Filtering After Shading With Stochastic Texture Filtering

Matt Pharr, Bartlomiej Wronski, Marco Salvi, Marcos Fajardo

NVIDIA

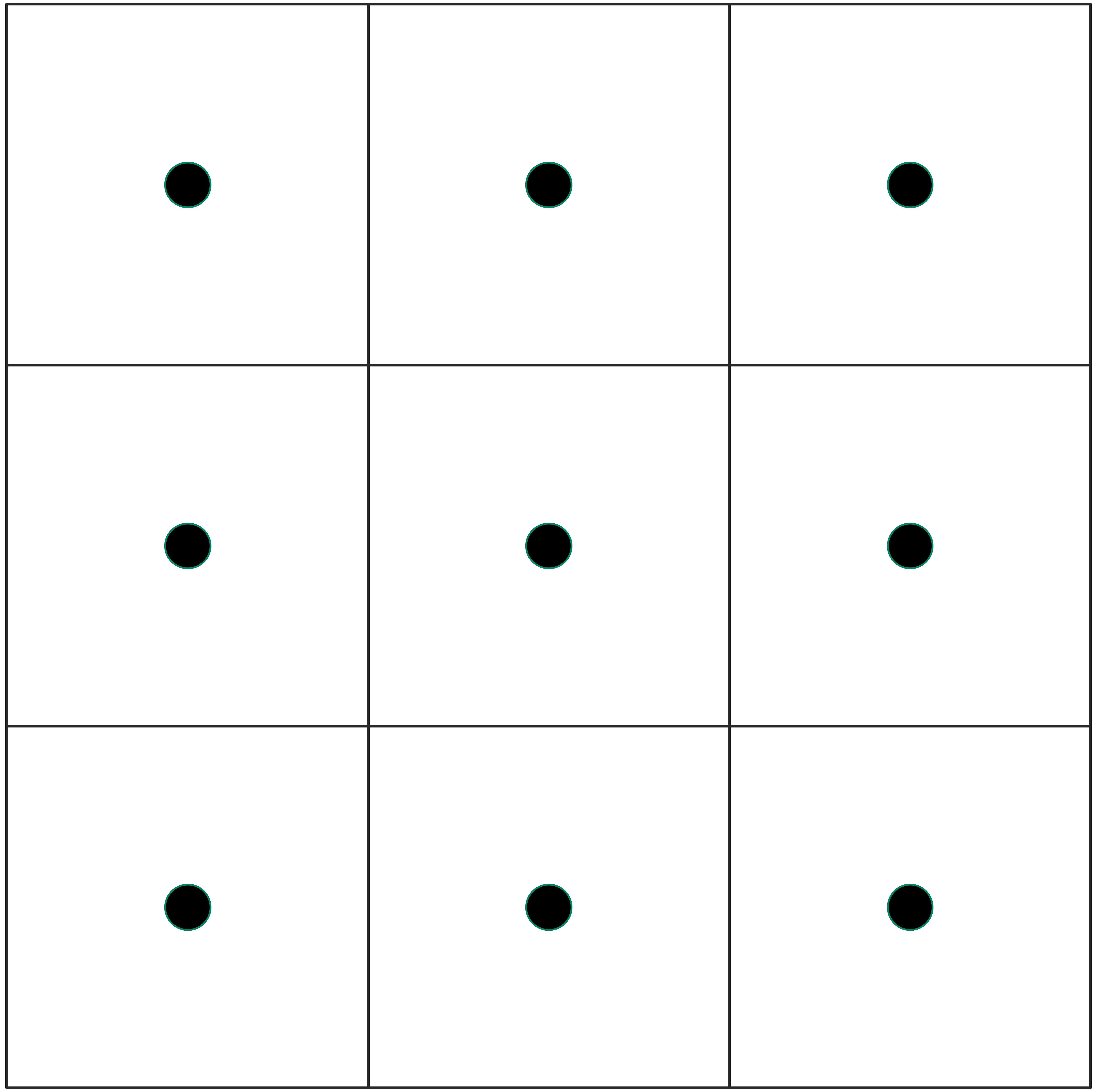
Intro – textures and texture filtering
(The way we traditionally teach it)

Textures – essential for high quality rendering

"Physically Based Rendering: From Theory To Implementation", 2004-2021 M. Pharr, W. Jakob, and G. Humphreys

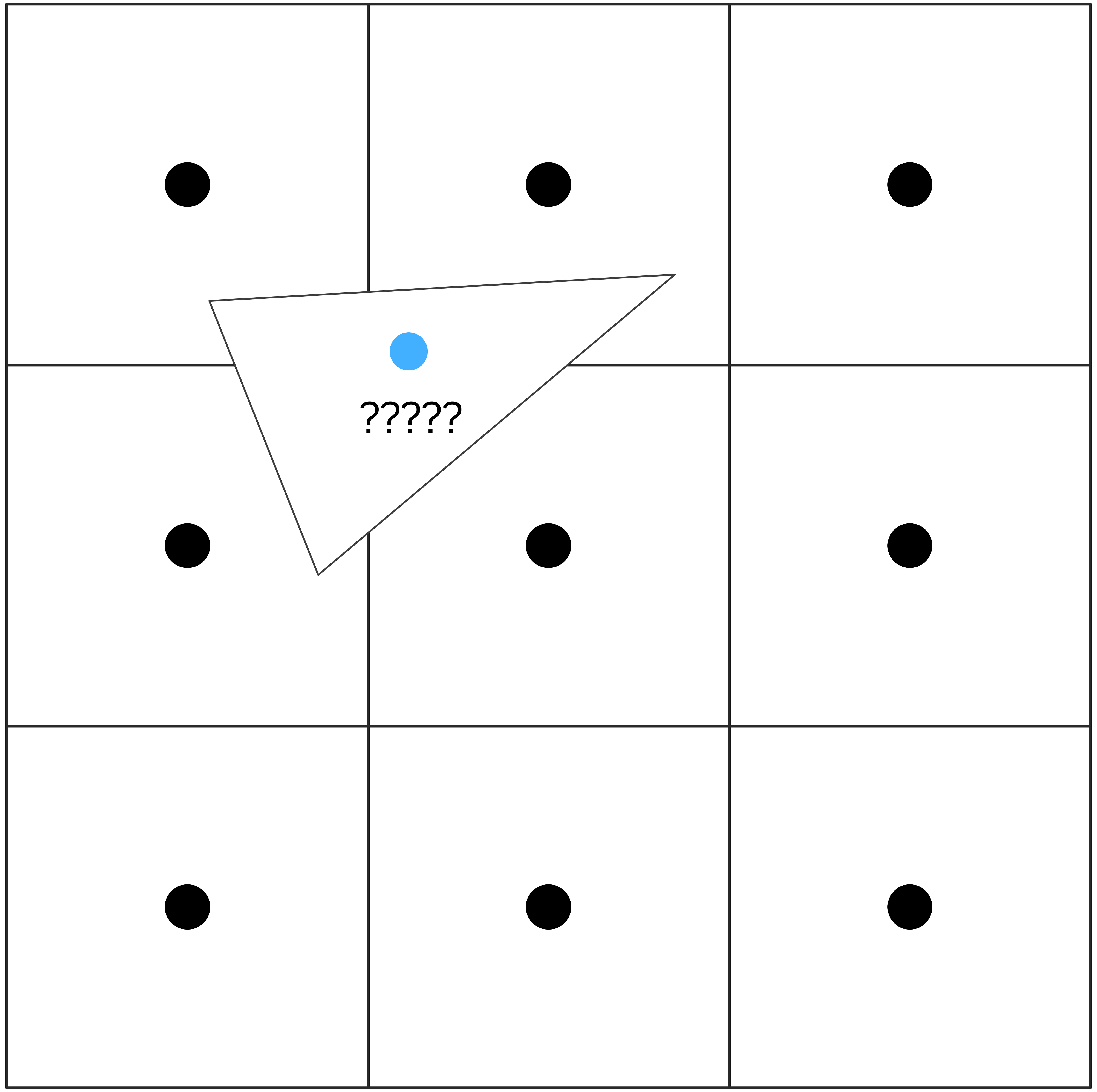
Texture mapping – why we need filtering?

- 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**



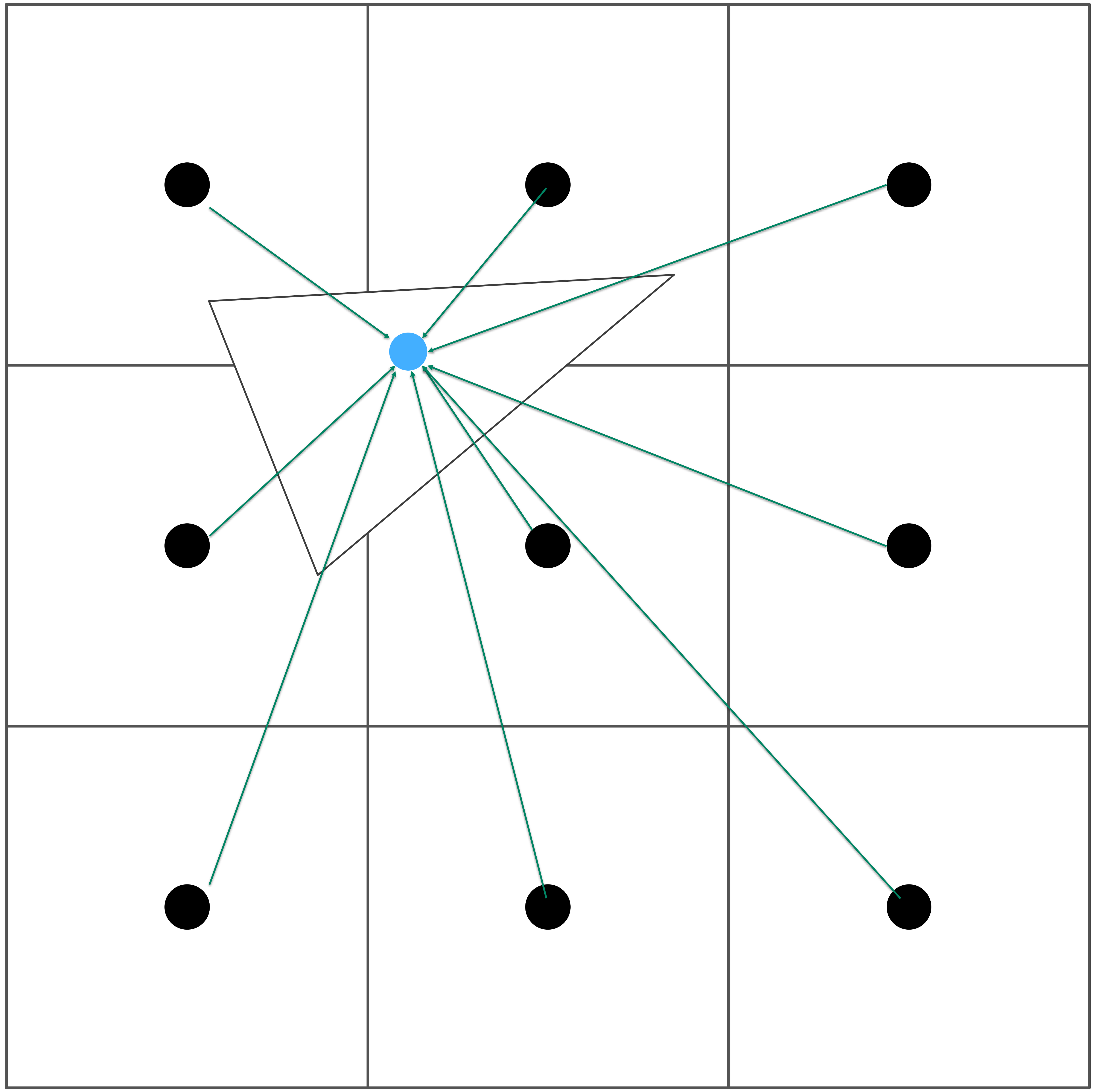
Texture mapping – why we need filtering?

- 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?



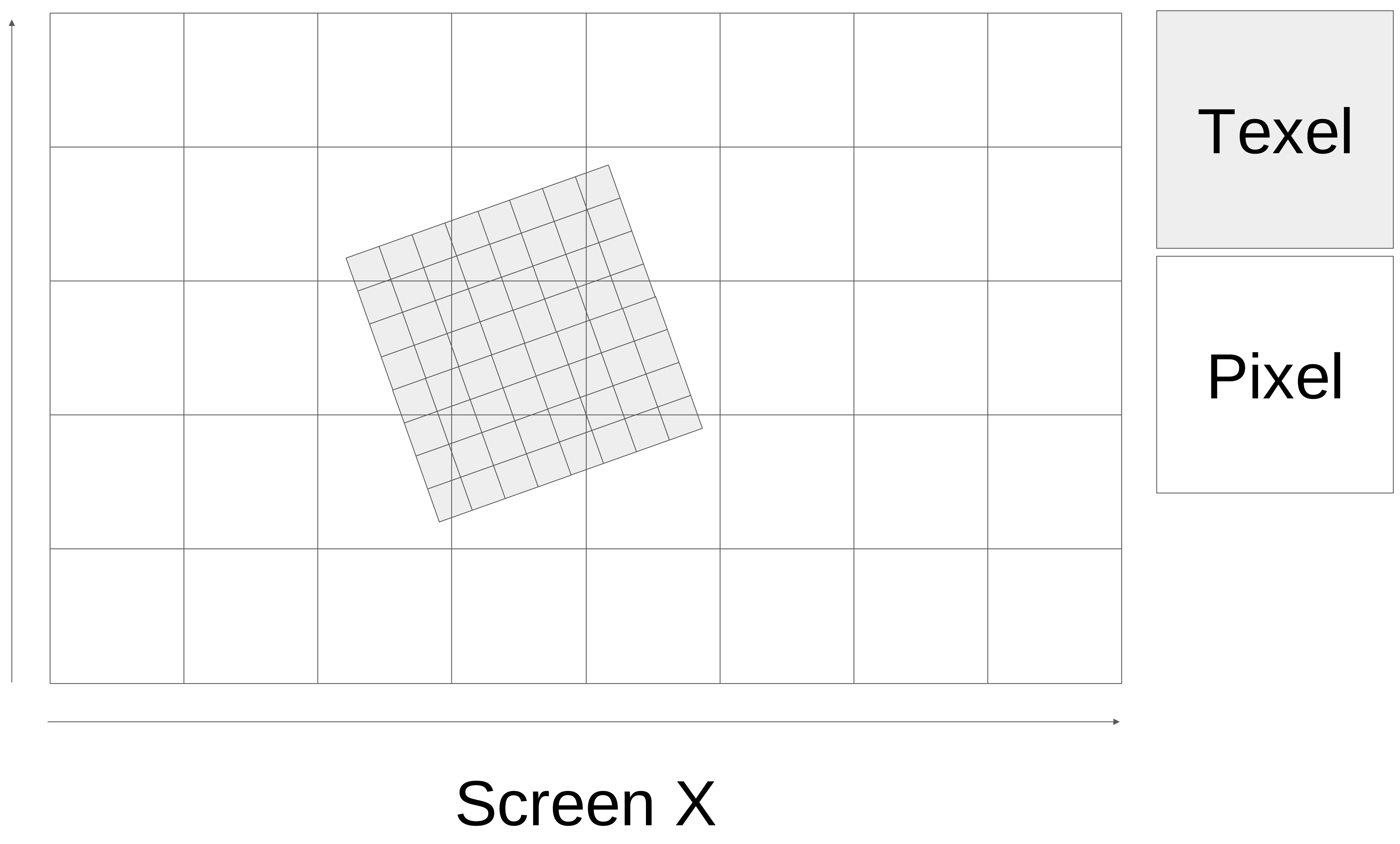
Texture mapping – why we need filtering?

- 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

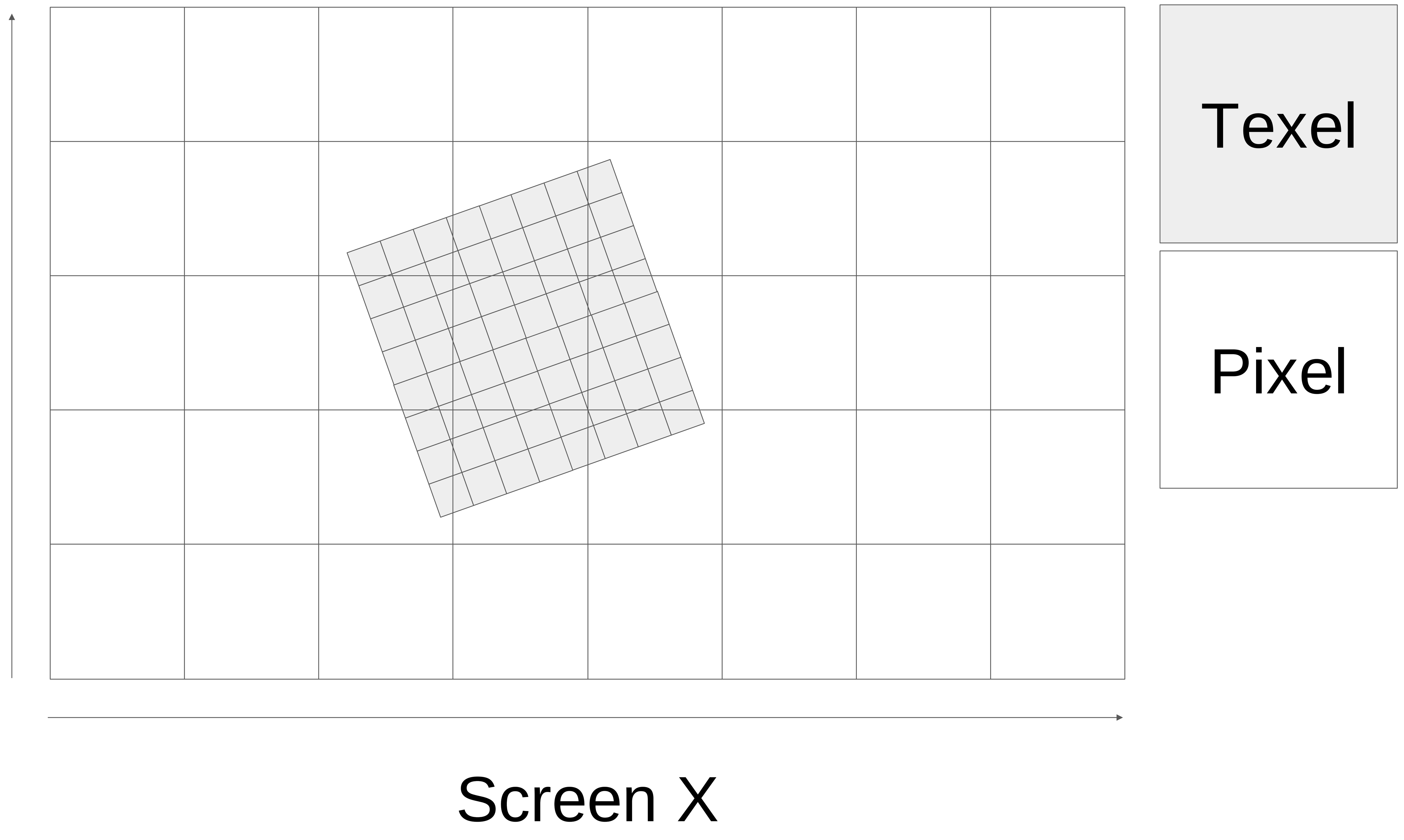


Texture filtering – minification

- 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



NVIDIA

Texture filtering – minification

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

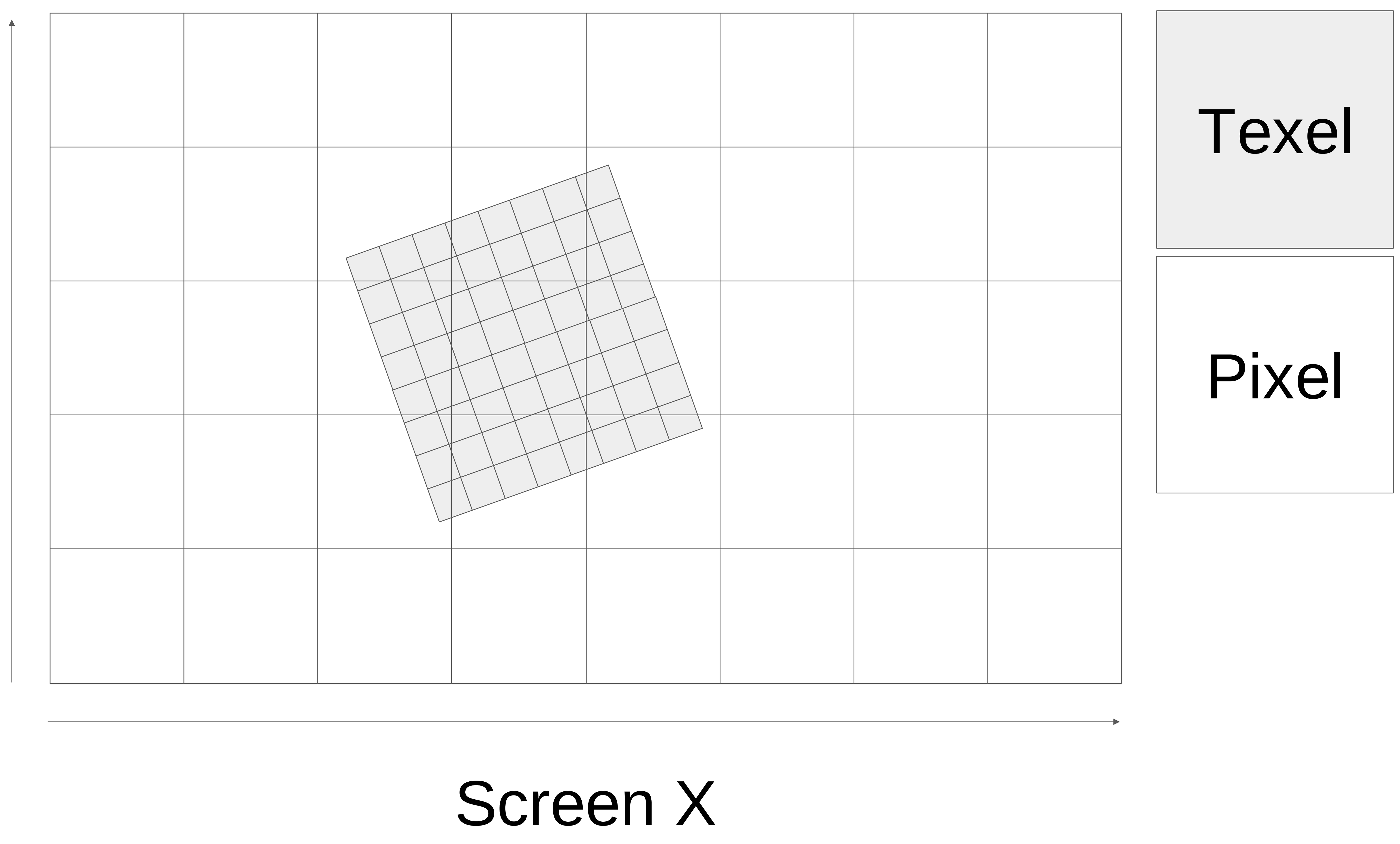
Solution – **full filter** – EWA, anisotropic filtering:

- 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

“Common knowledge”

Almost “axiomatic”

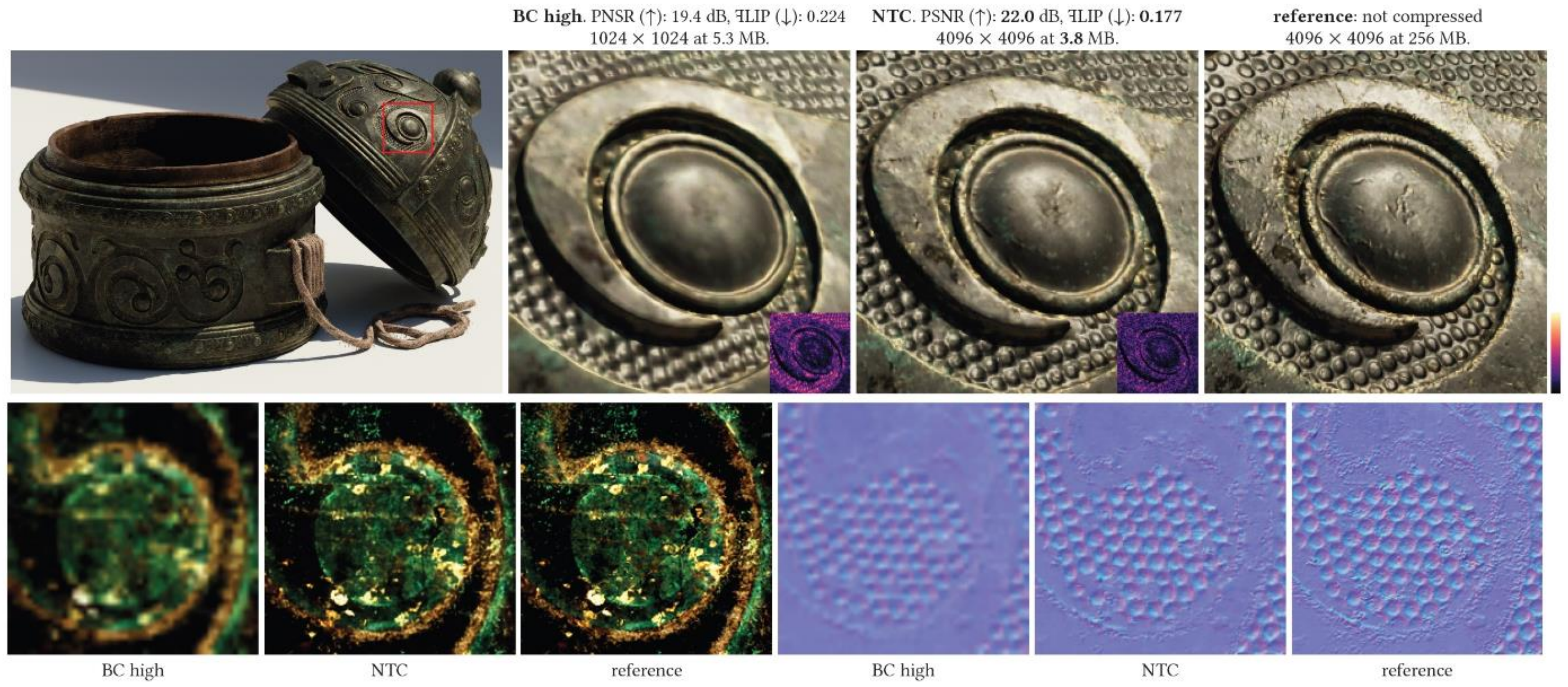
All modern graphics APIs standardize filtering

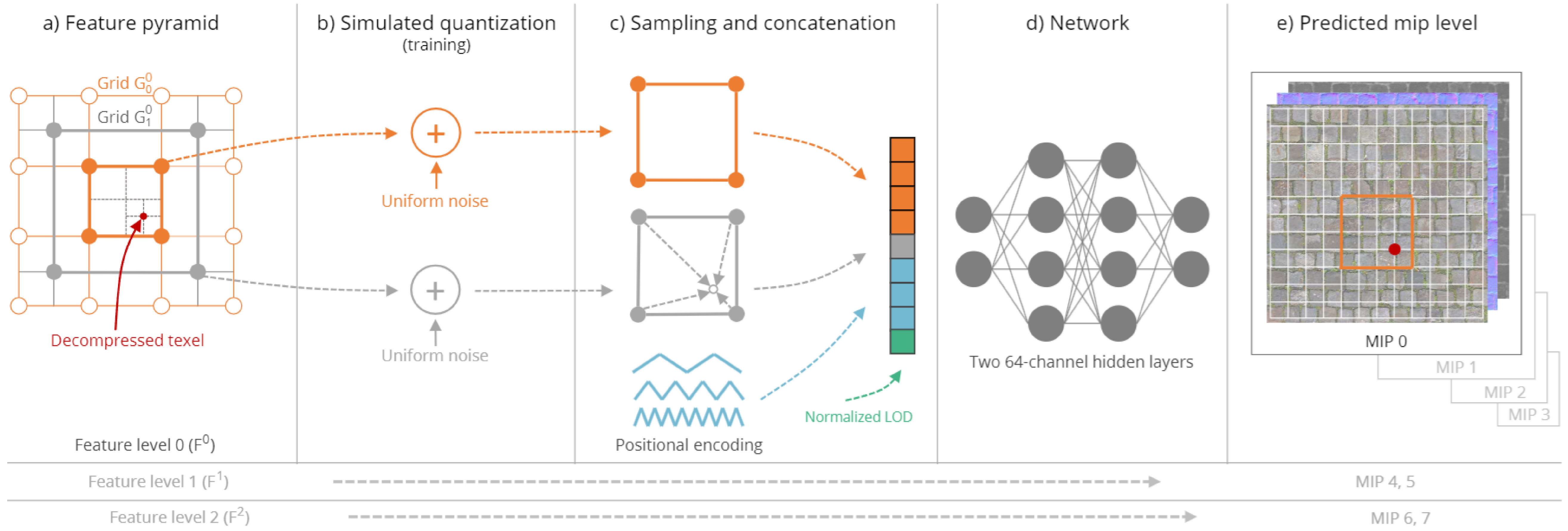
- Standard filters – (low-quality) bi/trilinear, anisotropic
- All modern GPUs have dedicated filtering hardware
- Very easy and attractive to use without questioning

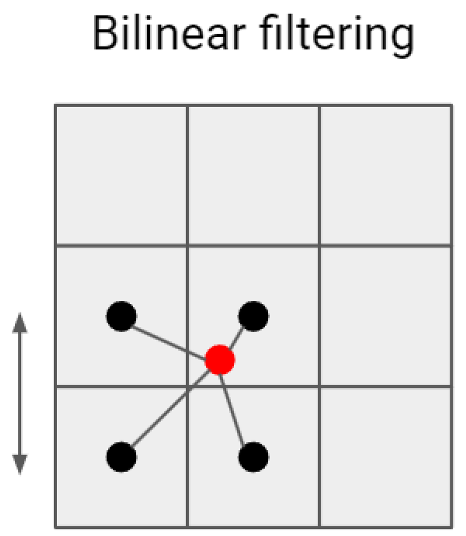
Feedback loop!

Motivation

Project beginning : Stochastic neural texture filtering (performance)

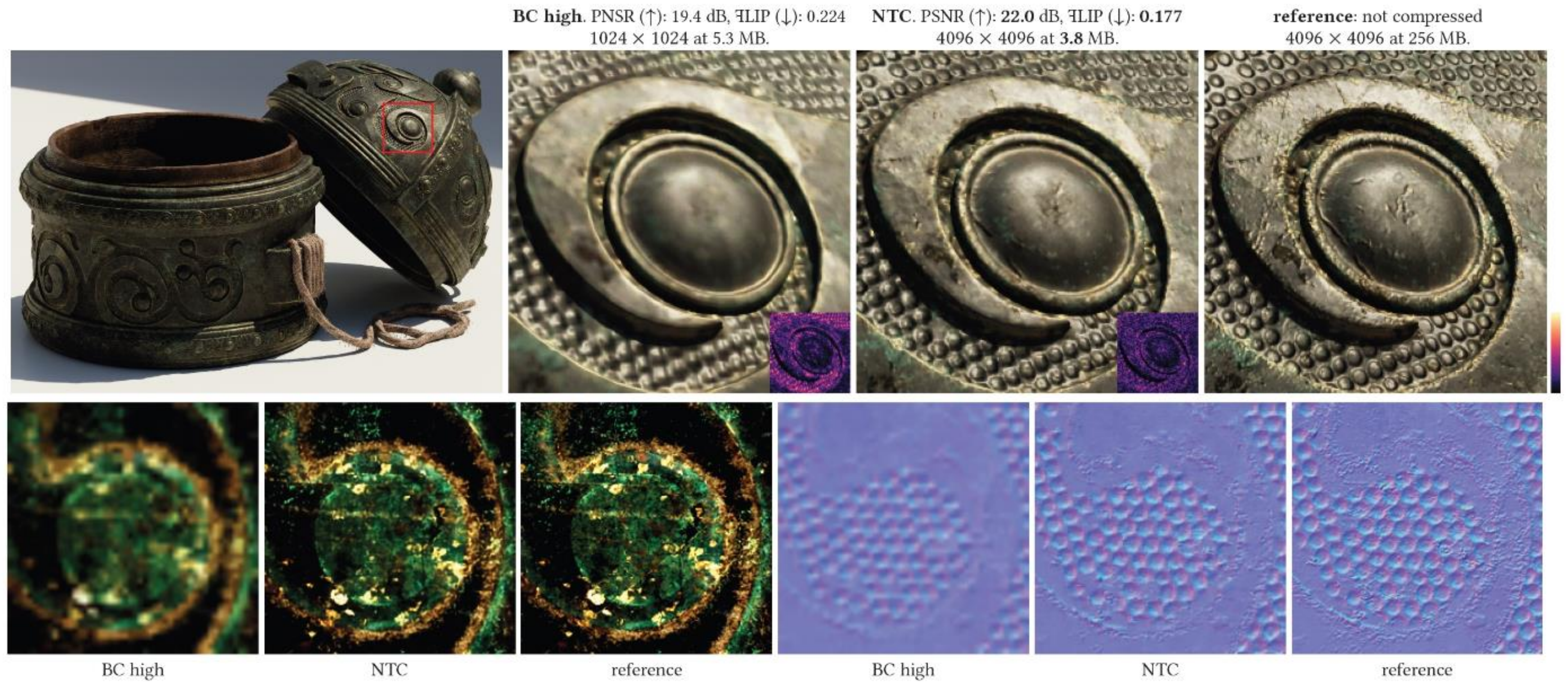


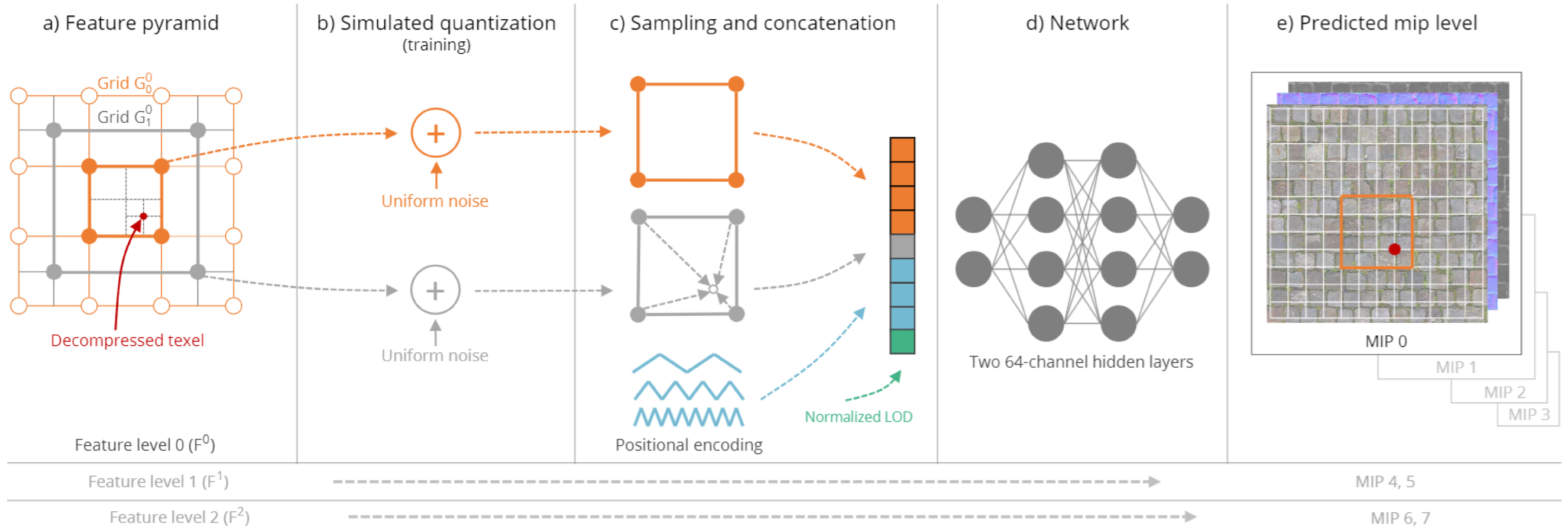


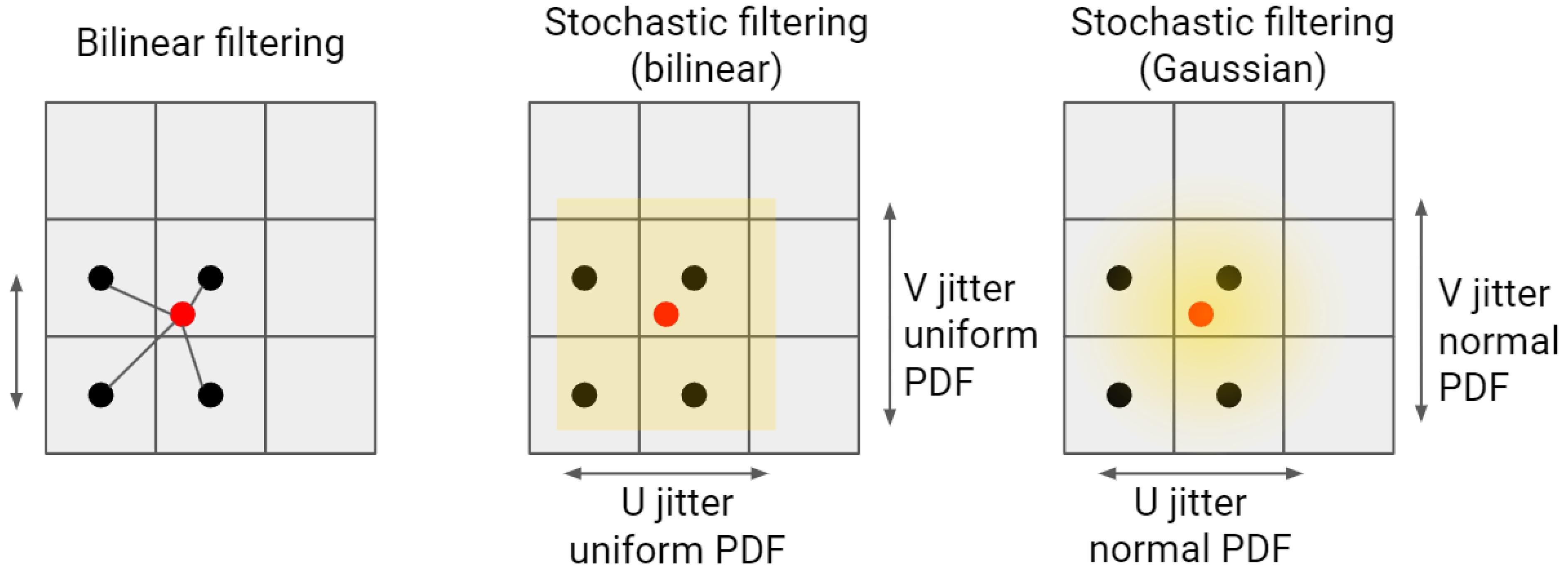


Random-Access Neural Compression of Material Textures, Vaidyanathan et al., Siggraph 2023

Project beginning : Stochastic neural texture filtering (performance)



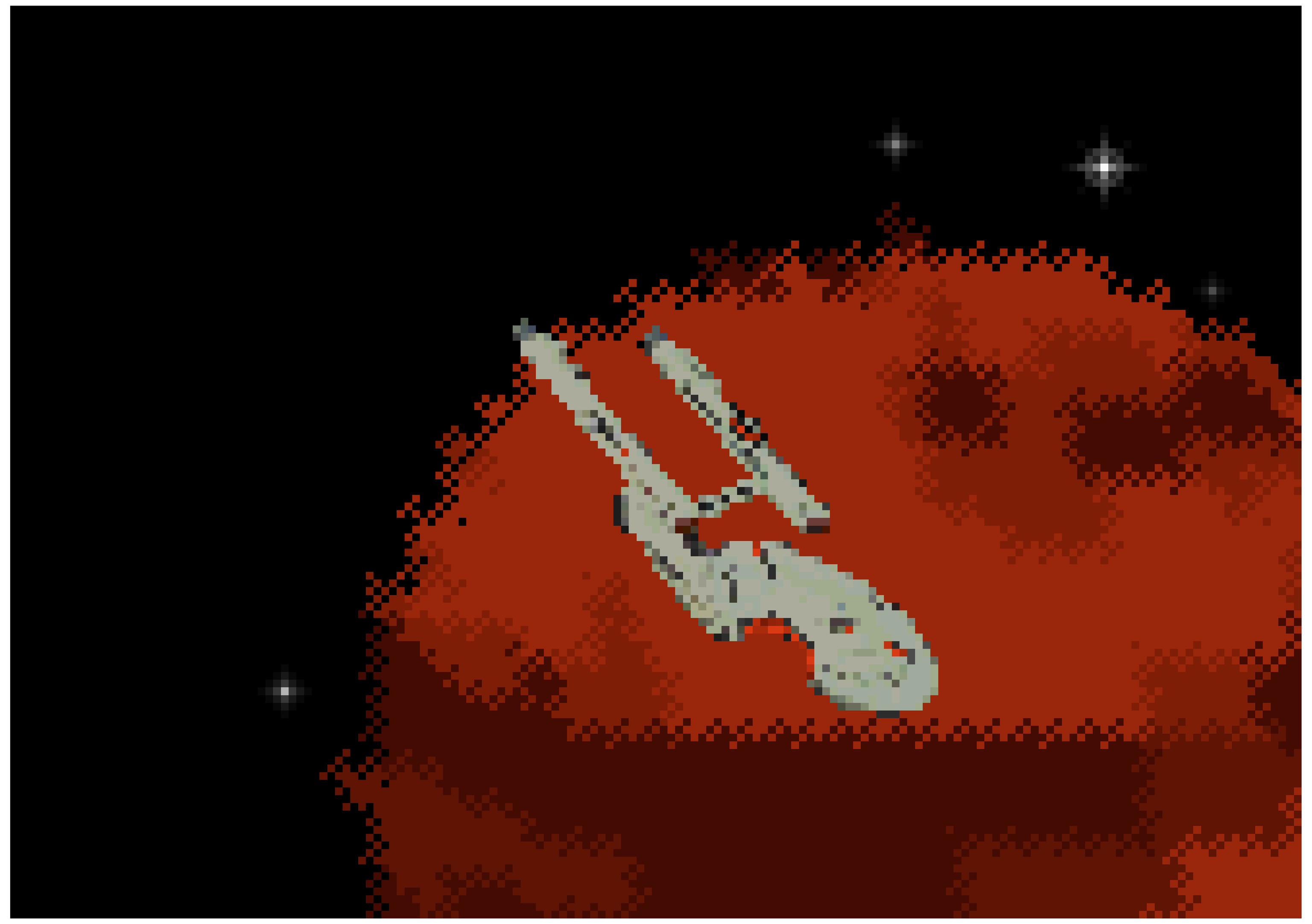




Random-Access Neural Compression of Material Textures, Vaidyanathan et al., Siggraph 2023

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, Wronski, 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...)
- *OpenIma*
- Dreamw
  magnific
- *Interacti*
  Hofmanı
  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

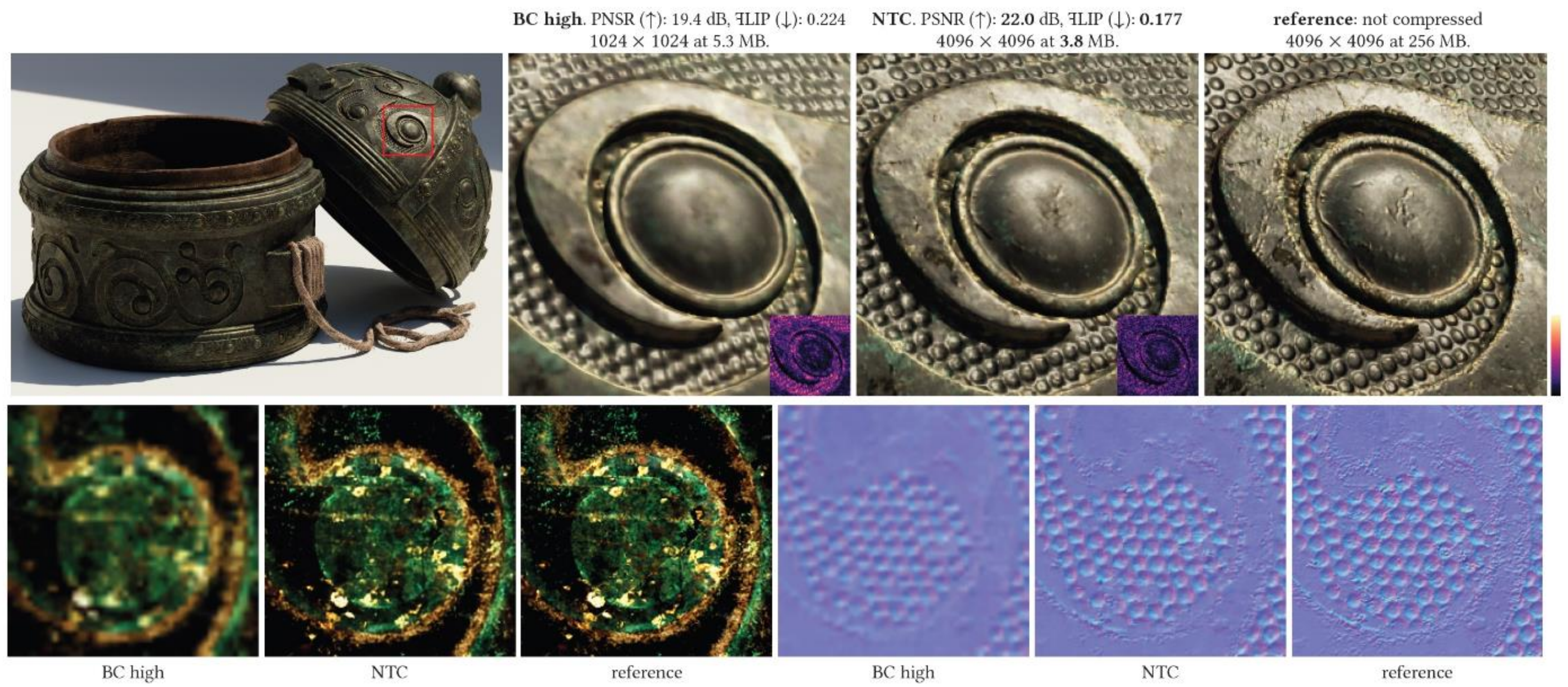
Related Work – Long History of Stochastic Filtering

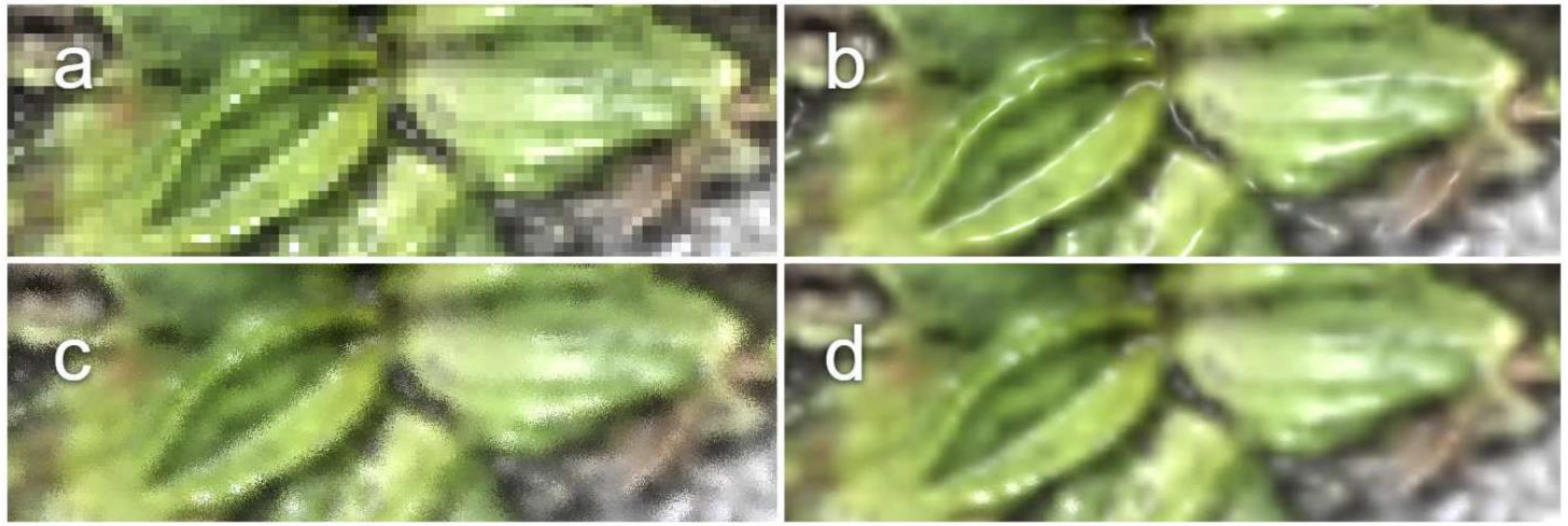
- Percentage closer filtering
- Original UE software rasterizer: texture-space dithered nearest lookups
- Star Trek, 25th Anniversary: dithered bilinear filtering
- Negative LOD biasing (UE + everyone using TAA/DLSS…)
- *OpenIma*
- Dreamw
magnific
- *Interacti*
Hofman
trilinear
- *Random-Access Neural 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…

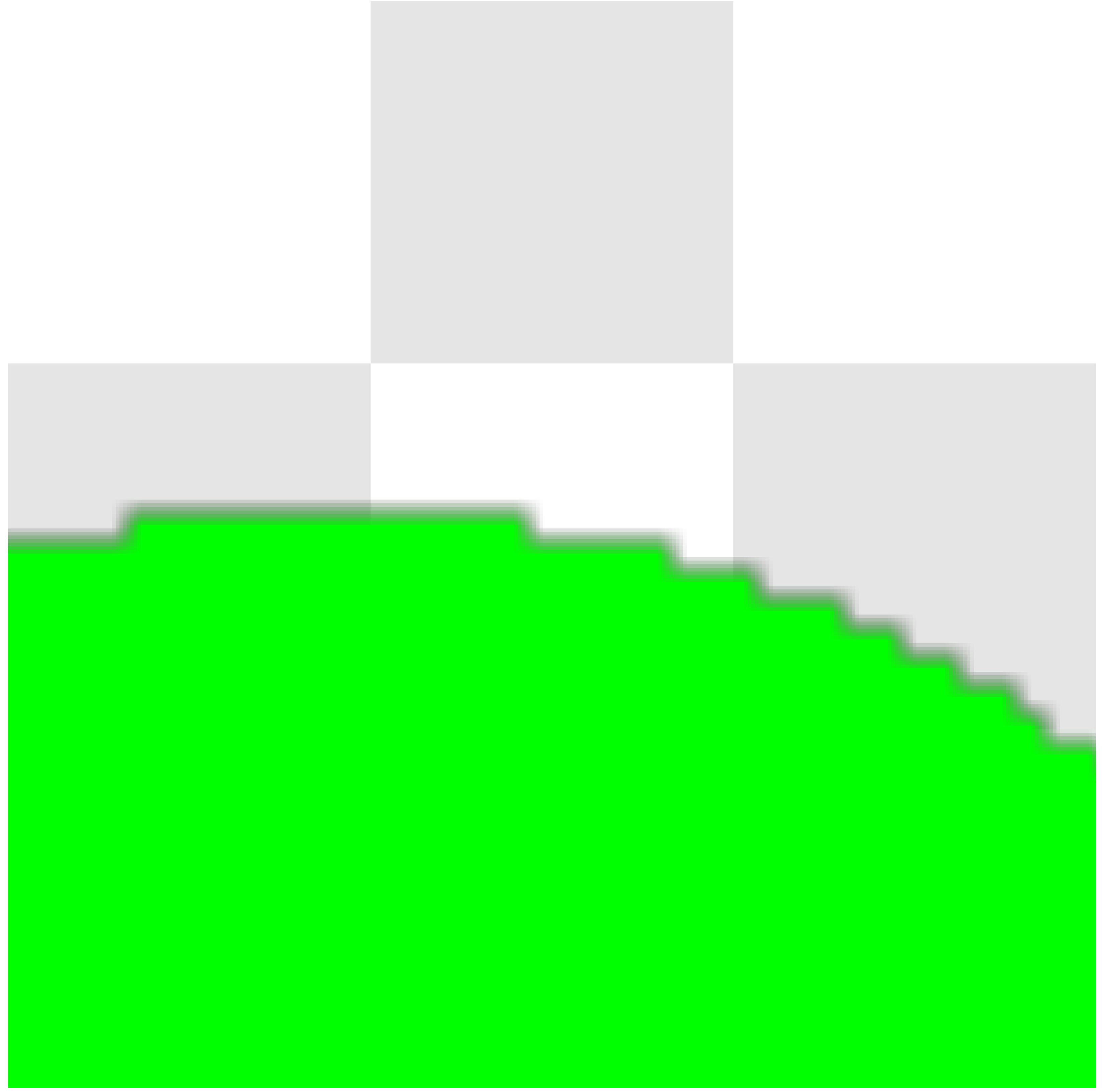
Random-Access Neural Compression of Material Textures, Vaidyanathan et al., Siggraph 2023

But which way is “correct”?

Bold question:
Are we teaching and using texture filtering “incorrectly”?!

Literature review and historical precedents

Precedent: Pre-multiplied alpha



filt(target)*filt(alpha) + (1-filt(alpha))*source

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

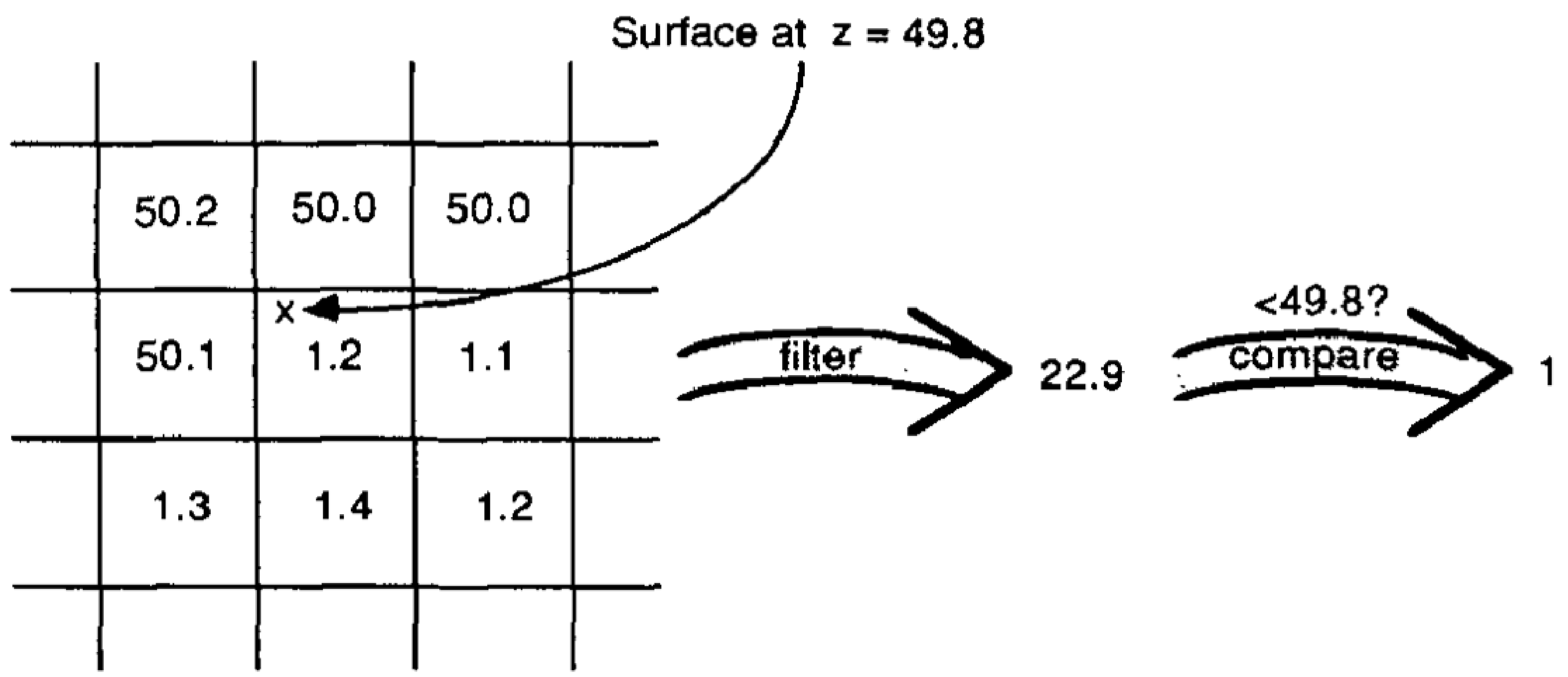
Precedent: Pre-multiplied alpha

filt(target)*filt(alpha) + (1-filt(alpha))*source

filt(target*alpha) + (1-filt(alpha))*source

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

Precedent: Percentage Closer Shadow Filtering

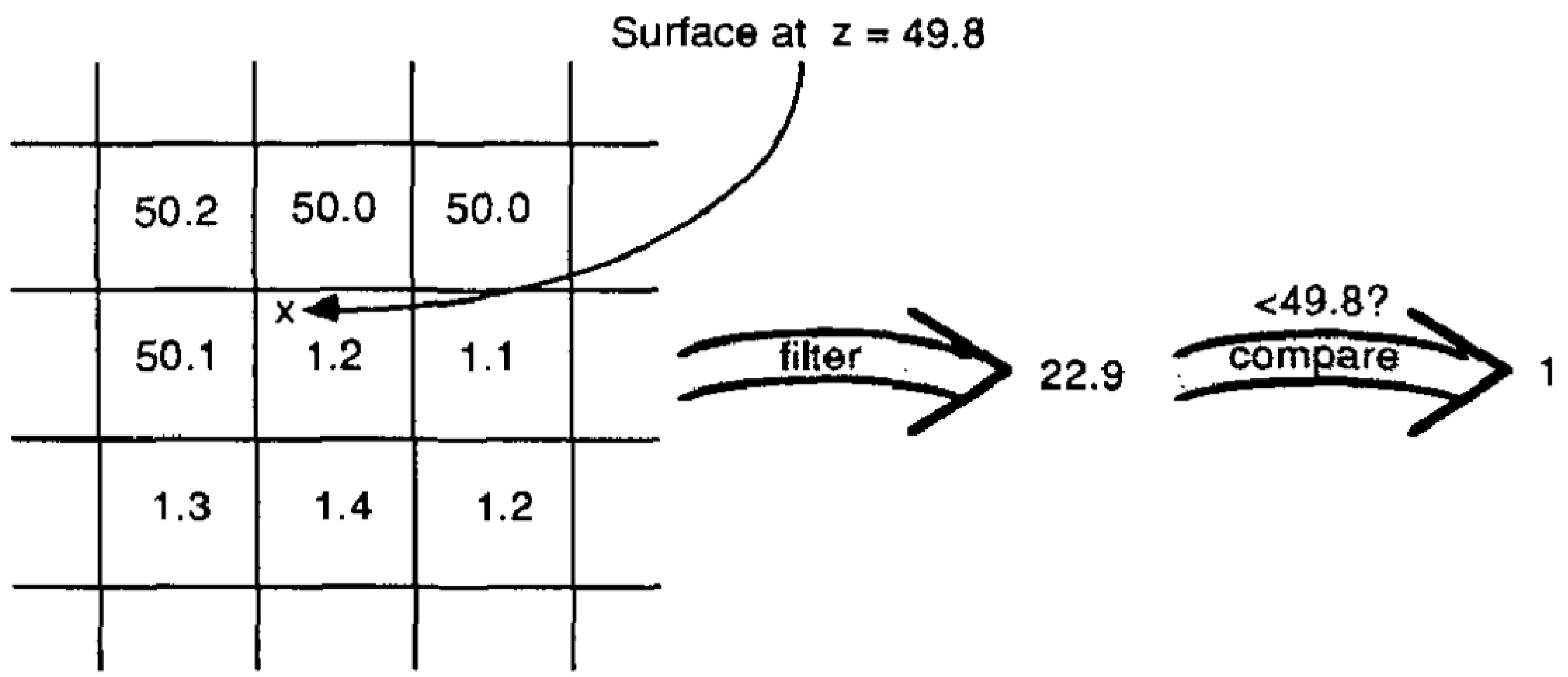


a) Ordinary texture map filtering. Does not work for depth maps.

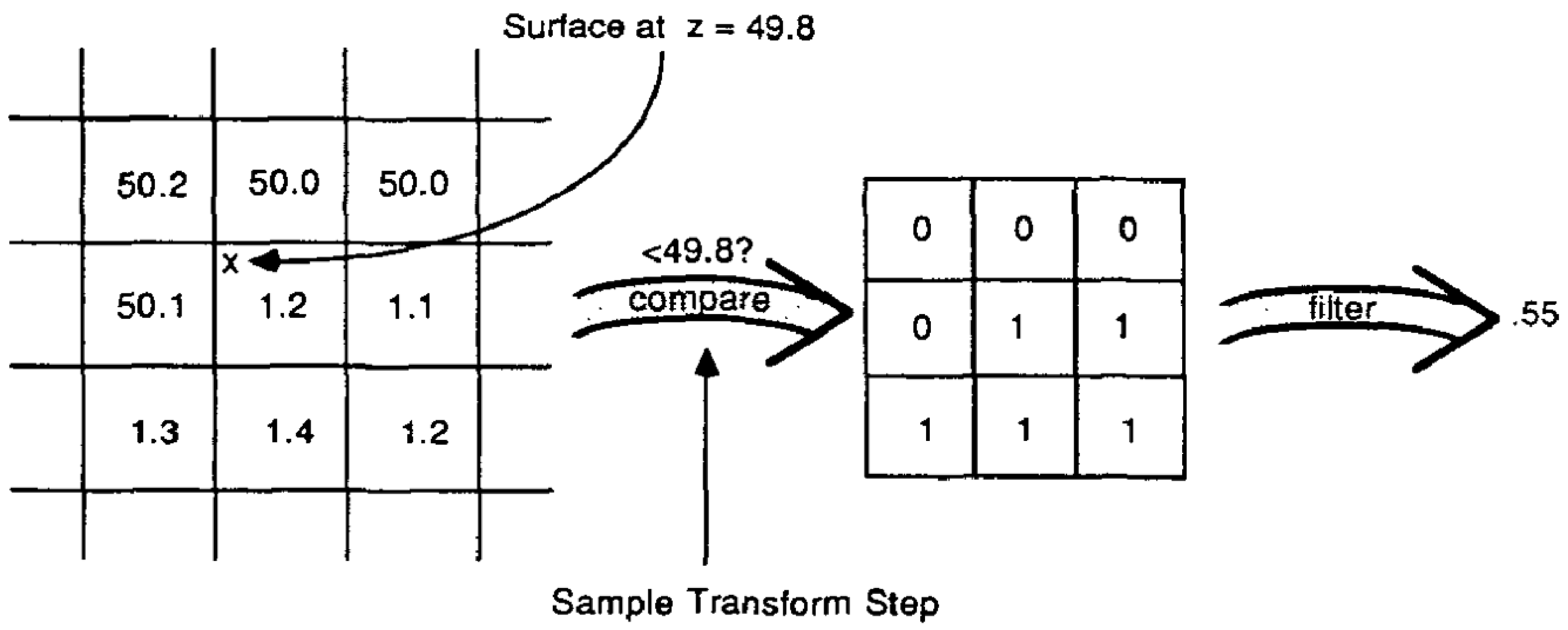
$$visibility = z < \int depth(u,v)\,\mathrm{d}u\,\mathrm{d}v$$

Rendering Antialiased Shadows With Depth Maps, Reeves et al., SIGGRAPH 1987.

Precedent: Percentage Closer Shadow Filtering



a) Ordinary texture map filtering. Does not work for depth maps.



b) Percentage closer filtering.

$$visibility = z < \int depth(u,v)\,\mathrm{d}u\,\mathrm{d}v$$

$$visibility = \int f(u,v)\,(z < depth(u,v))\,\mathrm{d}u\,\mathrm{d}v$$

Jitter (u,v) to sample f

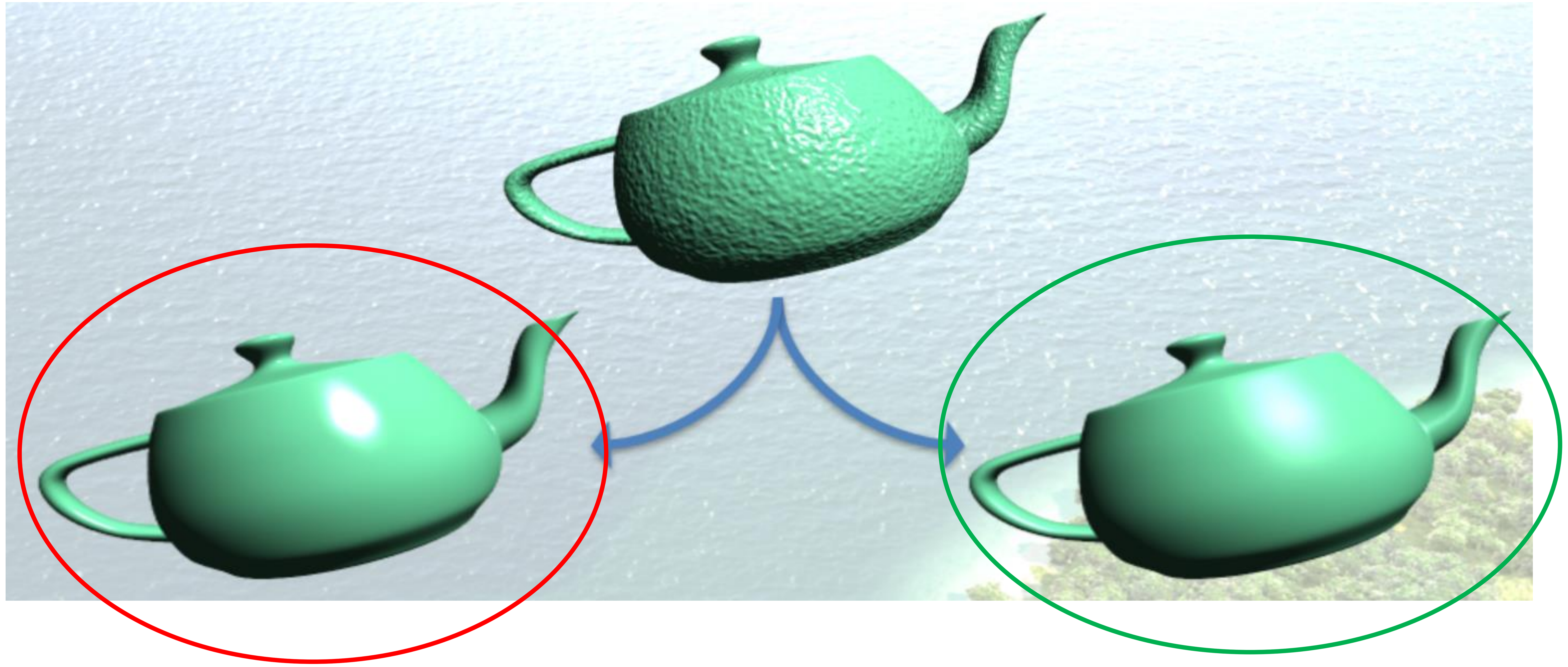
Rendering Antialiased Shadows With Depth Maps, Reeves et al., SIGGRAPH 1987.

Precedent: Percentage Closer Shadow Filtering

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.

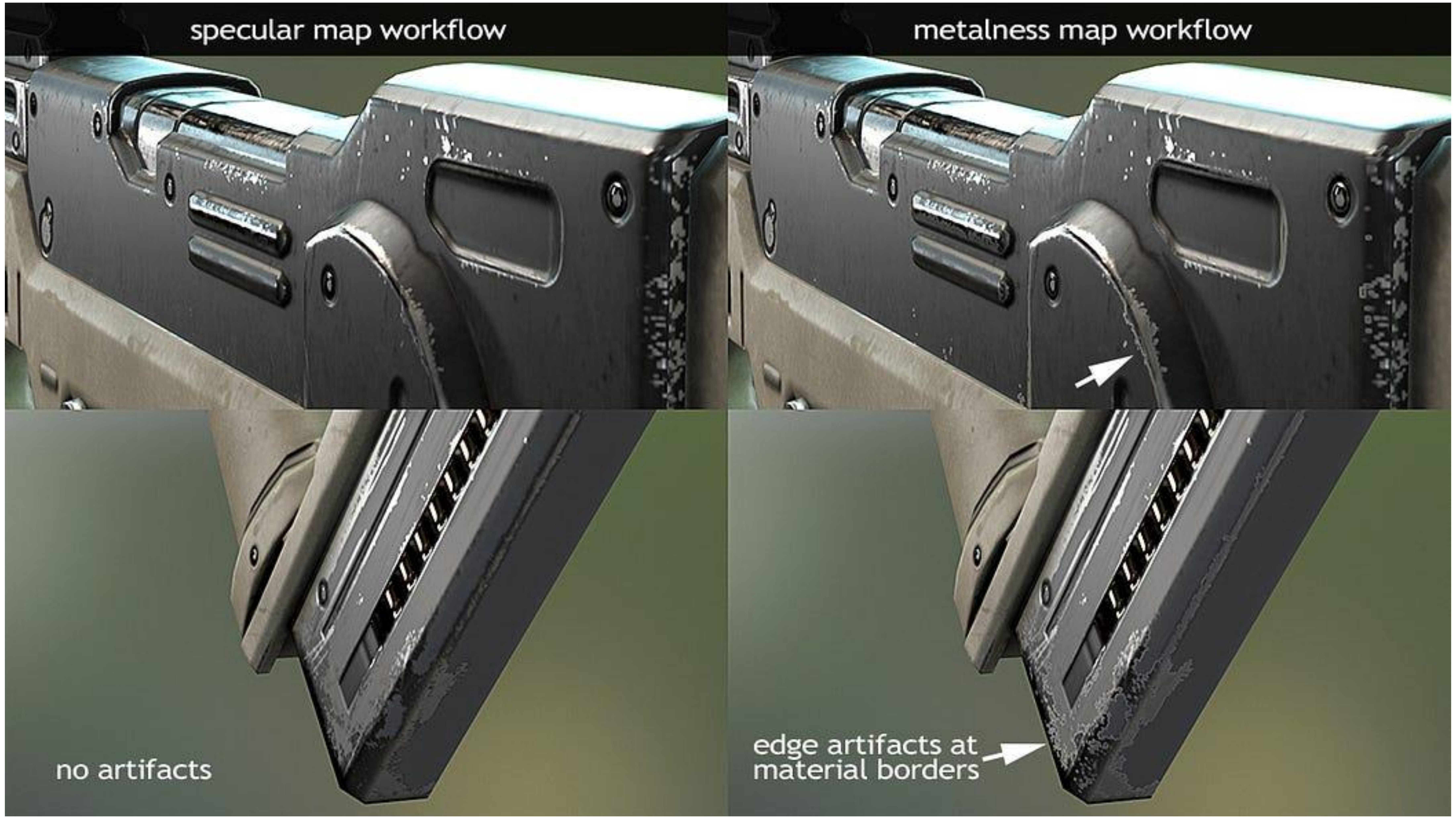
Rendering Antialiased Shadows With Depth Maps, Reeves et al., SIGGRAPH 1987.

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.

Rendering Antialiased Shadows With Depth Maps, Reeves et al., SIGGRAPH 1987.

