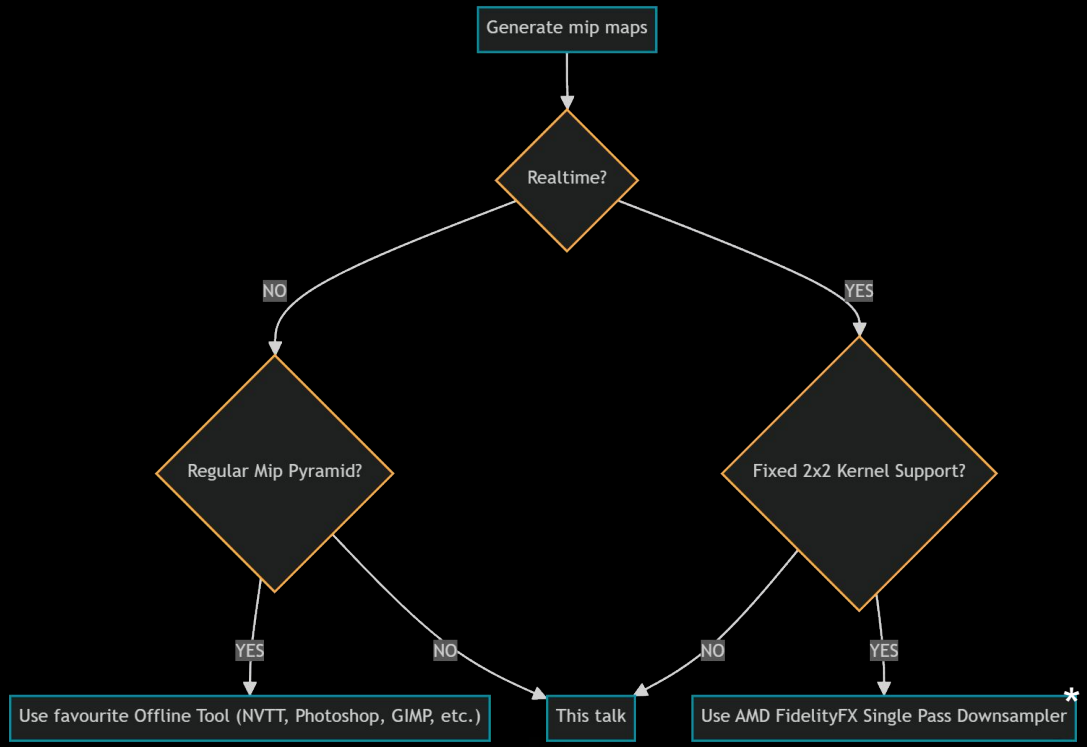
Vulkanised 2024

The 6th Vulkan Developer Conference
Sunnyvale, California | February 5-7, 2024

Perfect Mip-Maps with Poly-Phase Filtering Convolution on the GPU

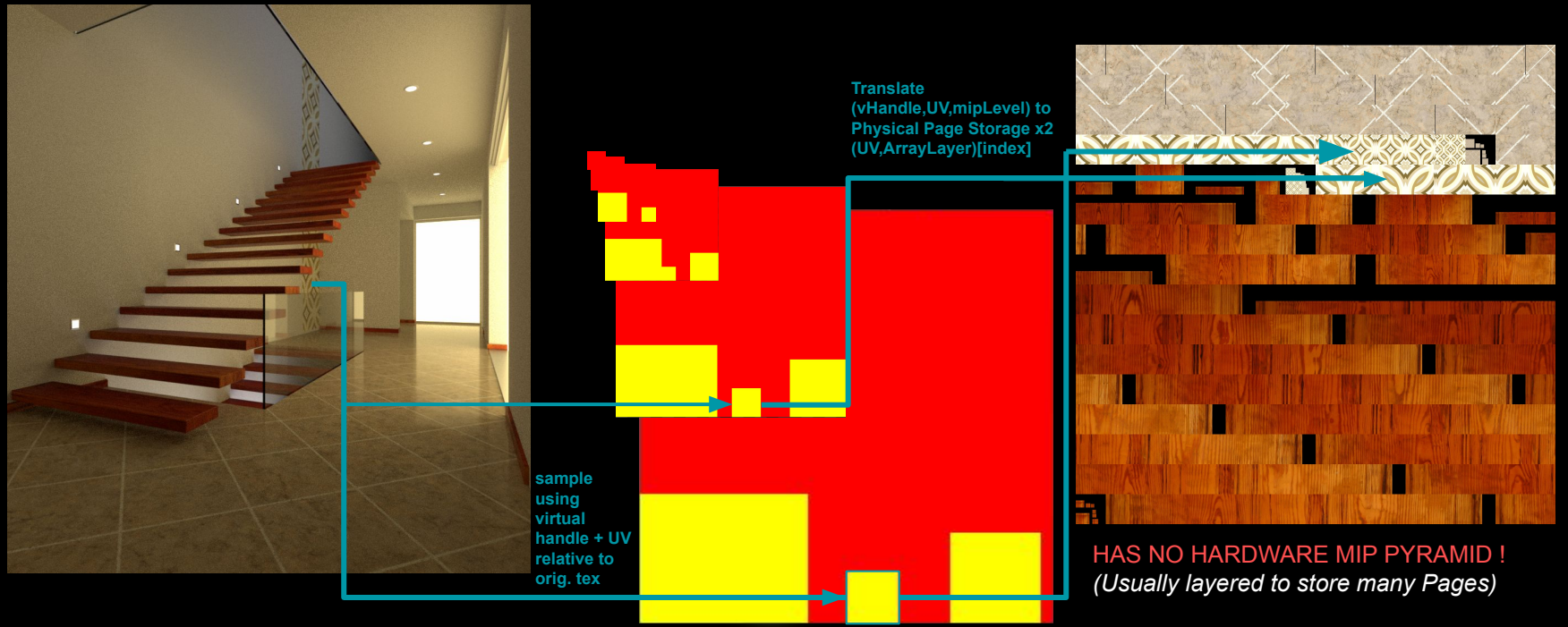
Arkadiusz Lachowicz, DevSH Graphics Programming
Lead Build System and Test Engineer
a.lachowicz@devsh.eu

Why should I care? No glGenerateMipMap!



What caused us to explore this problem? (1/3)

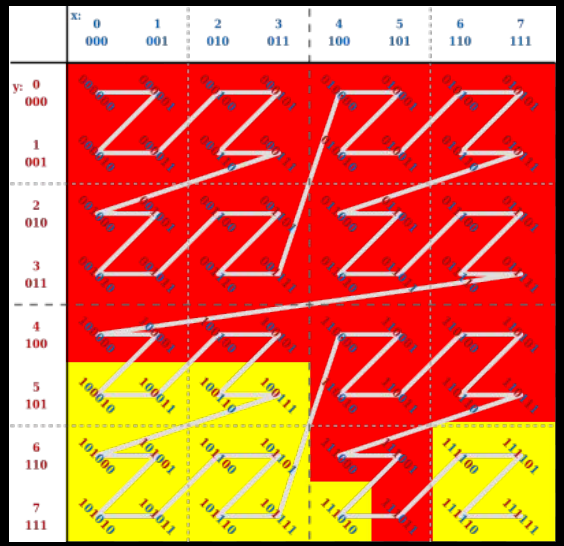
- Runtime loading of source texture assets, our Renderers are used in scene editors with rapid prototyping
- Building our own Asset Pipeline which can compile/bake Image Assets offline as well
- But mostly, having to implement Virtual Texturing to Path Trace before bindless became ubiquitous:



What caused us to explore this problem? (2/3)

Virtual Texturing imparts many constraints on the implementation:

- Each Page needs a border equal to half of the Anisotropy Level, so:
  - Mip levels are disjointly stored
  - No hardware mip-mapping
  - Manual trilinear filtering
  - special packing of the mip-tail *(levels smaller than a single Page)*
  - "Special" staging of resident pages - *nbl::asset::CPaddedCopyImageFilter*
- Individual Textures reserve as contiguous rectangles in the Page Table
  - The Page Table has a full mip-pyramid
  - