney Cloud scene rendered with path tracing. With this close-in viewpoint, trilinear filtering leads to blocky artifacts in the image. Tricubic filtering gives a much better result, though requires 64 voxel lookups into the NanoVDB representation. Stochastic filtering performs a single voxel lookup yet provides indistinguishable results, with overall rendering time speedups of 1.60× and 2.77× for the trilinear and tricubic filters. Times reported are for pbtr-v4 running on an NVIDIA 4090 RTX GPU, rendering at 1080p with 256 samples per pixel.
Minification & Magnification
DLSS + STBN Temporal Stability Test

HW Filtering
Max Aniso 16
1 spp + DLSS

Stochastic Bicubic
Max Aniso 64
1 spp + DLSS
Stress Test: Real-Time Stochastic Filtering, high contrast, no mip-maps


Residual noise, but... temporal filters improve every year. ML approaches can be trained on it.
Recommendations

• **Minification**: Filtering After Shading is *always better*
• **Minification**: Offline rendering can remove mip-maps: rendering Monte Carlo noise dominates
• **Minification**: Real-time rendering: “hybrid” (performance, temporal stability but some bias remains)
• **Minification**: Filtering After Shading is *always better*
• **Minification**: Offline rendering can remove mip-maps: rendering Monte Carlo noise dominates
• **Minification**: Real-time rendering: “hybrid” (performance, temporal stability but some bias remains)

• **Magnification**: Filtering After Shading is *unbiased and removes errors*
• **Magnification**: Filtering After Shading *simplifies logic* (alpha, metalness, texture padding)
• **Magnification**: Filtering After Shading *allows for better filters and new texture representations*
Recommendations

- **Minification**: Filtering After Shading is **always better**
- **Minification**: Offline rendering can remove mip-maps: rendering Monte Carlo noise dominates
- **Minification**: Real-time rendering: “hybrid” (performance, temporal stability but some bias remains)

- **Magnification**: Filtering After Shading is **unbiased and removes errors**
- **Magnification**: Filtering After Shading **simplifies logic** (alpha, metalness, texture padding)
- **Magnification**: Filtering After Shading **allows for better filters and new texture representations**
- **Magnification**: Filtering After Shading **can introduce aliasing**
- **Magnification**: It depends! Decide based on use-case, content type, maximum magnification
Recommendations

• You don’t have to go “all in”, we recommend a pragmatic approach:
  • There are trade-offs and cases where one is preferred over the other
  • Don’t stochastically sample something that relies on interpolation (e.g., SDF fonts)
  • Use STF/non-STF/different filters on different assets – only shader code changes!
Conclusions

• Our proposal of “filtering after shading” might seem radical...
• We simply formalize decades of the different film industry and gamedev practices!
  • Filtering after shading is unbiased and better for appearance preservation
  • We need to change the way we teach filtering and blending
Conclusions

- Our proposal of “filtering after shading” might seem radical...
- We simply formalize decades of the different film industry and gamedev practices!
  - Filtering after shading is unbiased and better for appearance preservation
  - We need to change the way we teach filtering and blending

- **Stochastic texture filtering** present for ~40y in literature in various one-off flavors
  - We explain the prior approaches and generalize them
  - We propose two families of techniques with different trade-offs
Conclusions

• Our proposal of “filtering after shading” might seem radical...
• We simply formalize decades of the different film industry and gamedev practices!
  • Filtering after shading is unbiased and better for appearance preservation
  • We need to change the way we teach filtering and blending

• Stochastic texture filtering present for ~40y in literature in various one-off flavors
  • We explain the prior approaches and generalize them
  • We propose two families of techniques with different trade-offs
  • We expand those to more filters, including negative lobe filters
  • Source code of efficient implementations – drop-in, zero integration cost!
Summary

- Filtering after shading by stochastic texture filtering is a valuable tool:
  - Remove workarounds and simplify code (alpha, specular AA, virtual texture padding)
  - Enables efficient filtering of novel compression and storage formats – neural, octrees, DCT, ...?
Summary

• Filtering after shading by stochastic texture filtering is a valuable tool:
  • Remove workarounds and simplify code (alpha, specular AA, virtual texture padding)
  • Enables efficient filtering of novel compression and storage formats – neural, octrees, DCT, ...?
  • Efficient better filters (isotropic, smooth, negative weights) – let’s get rid of bilinear!
Summary

• Filtering after shading by stochastic texture filtering is a valuable tool:
  • Remove workarounds and simplify code (alpha, specular AA, virtual texture padding)
  • Enables efficient filtering of novel compression and storage formats – neural, octrees, DCT, ...?
  • Efficient better filters (isotropic, smooth, negative weights) – let’s get rid of bilinear!
  • Beyond textures: optimize and stochastically sample complex shader graphs

• Intrigued? Disagree? Outraged? Let’s chat! 😊
Thank you!
Questions?
Enable Custom Texture Compression/Storage Algorithms

Neural textures
3.6 MB
15x more texels
stochastic Gaussian filtering
1.15 ms @ 4K

BCx textures
8.3 MB
lower resolution
trilinear filtering
0.49 ms @ 4K
Minification vs Magnification jitter

Minification

Screen X

Magnification

Screen X

Texel

Pixel

UV jitter

XY jitter
Bonus: unexpected consequence

- Something that bothered me for many years...
- We always recommend decoding to linear before generating mip-maps (minification)...
- But why upsampling/sharpening looks way better applied in sRGB/gamma space?
- **Gamma conversion in either direction - introduces aliasing!**
- Doing/undoing gamma correction: Alias -> upsample -> Alias

Figure credit: *A Fresh Look at Generalized Sampling*, Diego Nehab and Hugues Hoppe
Sampling Texture Filters – Negative Lobes

- Image Processing uses almost exclusively negative lobe filters
- Approximations of a “perfect” interpolation filter
- Sharp, anti-aliased

B-Spline Bicubic  Mitchell  Lanczos3
Sampling Texture Filters – Negative Lobes

- Image Processing uses almost exclusively negative lobe filters
- Approximations of a “perfect” interpolation filter
- Sharp, anti-aliased
- Examples: Sinc, Lanczos, Mitchell...
Sampling Texture Filters – Negative Lobes

• Sample proportionally to abs(f) -> works, but...
• Generates negative values
• Very high variance and noise
Sampling Texture Filters – Negative Lobes

• Sample proportionally to abs(f) -> works, but...
• Generates negative values
• Very high variance and noise

Solution – positivization
• Importance sample the positive and negative parts separately
• Always evaluate two samples
• Weight sum always positive
• 2X the cost
• Low variance

$$\langle F \rangle = \sum_{i} w_i^+ t_{j^+} - \sum_{i} w_i^- t_{j^-}.$$
Bilinear

Mitchell

Positivization – Results