Textures and the Rendering Equation

Filtering Before Shading (Standard Practice Today)

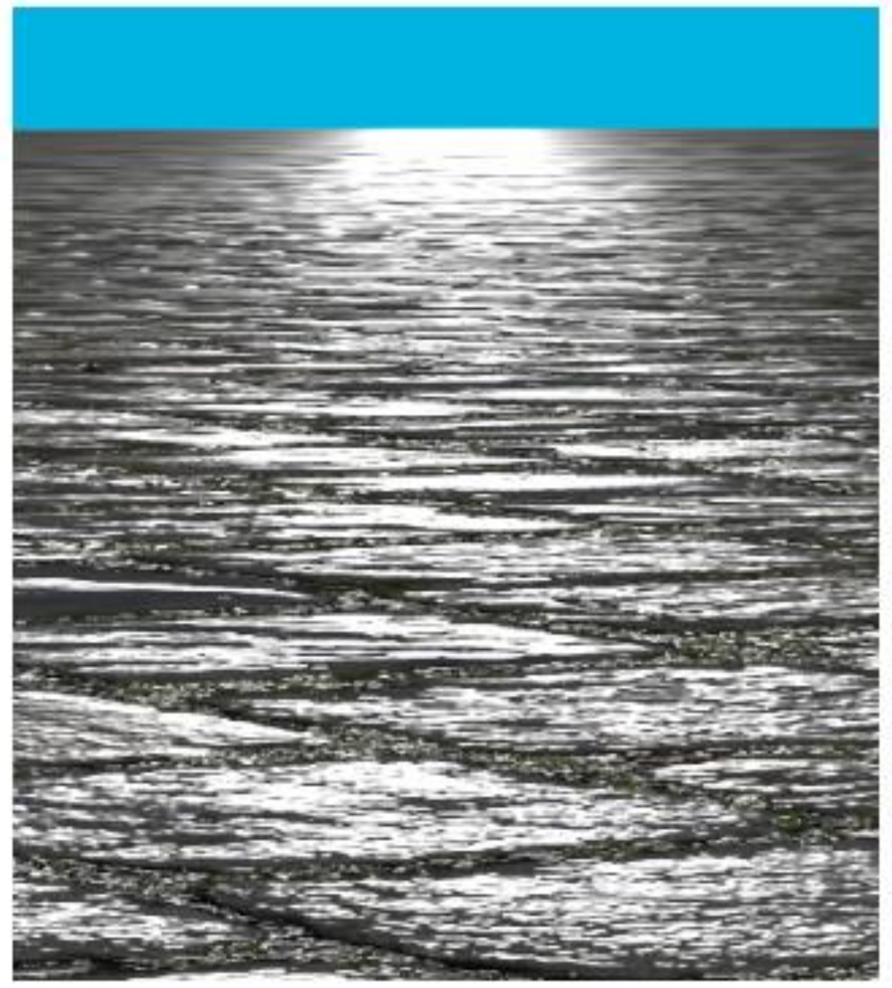
$$L_o(p,\omega_o) = \int_{\mathbb{S}^2} \overbrace{f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i}^{\text{shading}}$$

Filtering Before Shading (Standard Practice Today)

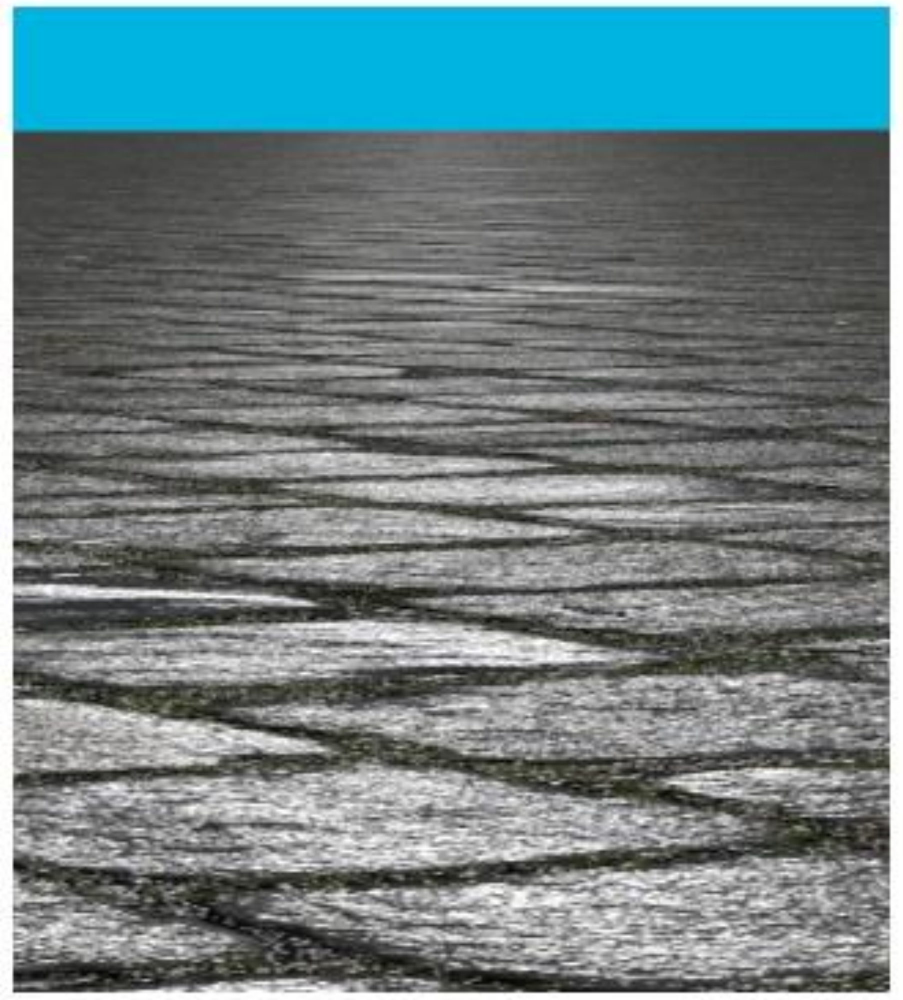
$$L_o(p,\omega_o) = \overbrace{\int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i}^{\text{shading}}$$

$$\textbf{reflectance} = \underbrace{\int f(u,v)\, tex_{\text{reflectance}}(u,v)\, \mathrm{d}u\, \mathrm{d}v}_{\text{filtering}}$$

$$\textbf{normal} = \underbrace{\int f(u,v)\, tex_{\text{normalmap}}(u,v)\, \mathrm{d}u\, \mathrm{d}v}_{\text{filtering}}$$



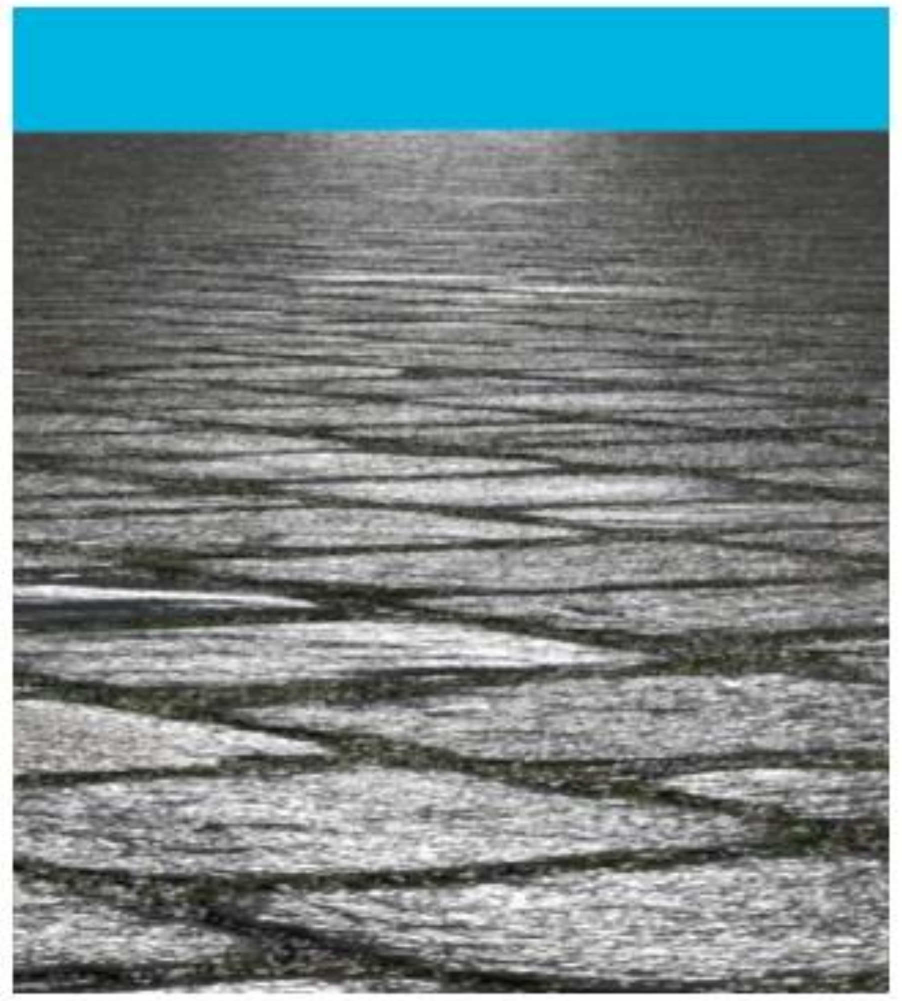
HW 16x Aniso



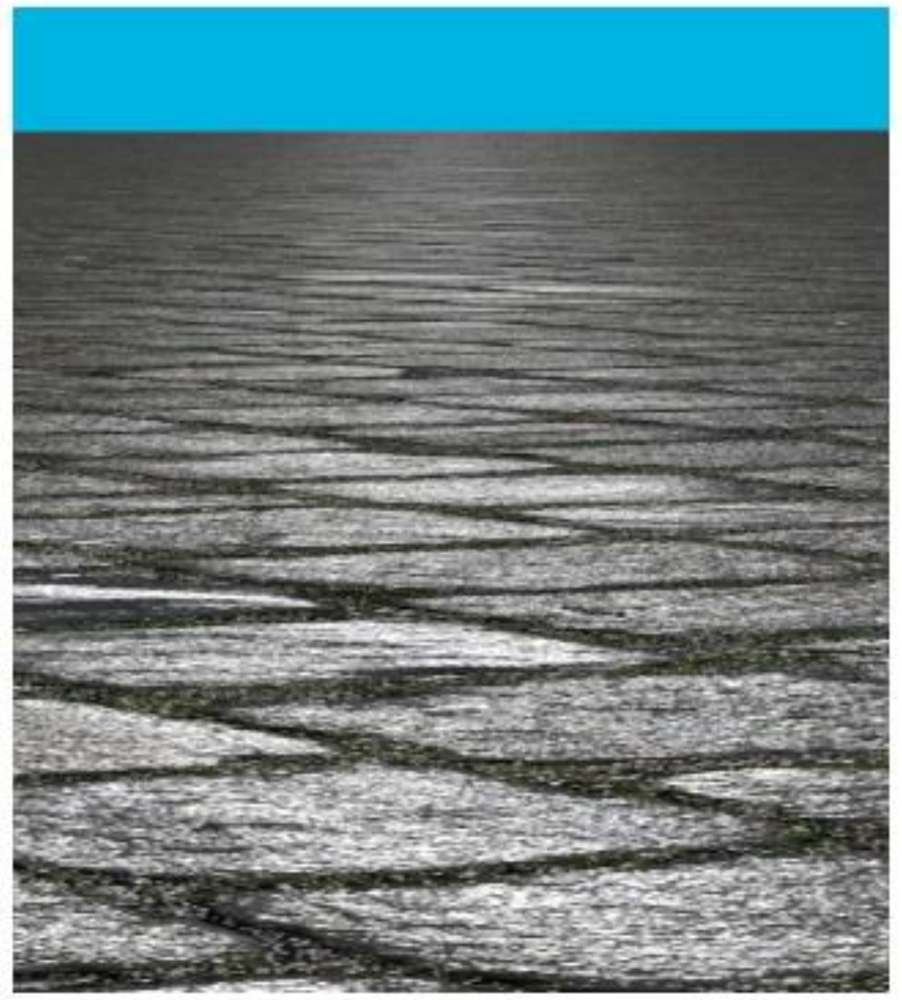
Reference

Filtering After Shading

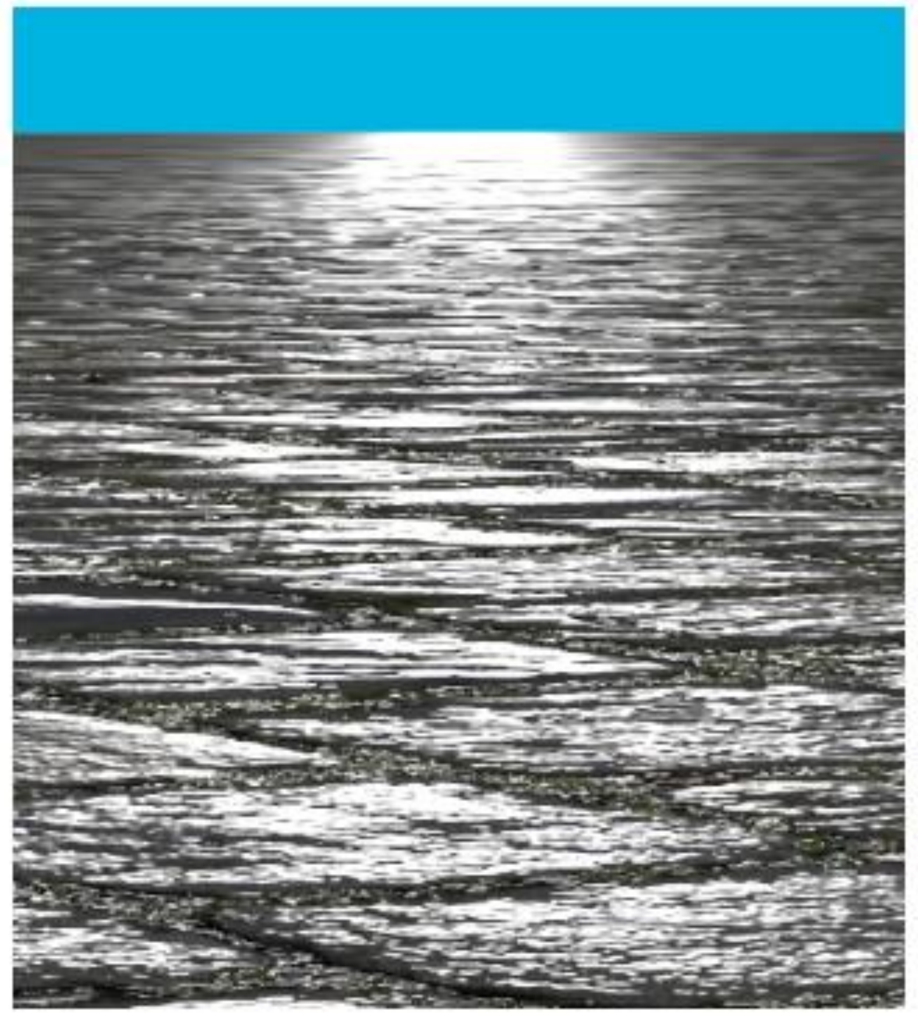
$$L_o(p,\omega_o) = \overbrace{\int f(u,v)}^{\text{filtering}} \left(\overbrace{\int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i}^{\text{shading}} \right) \mathrm{d}u\, \mathrm{d}v$$



Filter after shading
(real-time implementation)



Reference



Filter before shading

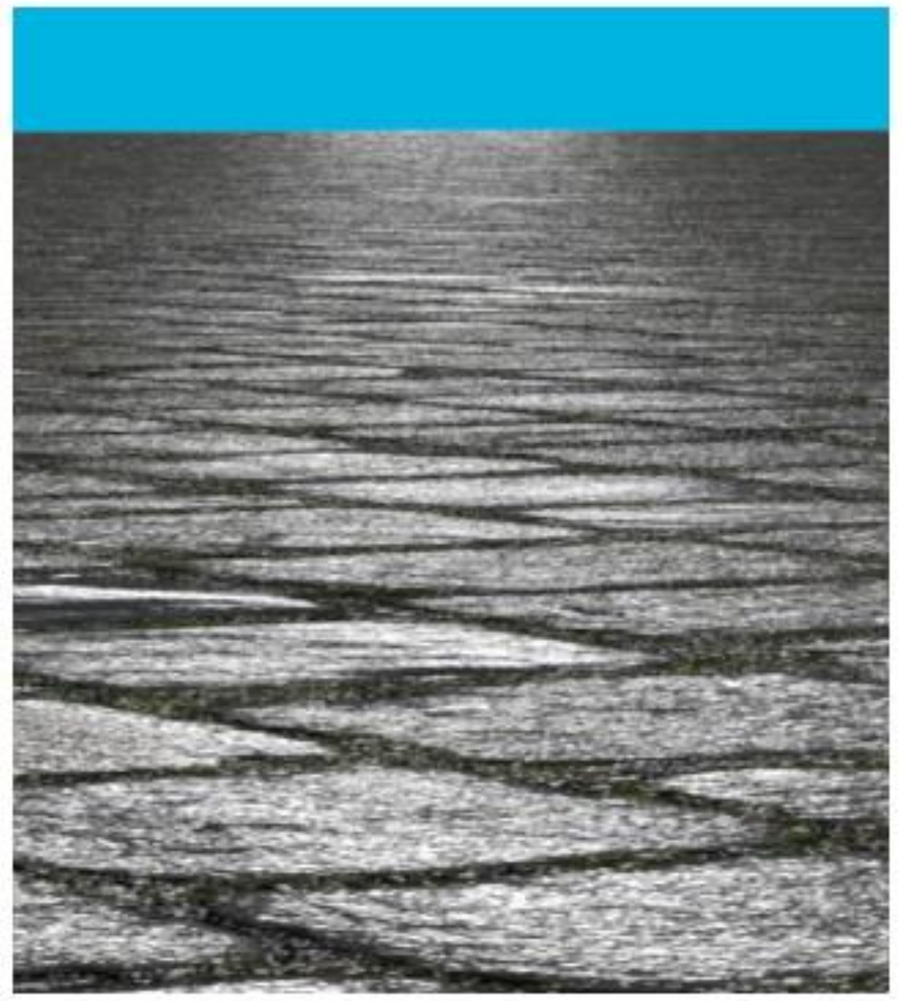
Filtering After Shading

filtering ⏞ $\int f(u,v)$; shading ⏞ $\int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, d\omega_i$

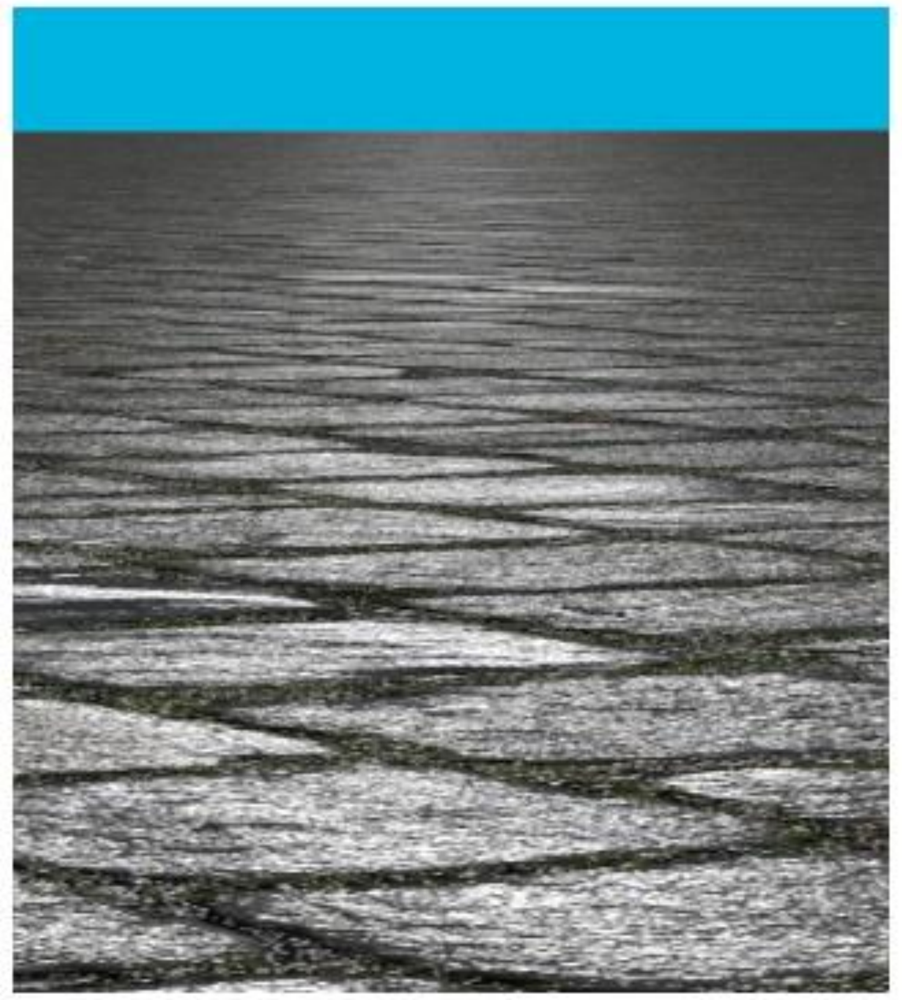
$$L_o(p,\omega_o) = \int f(u,v) \left(\int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, d\omega_i \right) du\, dv$$

↑

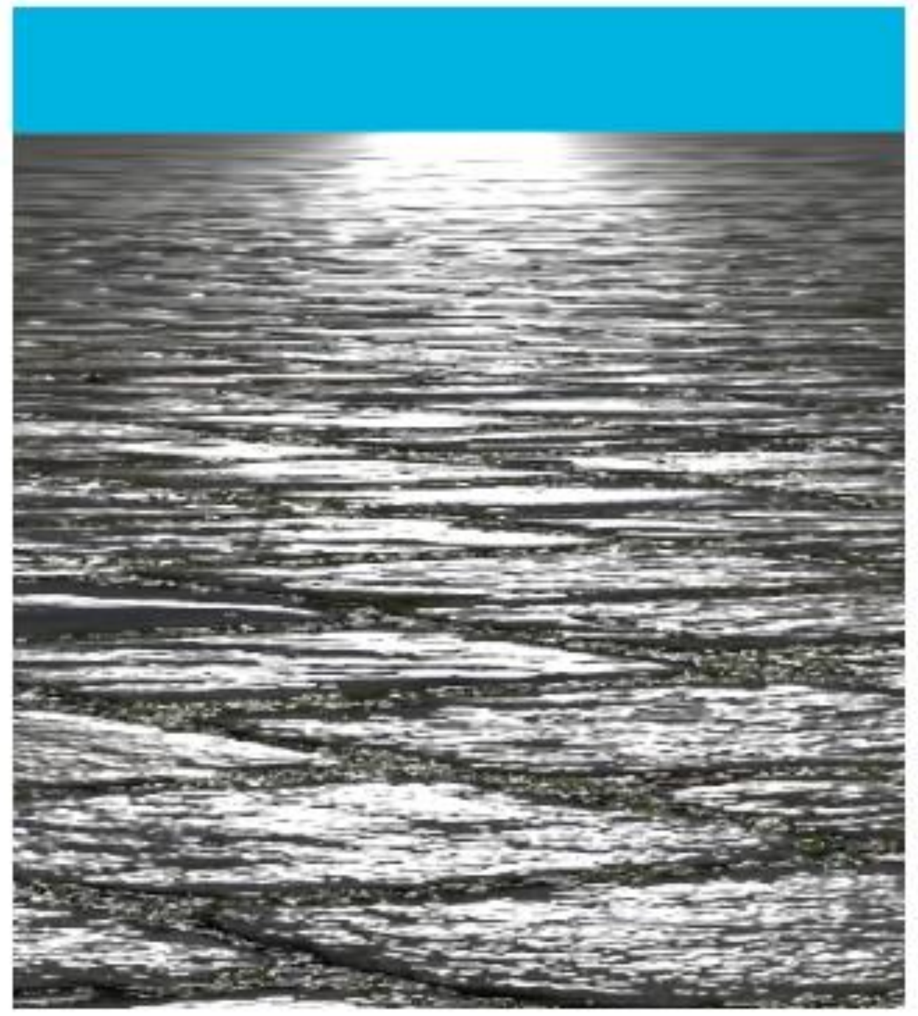
????



Filter after shading
(real-time implementation)



Reference



Filter before shading

Filtering After Shading

$$L_o(p,\omega_o) = \overbrace{\int f(u,v)}^{\text{filtering}} \left(\overbrace{\int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i}^{\text{shading}} \right) \mathrm{d}u\, \mathrm{d}v$$

- Use Monte Carlo!
- Sample $(u', v') \sim f(u,v)$

Filtering After Shading

$$L_o(p,\omega_o) = \overbrace{\int f(u,v)}^{\text{filtering}} \Bigg(\overbrace{\int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i}^{\text{shading}} \Bigg) \,\mathrm{d}u\, \mathrm{d}v$$

- Use Monte Carlo!
- Sample $(u',v') \sim f(u,v)$
- Estimator: $L_o(p,\omega_o) \approx \dfrac{f(u,v)}{p(u,v)} \left(\int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i \right) \mathrm{d}u\, \mathrm{d}v$

$$= \int_{\mathbb{S}^2} f_r(\omega_i,\omega_o)\, L_i(p,\omega_i)\, |\cos\theta_i|\, \mathrm{d}\omega_i$$

$$\textbf{reflectance} = tex_{\text{reflectance}}(u',v') \qquad \textbf{normal} = tex_{\text{normalmap}}(u',v')$$

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

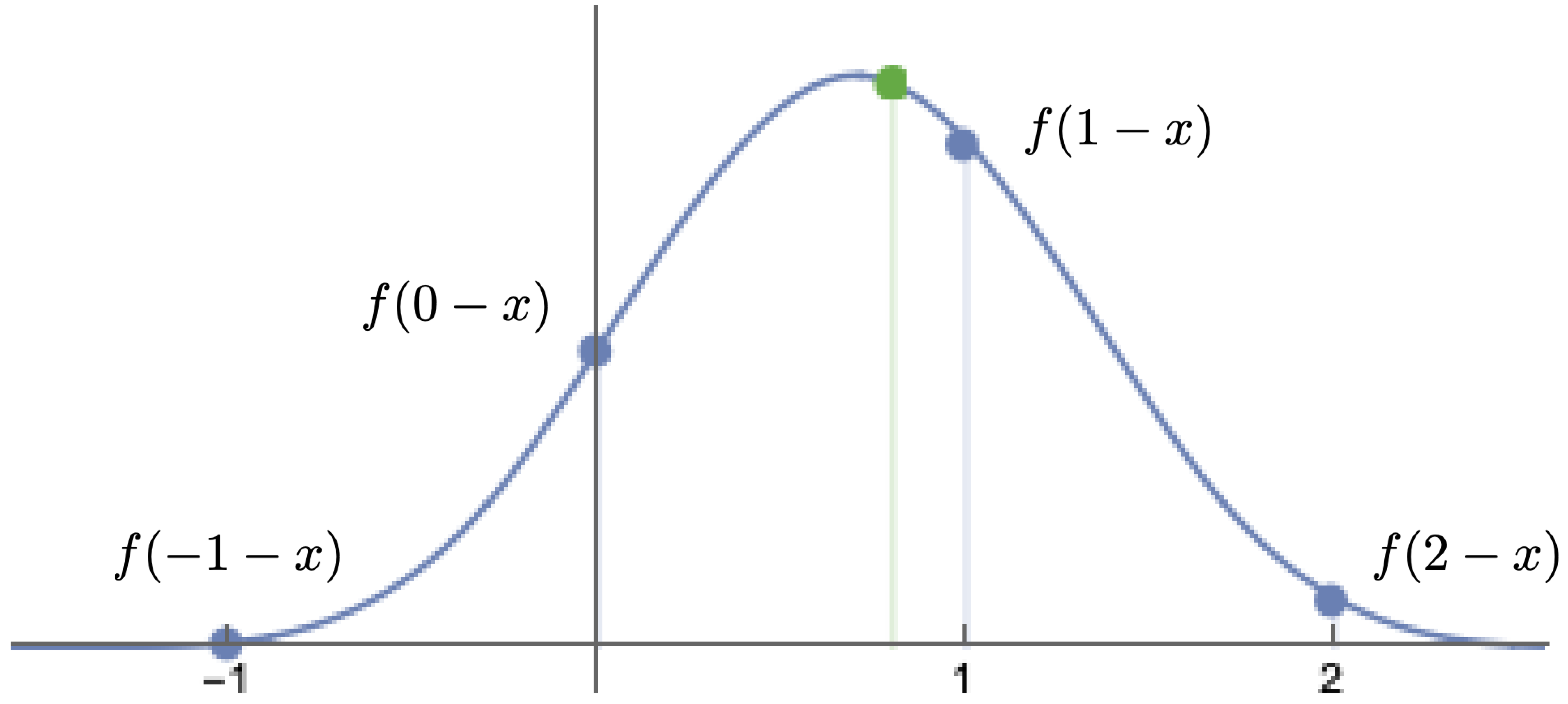
White noise | (Spatiotemporal) Blue Noise

Single frame

DLSS

Two families of methods

Sampling Texture Filters – Discrete, 1D



$$lookup(x) = \int f(u-x)\,t(u)\,\mathrm{d}u = \sum_{u=-1}^{2} \underbrace{f(u-x)}\,t(u)$$

Chose a sample with probability ~f

Filter Reservoir Sampling

- **Importance sampling**: Sample a texel with probability p~f
- Sample an array of weights or online through weighted reservoir sampling
- Details in the paper

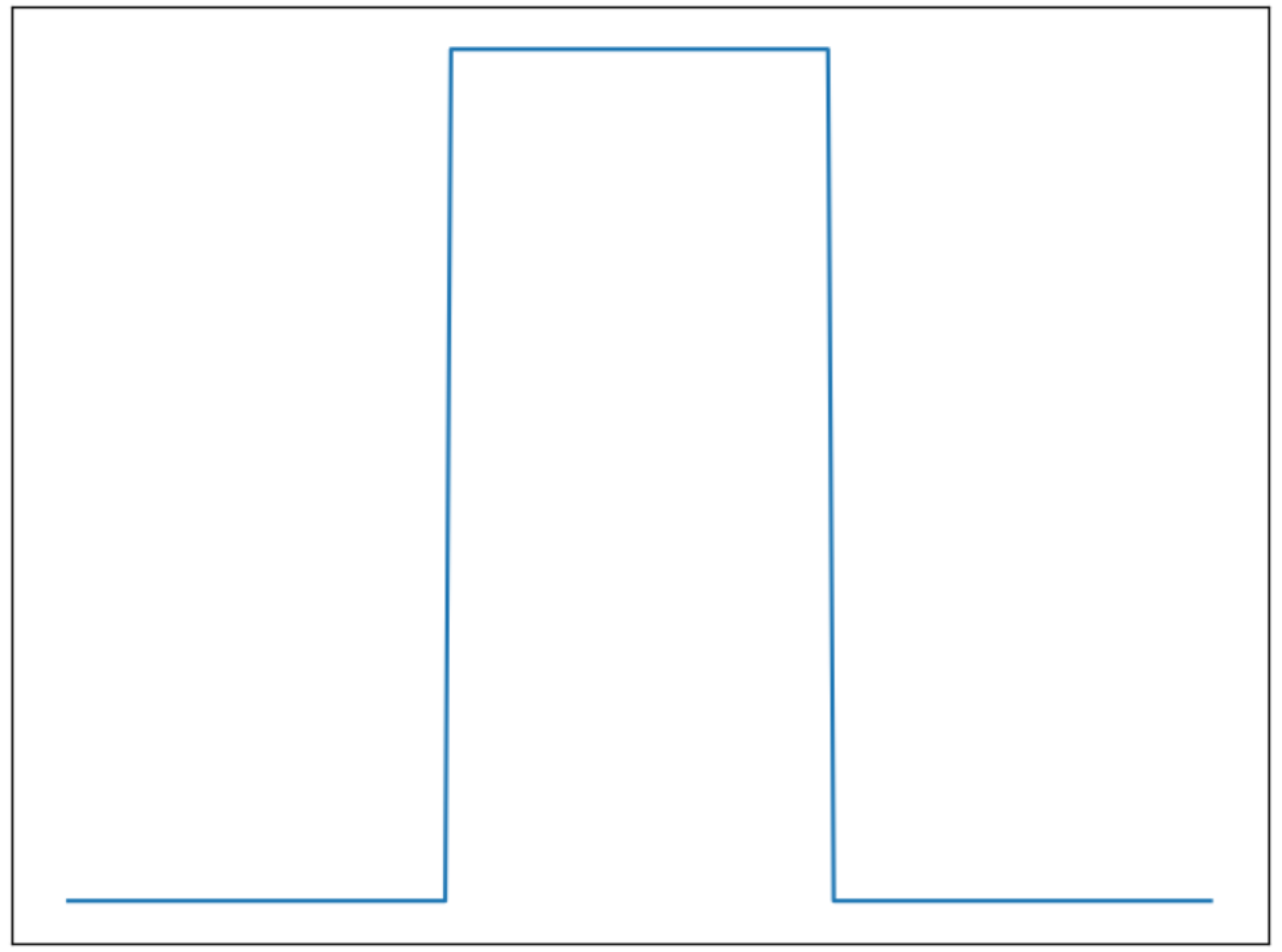
Sampling Texture Filters
Disadvantages of Filter Reservoir Sampling

- 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)

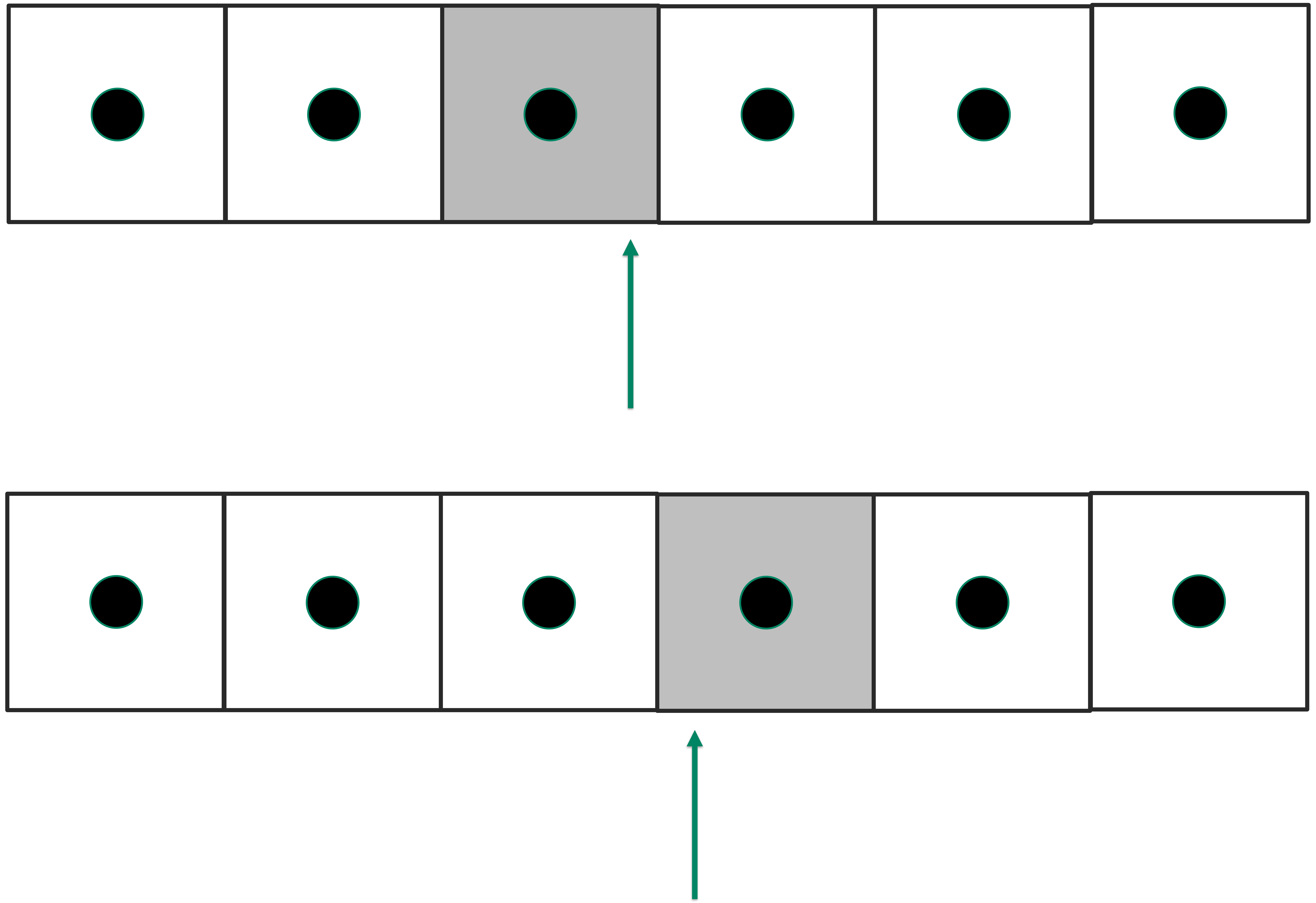
Sampling Texture Filters
Disadvantages of Filter Reservoir Sampling

- Discrete filter sampling – with large filters, can be costly
- Evaluate filter function K^M or K*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.

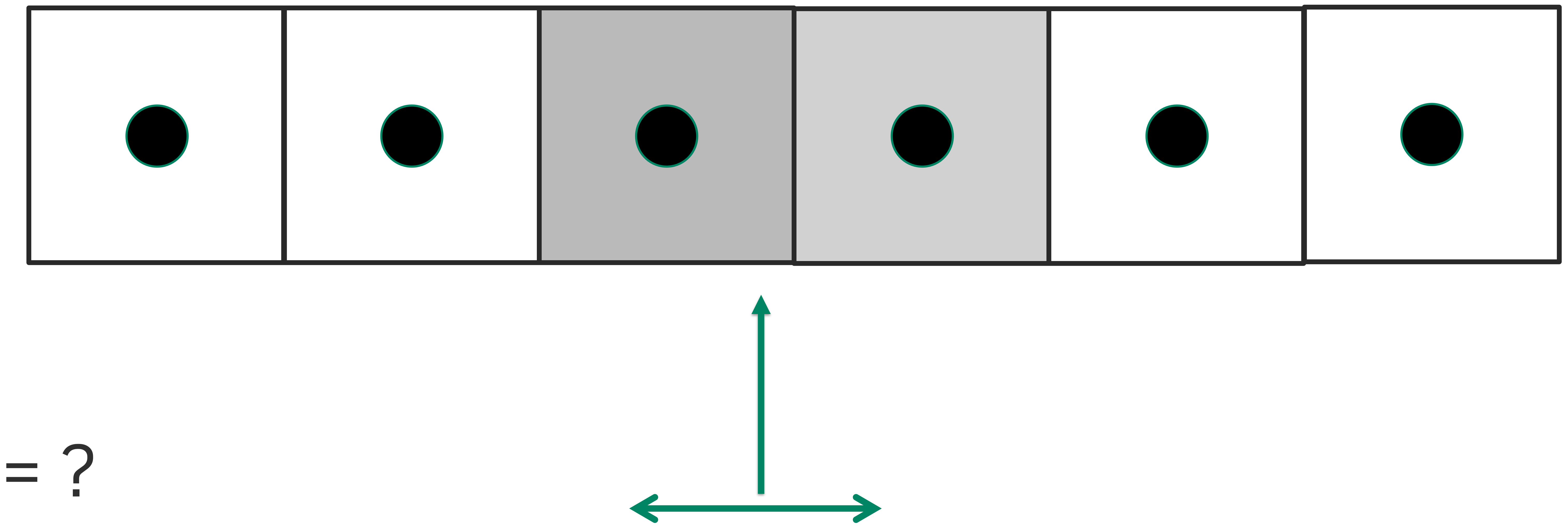
Magnification
What happens when you take a nearest-neighbor sample?



Nearest neighbor = box filter

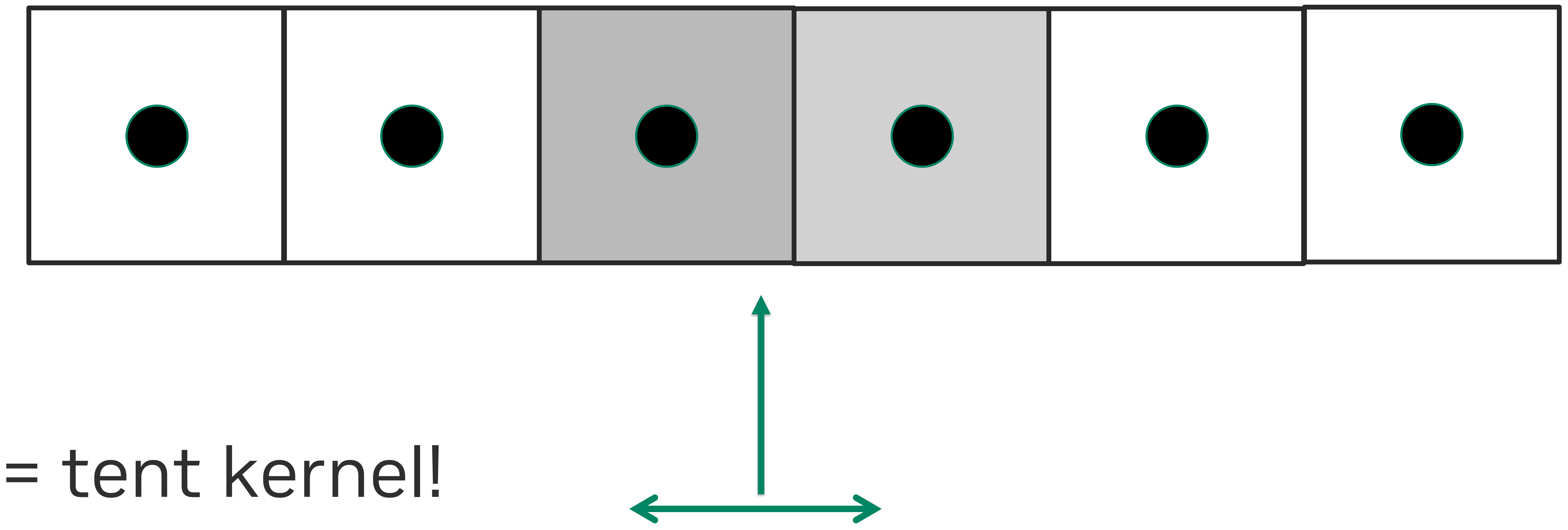


Magnification
Filter Importance Sampling through UV jittering



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 * Box Kernel Convolution

Magnification
Filter Importance Sampling through UV jittering

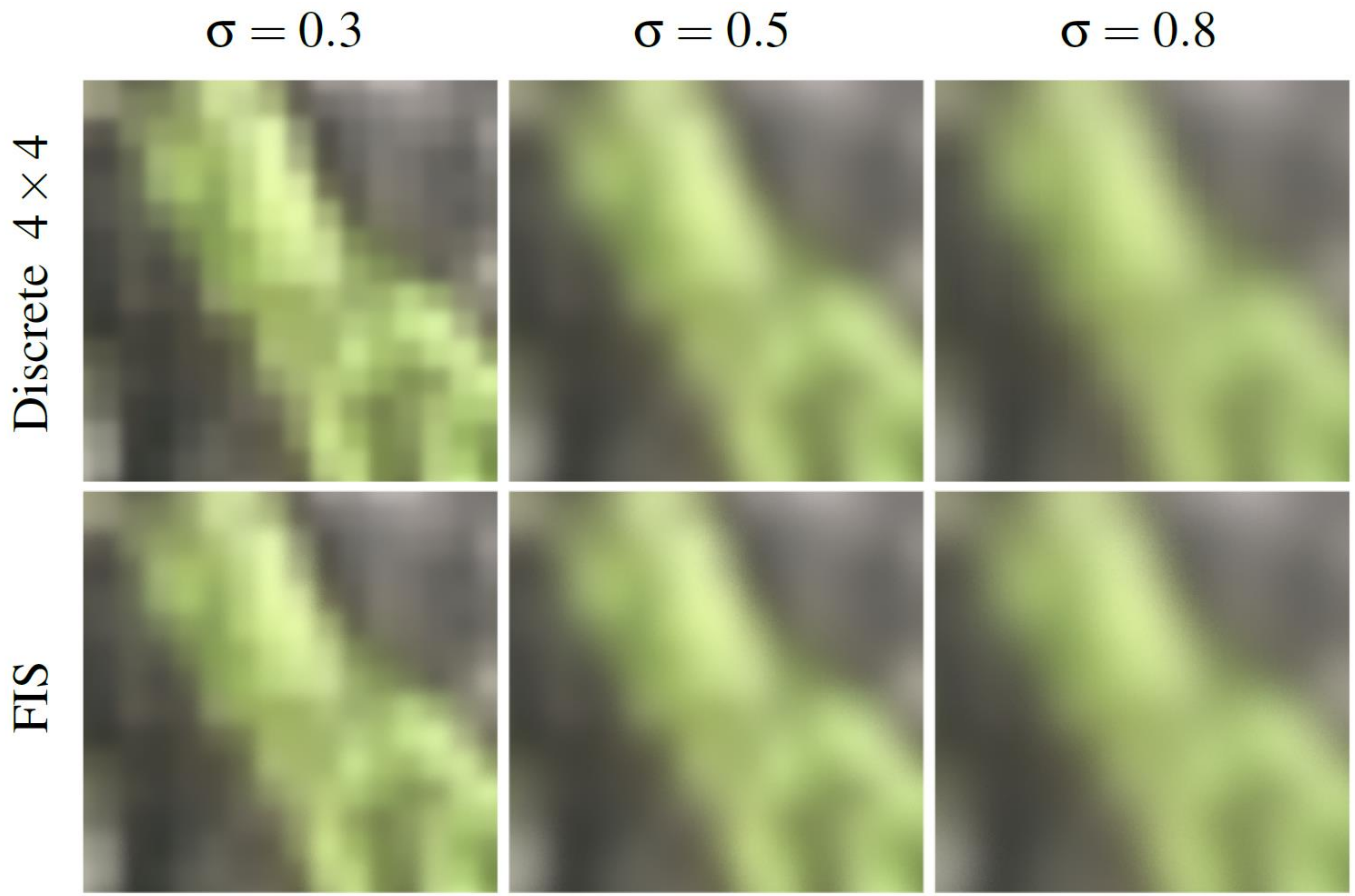
Linear/tent UV jitter + nearest neighbor box = quadratic B-Spline
Quadratic UV jitter + nearest neighbor box = **cubic B-Spline**

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

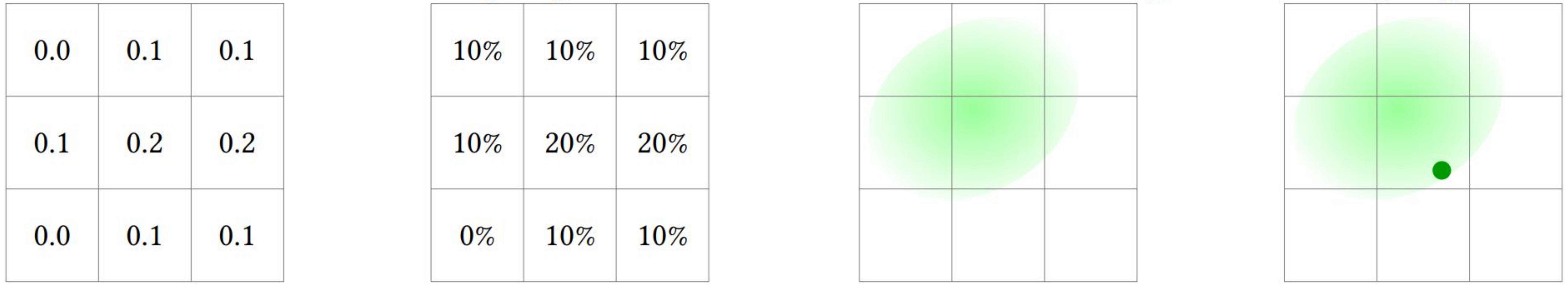
- For B-Spline filters, this additional box is desirable!
- Can sample other, including infinite spatial support filters
- Jitter UVs according to PDF deconvolved with a box
- For many other filters -> can still be advantageous (prevent Gaussian under-sampling)



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

Filter Reservoir Sampling **Filter Importance Sampling**

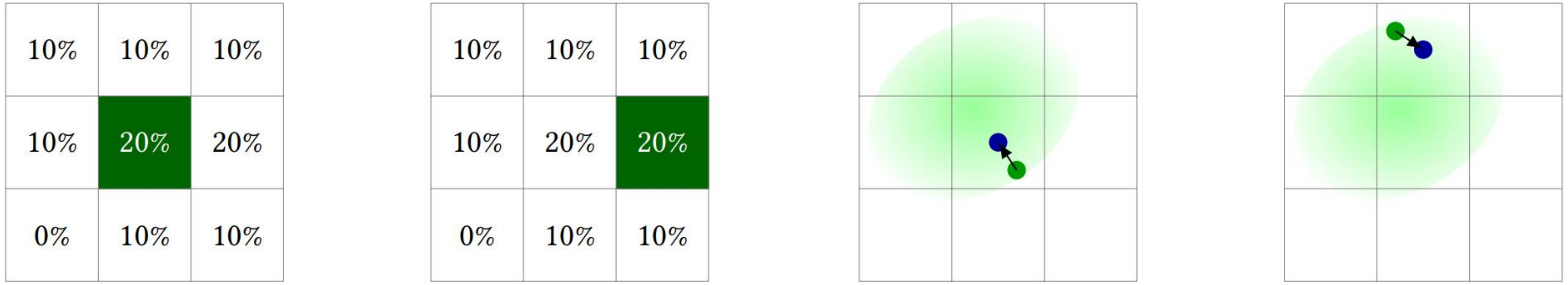


(a) Find texture texels weights.

(b) Convert weights to probabilities.

(a) Select continuous filter PDF.

(b) Sample the continuous distribution to find UV offset.



(c) Randomly select one texel.

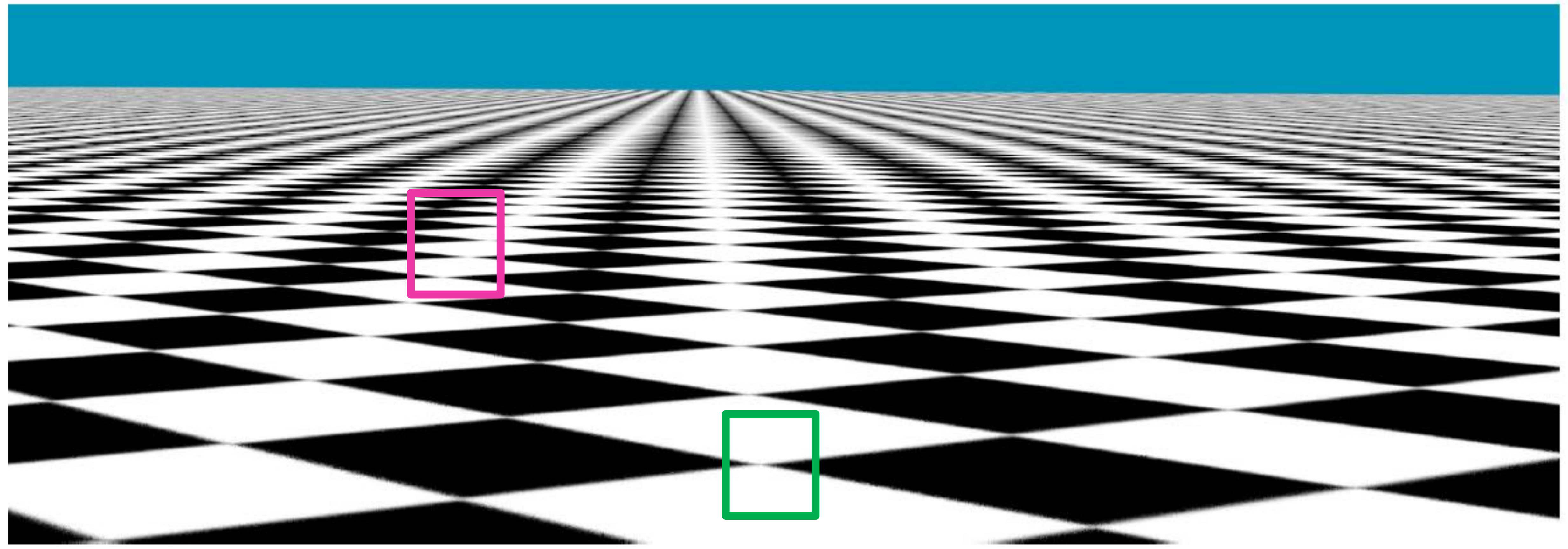
(d) Repeat in the next frame, selecting a different texel.

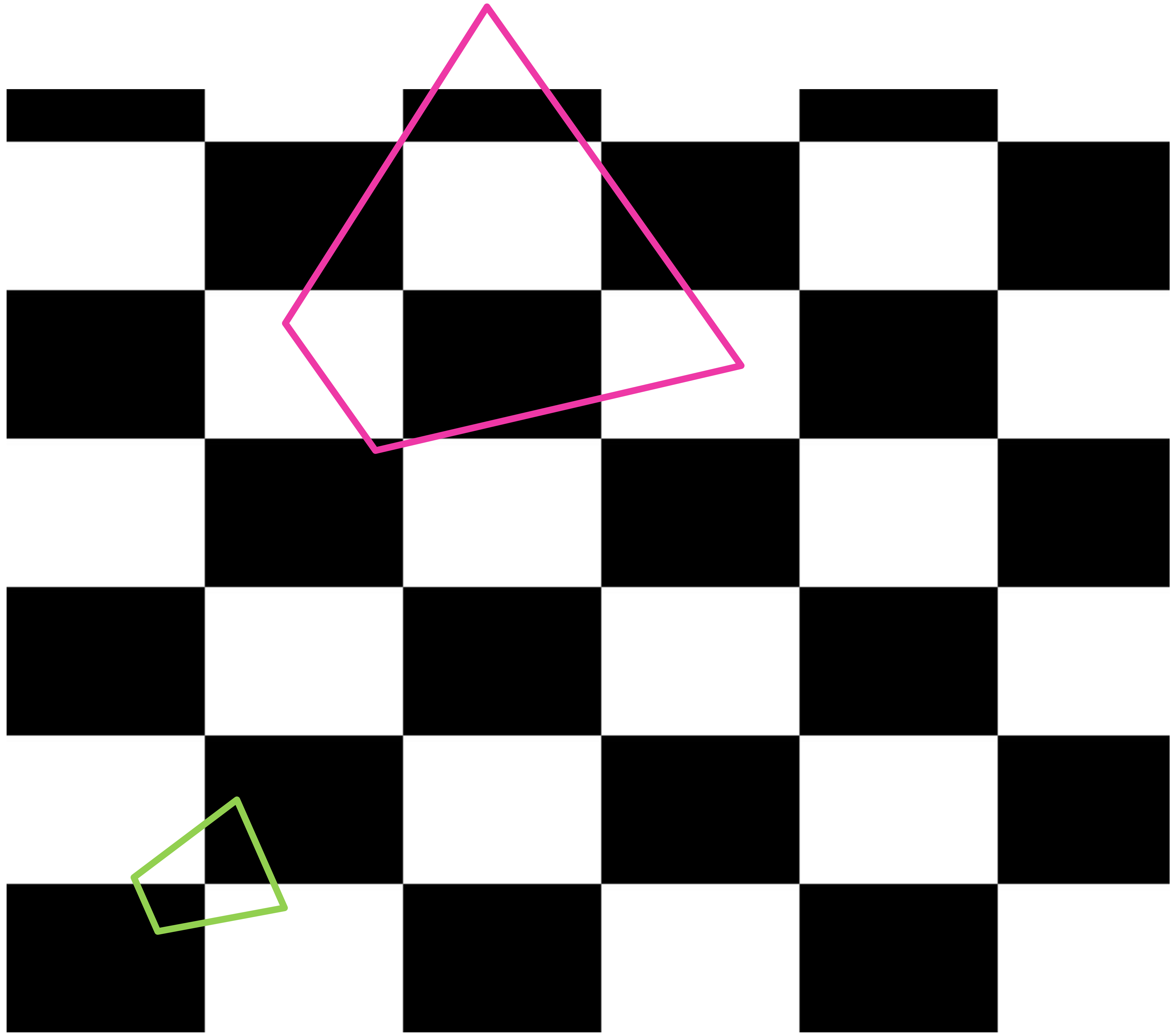
(c) Align UV offset with texel center and sample the texel.

(d) Repeat steps *b* and *c* in the next frame.

Minification
Stochastic Filtering After Shading

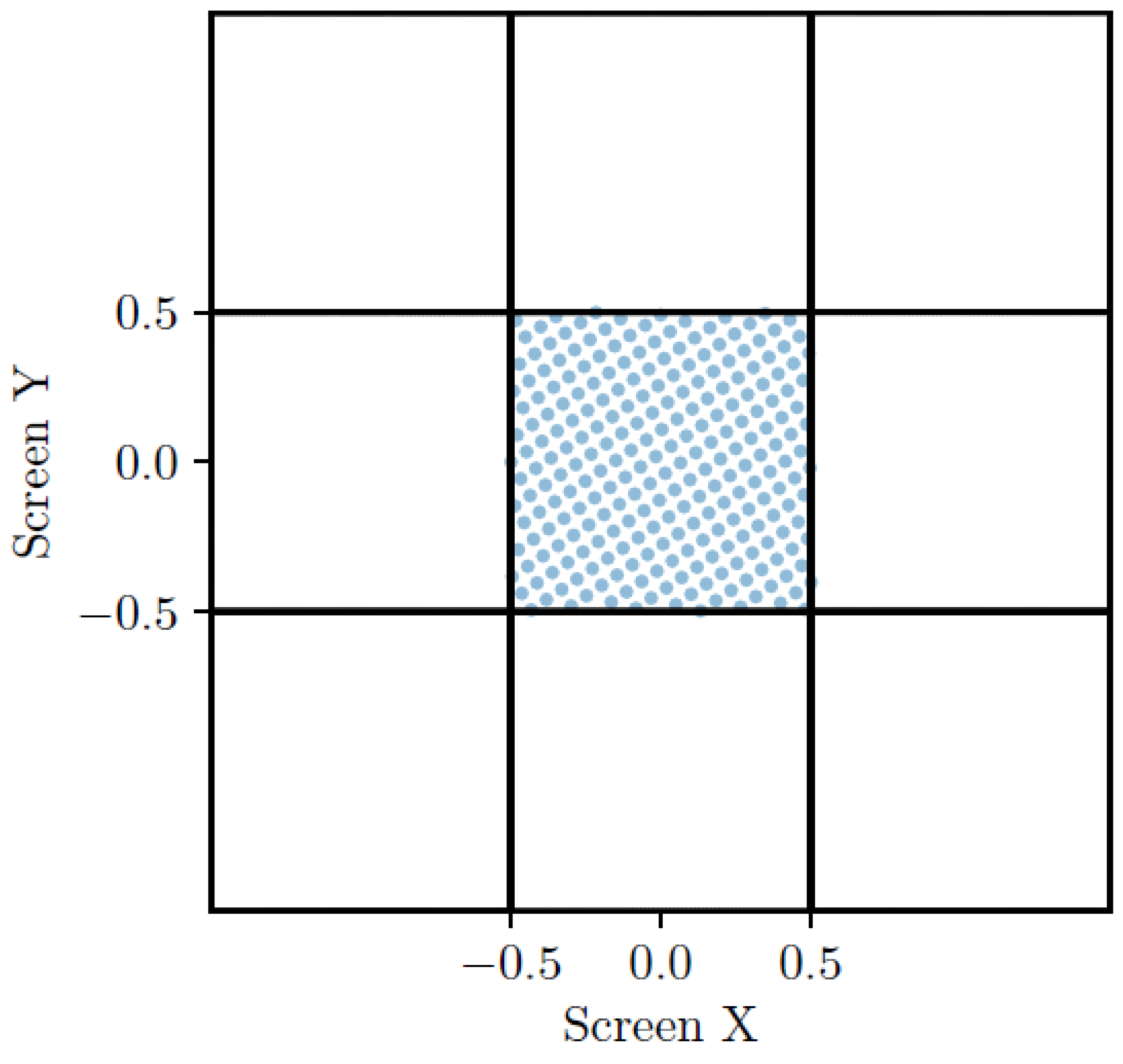
- Anisotropic filtering or elliptically weighted average
- Many pixels, non-uniform mapping for jittering
- There's a simpler, **already-used** method!





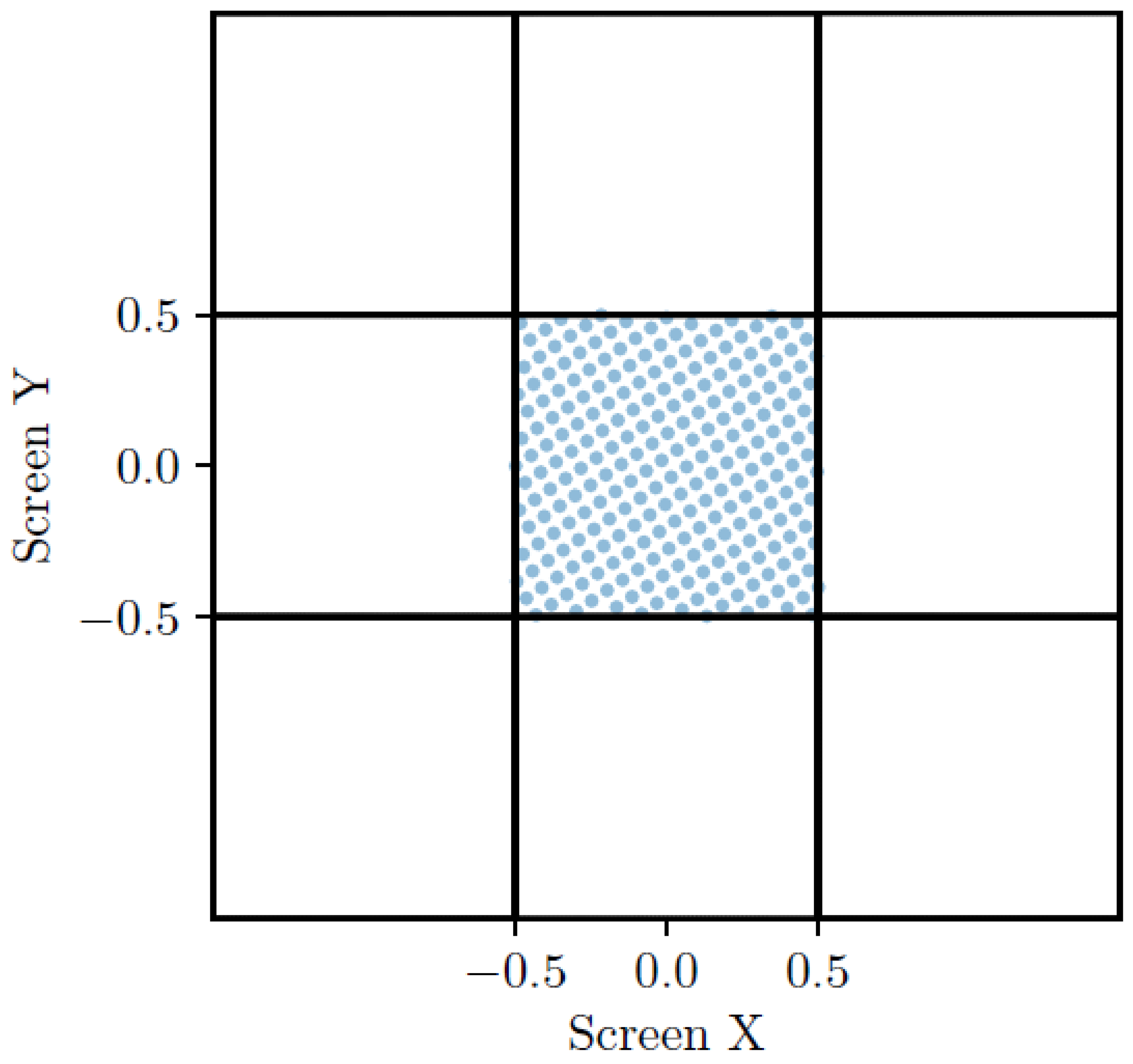
Minification
Stochastic Filtering After Shading

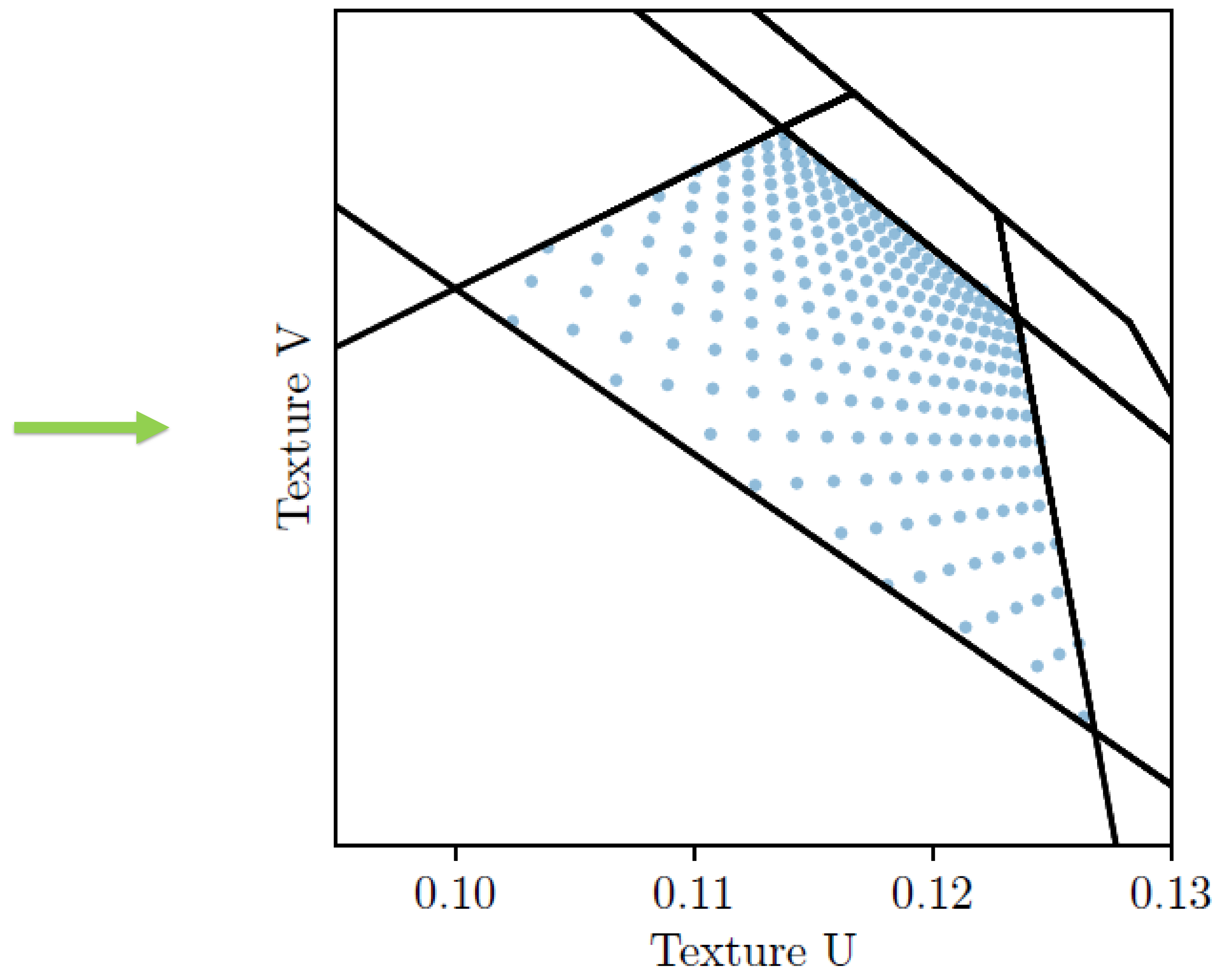
- Common practice – jitter the projection matrix for anti-aliasing reconstruction filter
- Used offline (e.g., MoonRay) and real-time (TAA, DLSS)



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**!



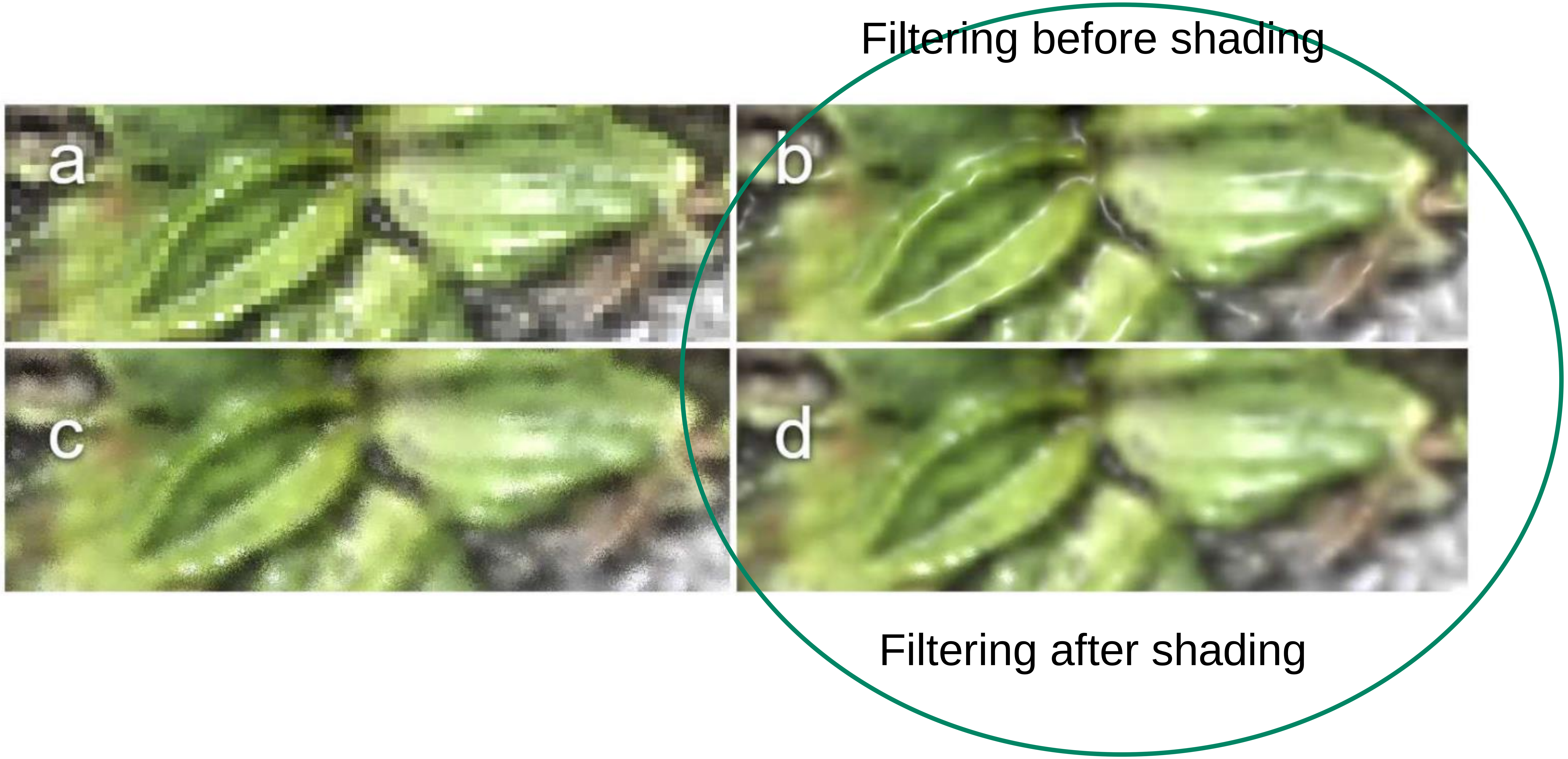


Minification
Stochastic Filtering After Shading

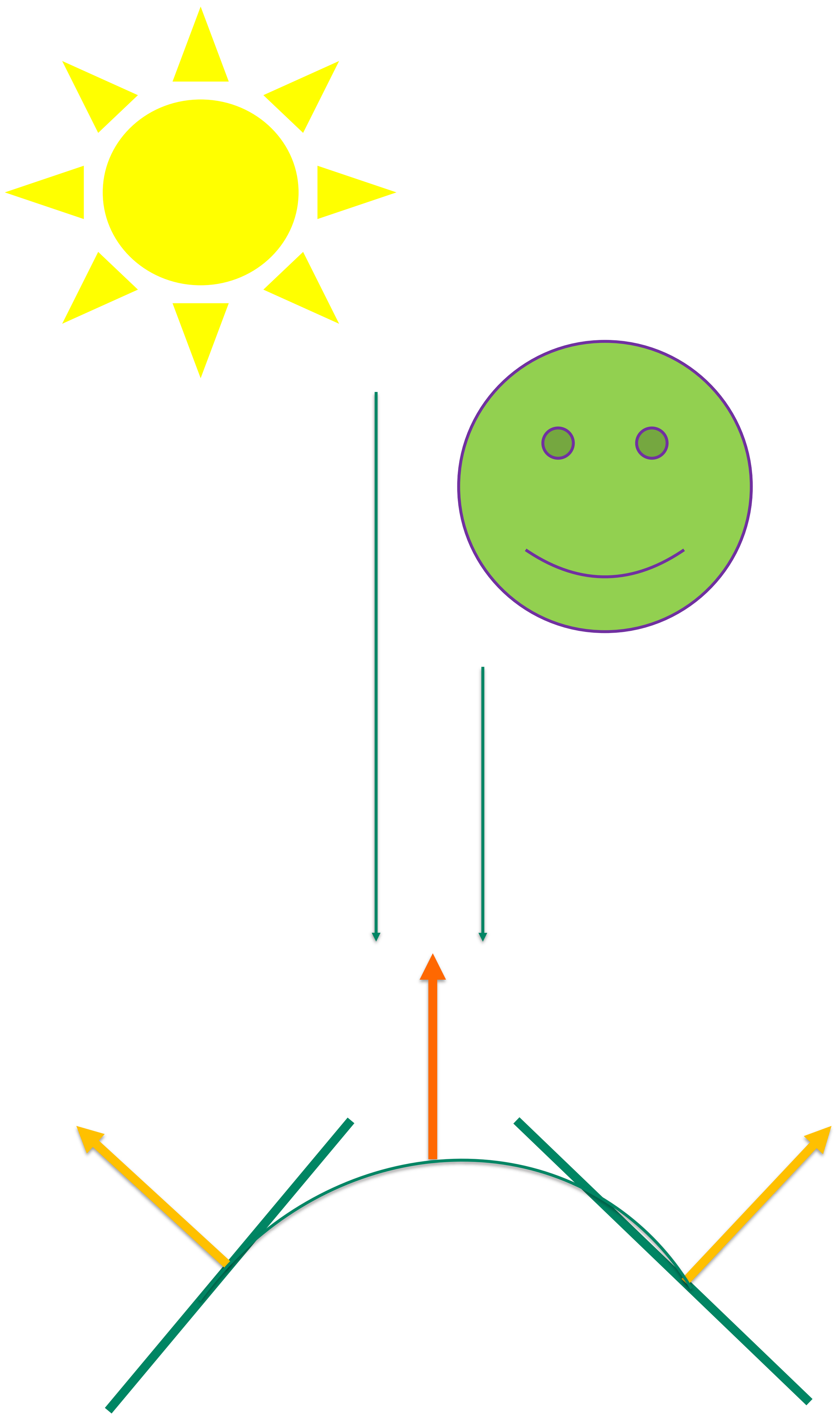
- Common practice – jitter the projection matrix for anti-aliasing reconstruction filter
- Used offline (e.g., MoonRay) and real-time (TAA, DLSS)
- Projects to trapezoid, minification supersampling -> filtering after shading!
- Add magnification/translation UV jitter -> **unified minification and magnification**

Appearance change and possible aliasing

Magnification specular appearance change

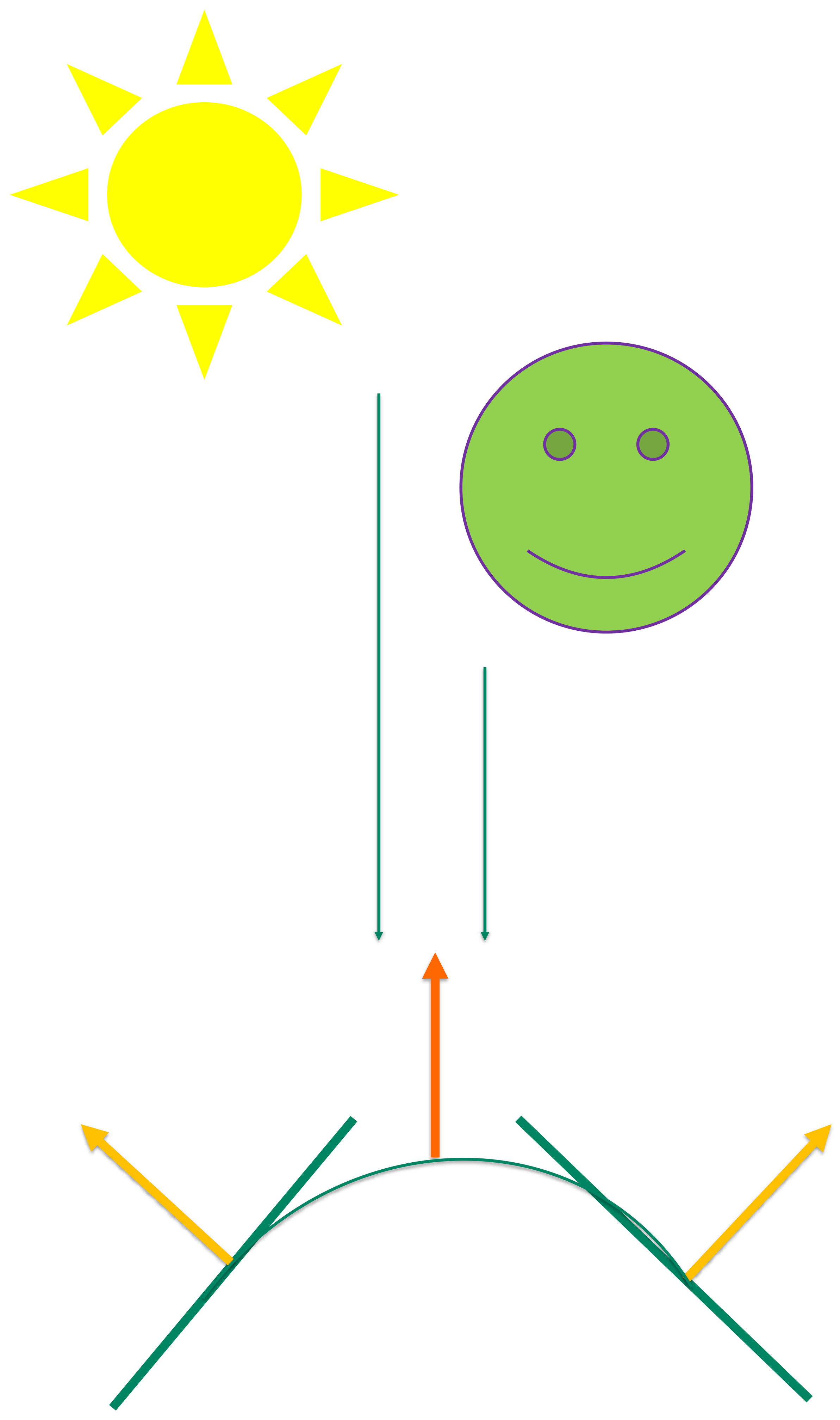


Appearance change explained

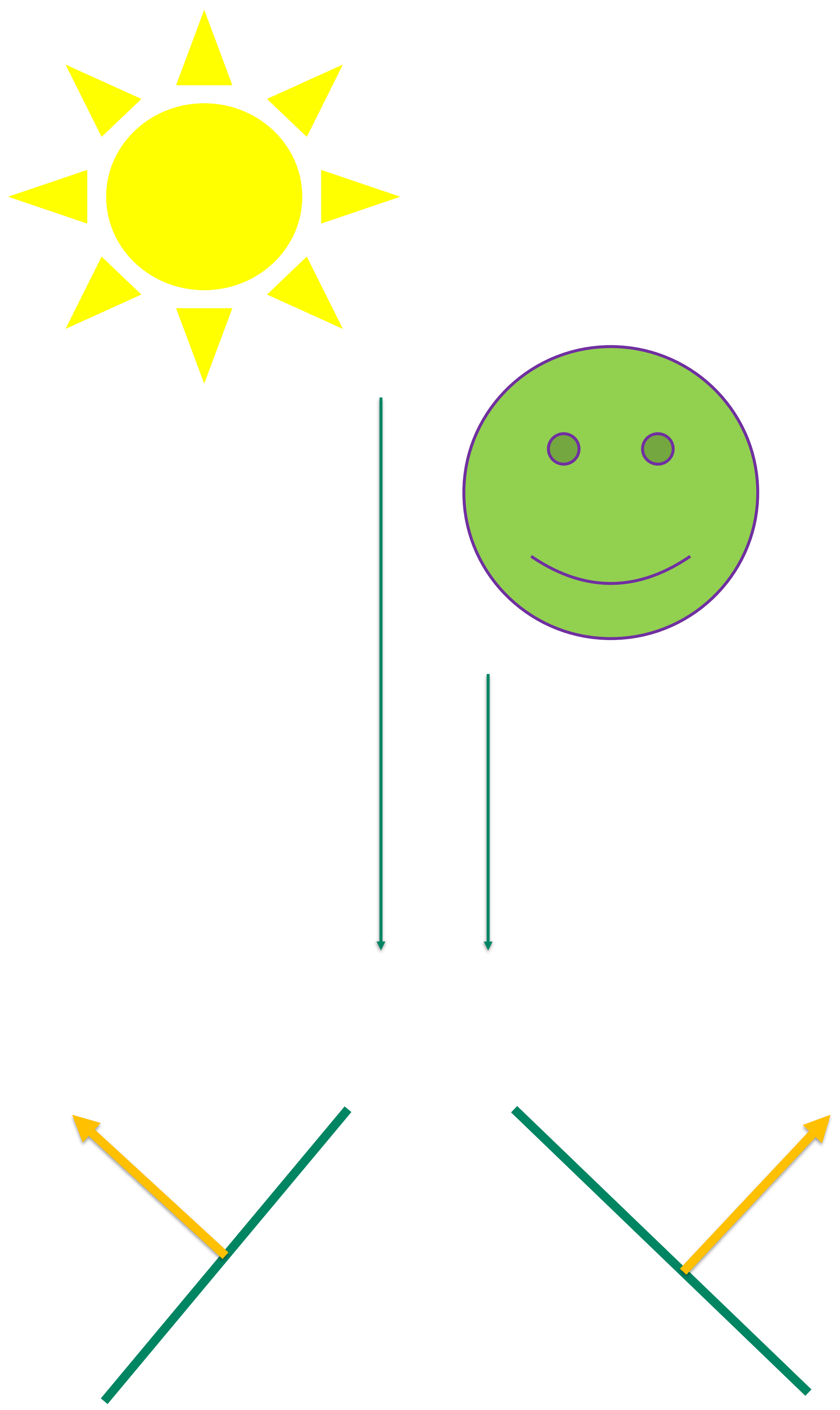


Filtering before shading:
Interpolated surface and normals

Appearance change explained

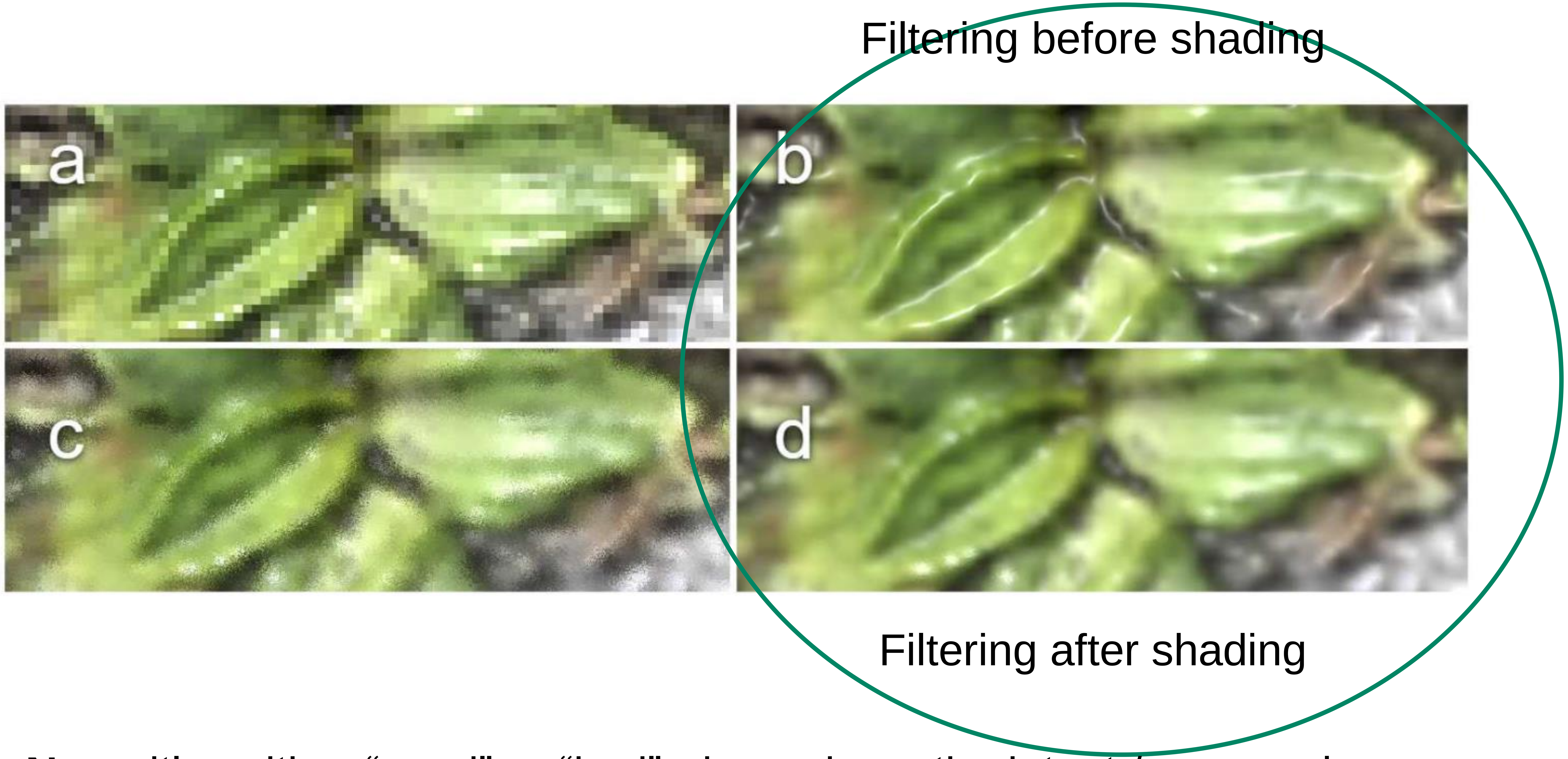


Filtering before shading:
Interpolated surface and normals



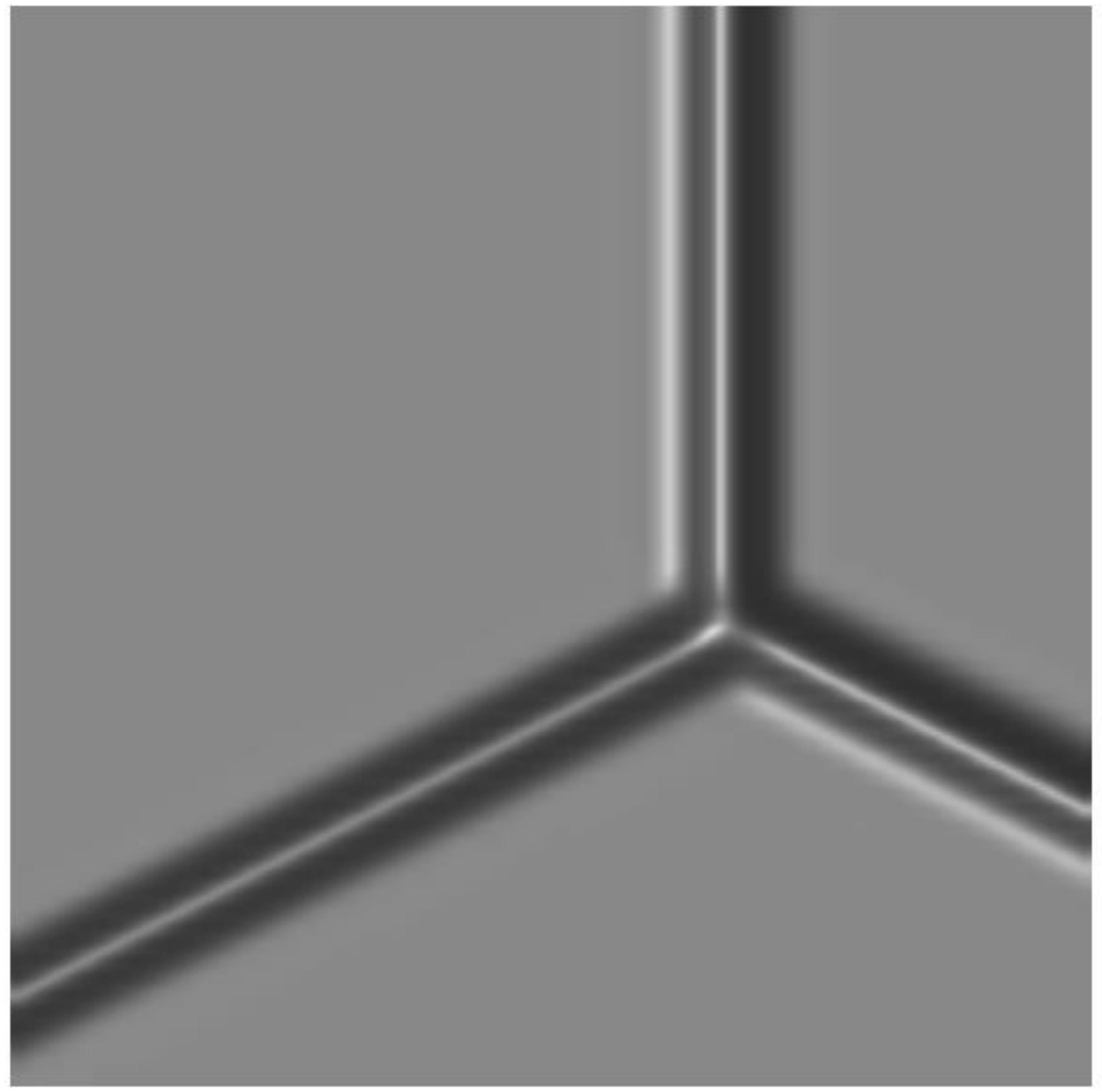
Filtering after shading:
Two adjacent geometric facets and normals

Filtering lighting does not produce surface curvature

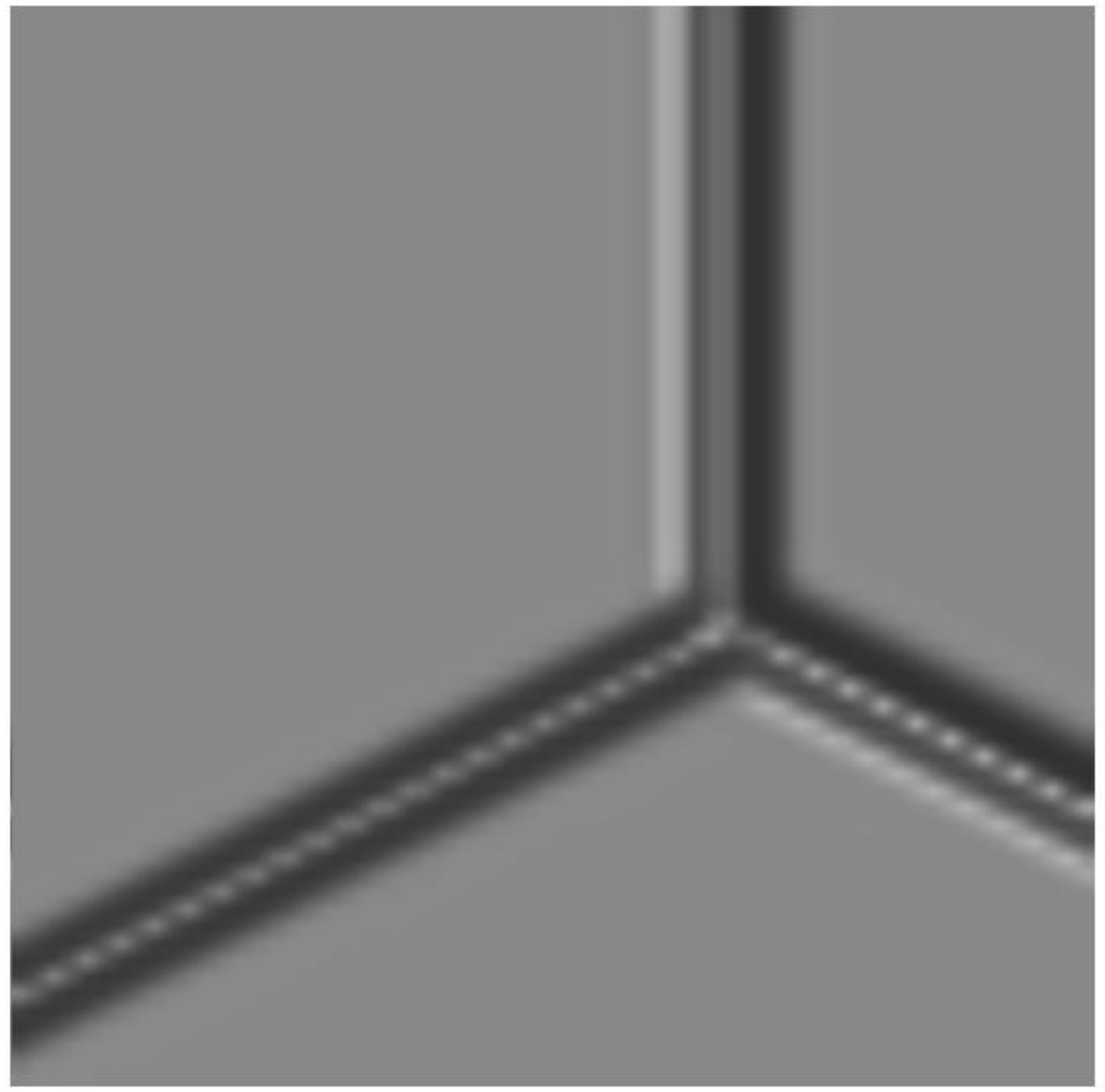


Note: it's neither "good" or "bad", depends on the intent / assumption
But it changes the appearance – artists need to be aware!

Worse example – magnification aliasing



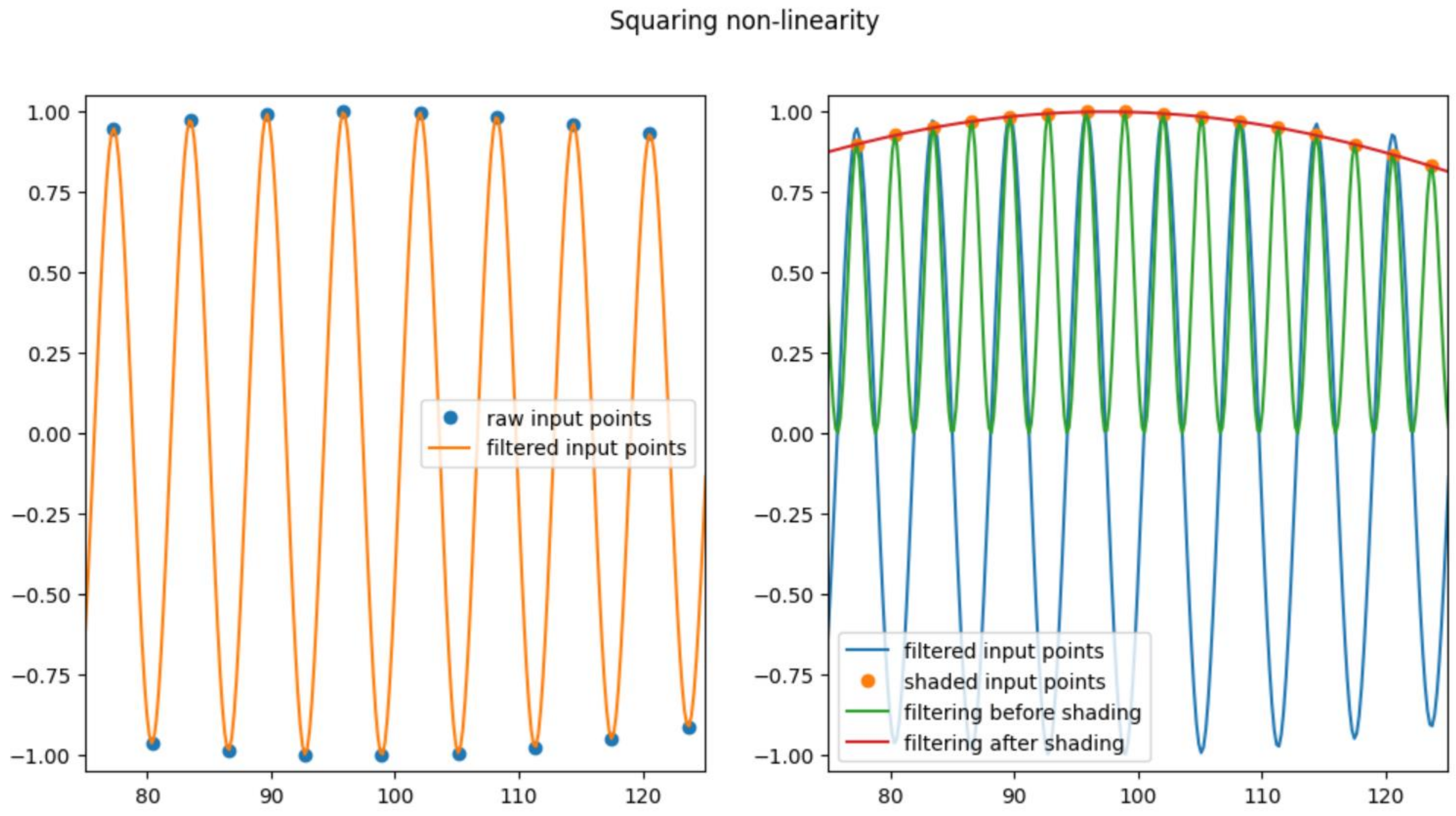
Filtering before shading



Filtering after shading

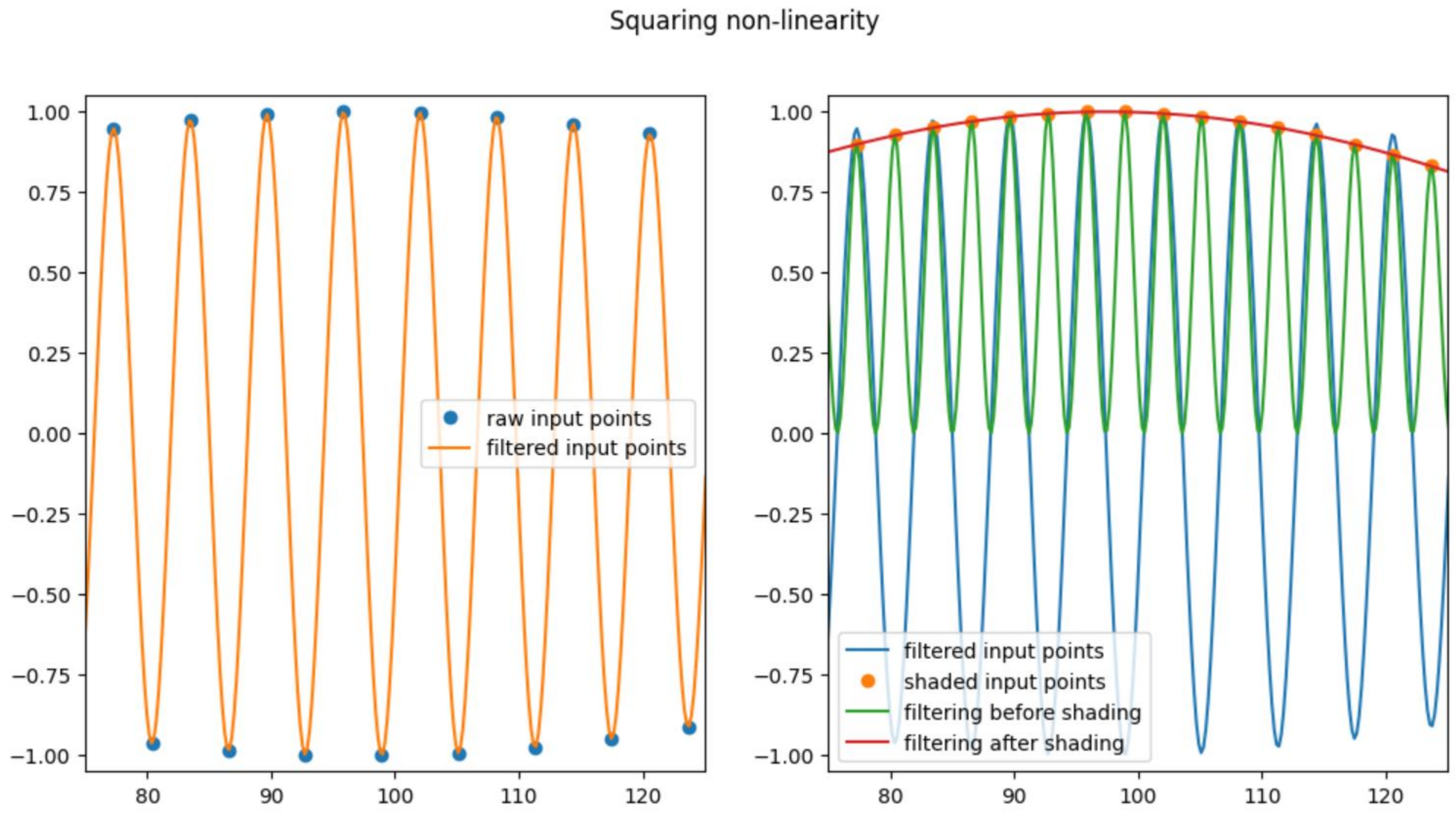
Non-linearity introduced aliasing

- Any non-linearity always introduces new, higher signal frequencies (“harmonics”)
- When applied to discrete signals… **those frequencies alias immediately**



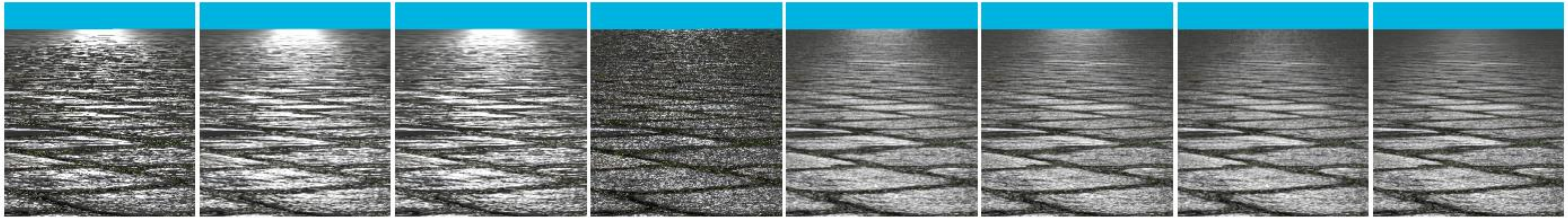
Non-linearity introduced aliasing

- 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

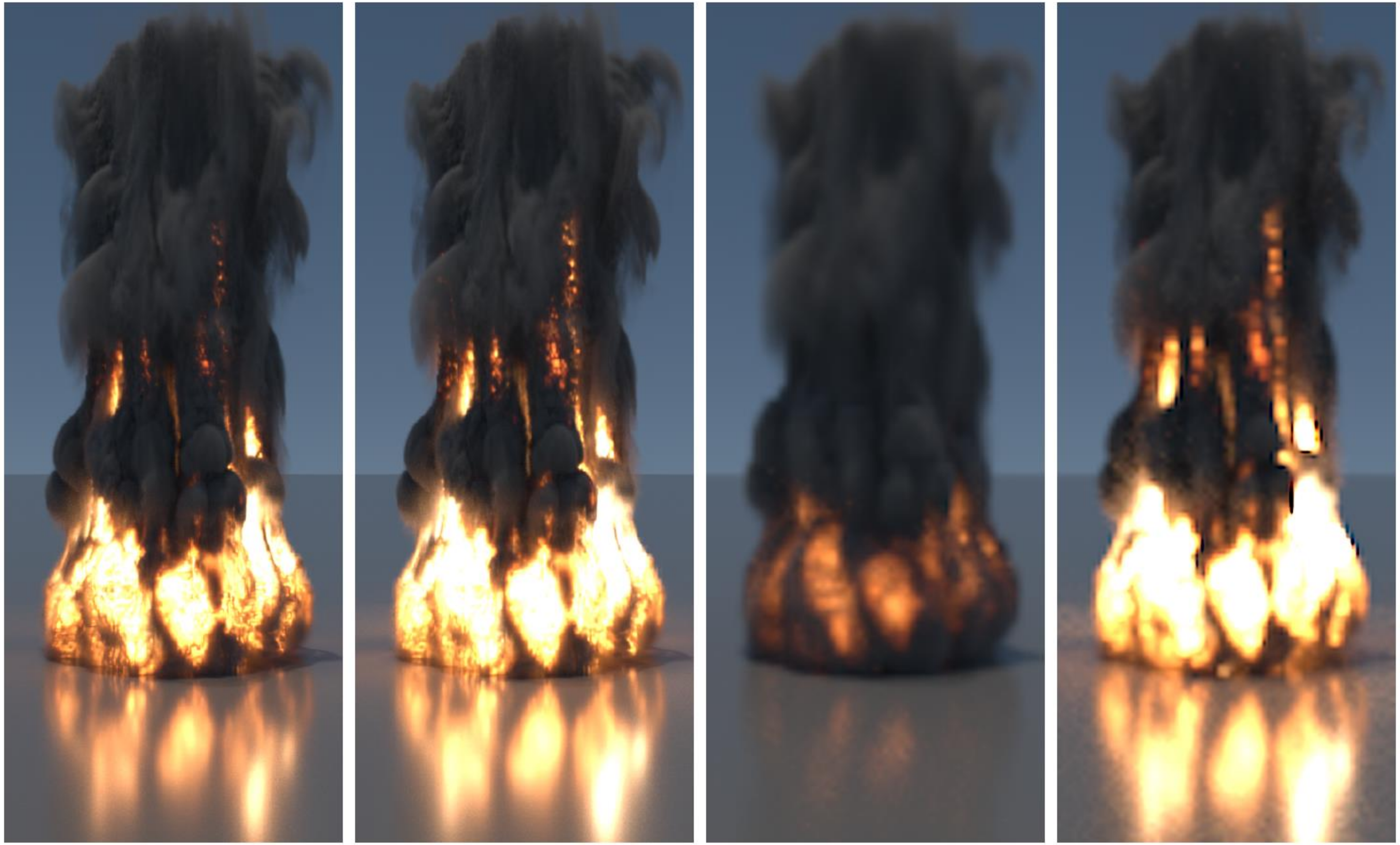
Appearance preservation – real time



(a) Filter Before 1 spp | (b) Filter Before 1 spp + DLSS | (c) Filter Before 1024 spp | (d) Filter After 1 spp | (e) Filter After 1 spp + DLSS | (f) Filter After 1024 spp | (g) Filter After LOD 0, 1 spp, DLSS | (h) Reference 1024 spp

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!

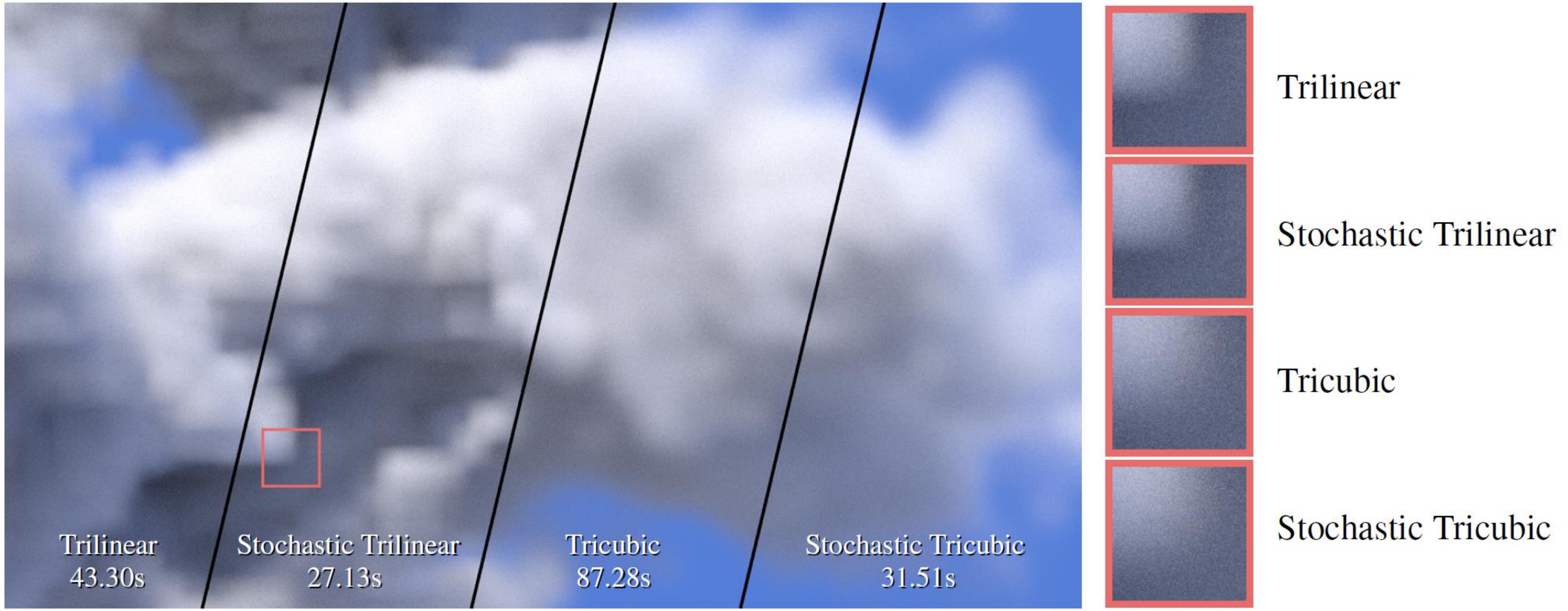
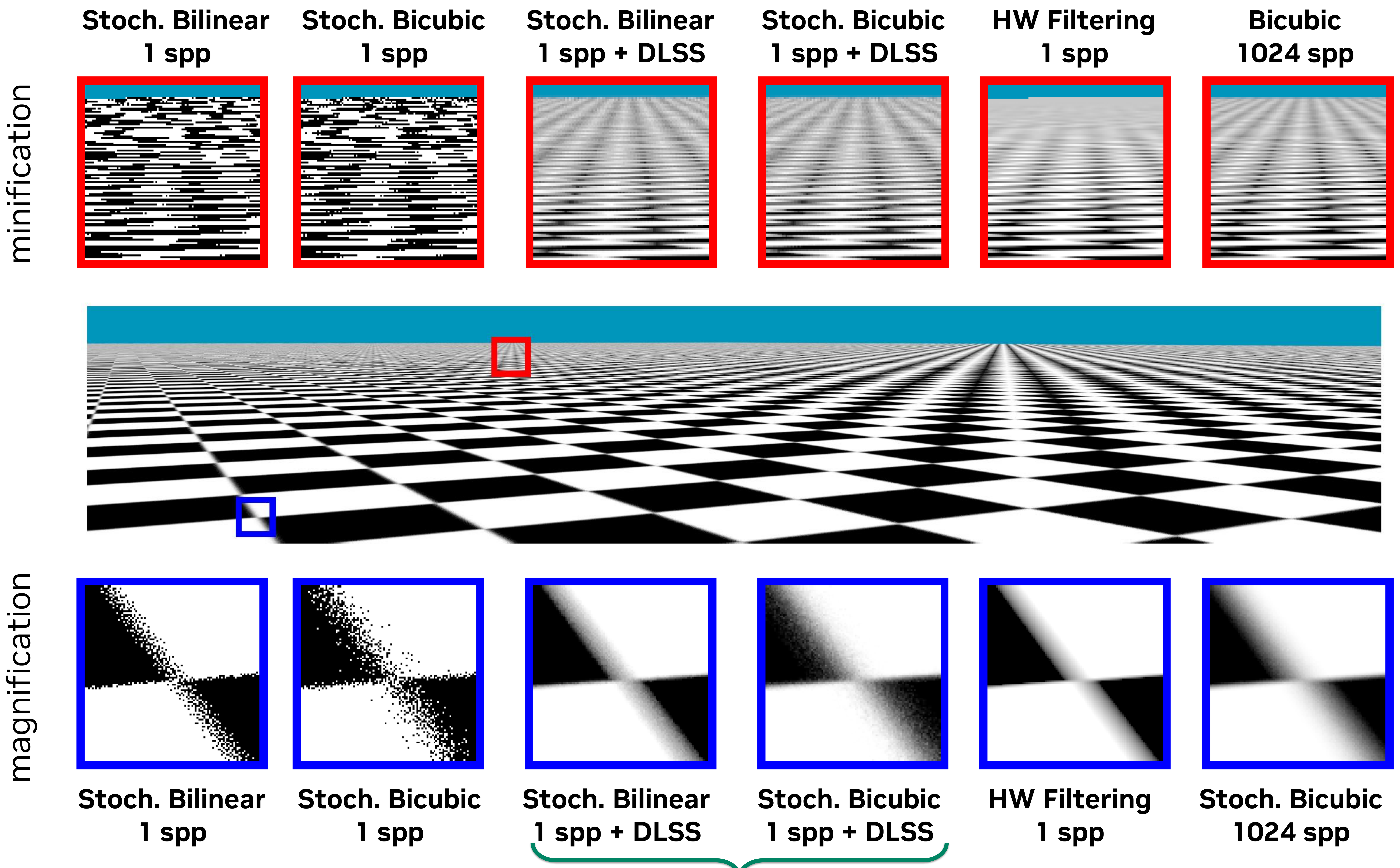


Figure 1: A section of the *Disney Cloud* scene rendered with path tracing. With this close-in viewpoint, trilinear filtering leads to blocky artifacts in the image. Tricubic filtering gives a much better result, though requires 64 voxel lookups into the *NanoVDB* representation. Stochastic filtering 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 *pbrt-v4* running on an NVIDIA 4090 RTX GPU, rendering at 1080p with 256 samples per pixel.

Minification & Magnification
DLSS + STBN Temporal Stability Test

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)

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**

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**! ☺

NVIDIA.

Thank you!
Questions?

Bonus Slides

Enable Custom Texture Compression/Storage Algorithms

Minification vs Magnification 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

Upsampling in sRGB

Upsampling in linear

Figure credit*: A Fresh Look at Generalized Sampling,* Diego Nehab and Hugues Hoppe

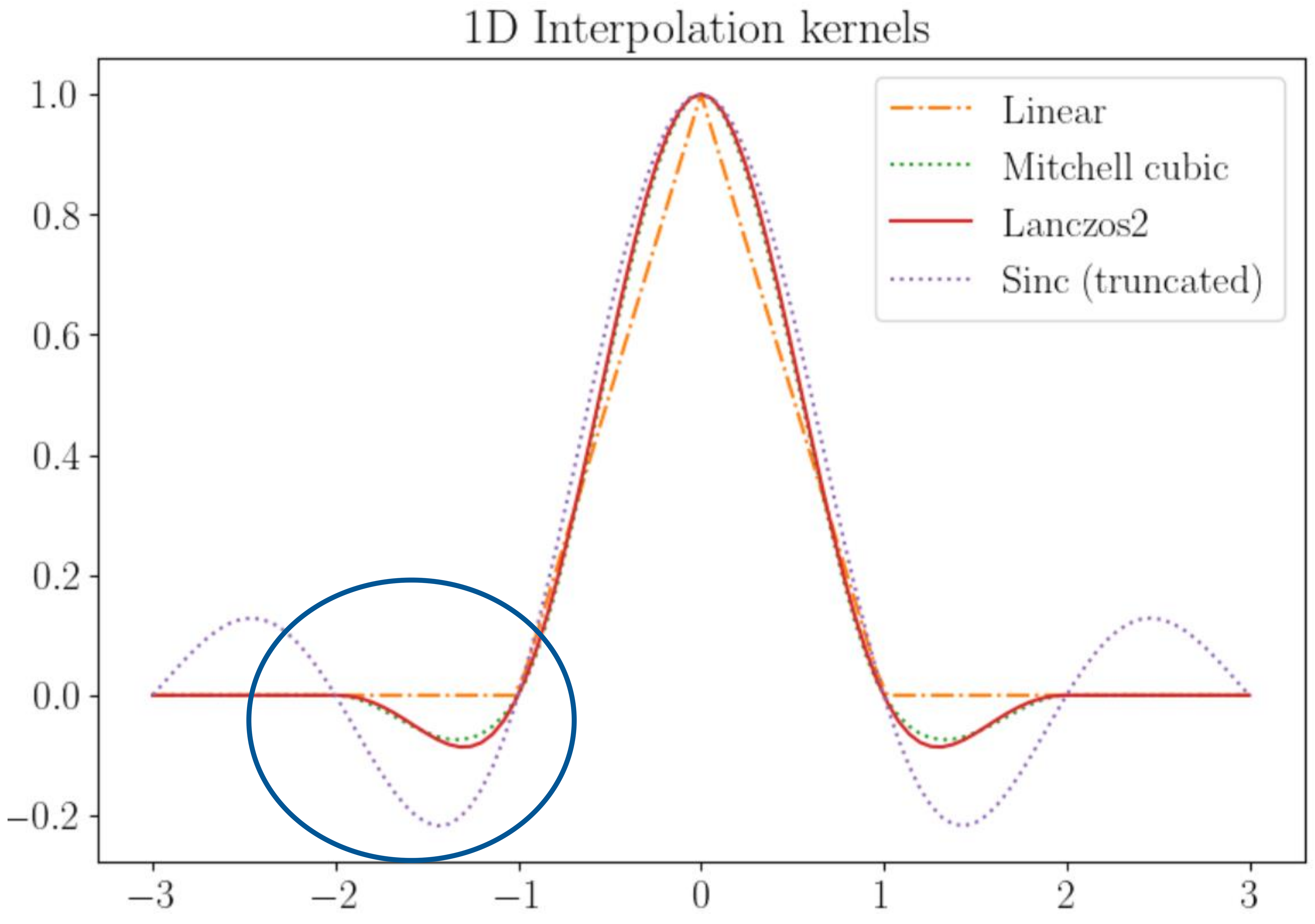
Sampling Texture Filters – Negative Lobes

- 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…

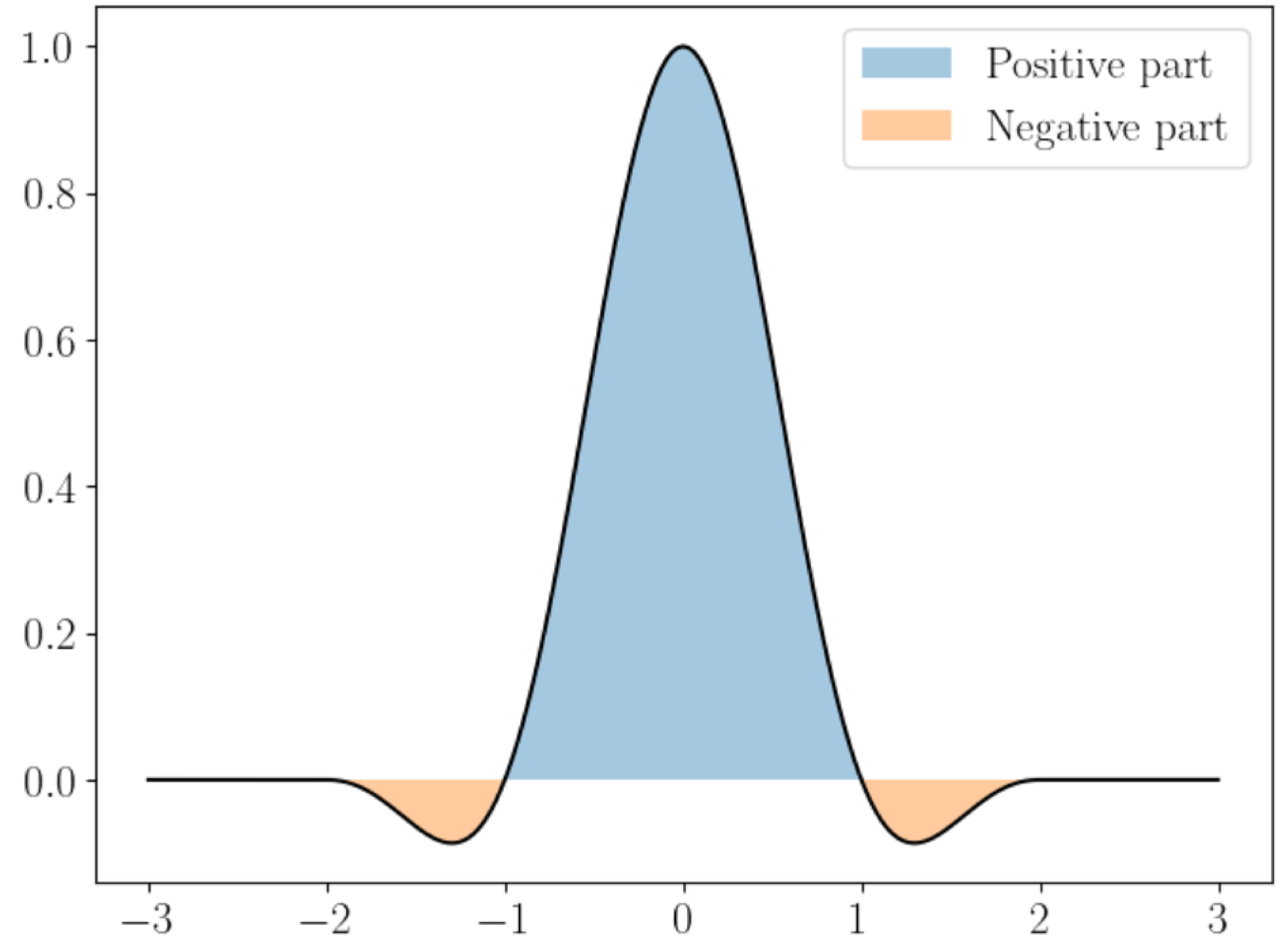


Sampling Texture Filters – Negative Lobes

- 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**

- **I**mportance sample the positive and negative parts separately
- Always evaluate two samples
- Weight sum always positive
- 2X the cost
- Low variance

$$\langle F \rangle = \sum_i w_i^+ t_{j^+} - \sum_i w_i^- t_{j^-} .$$

Positivization – Results

Bilinear

Mitchell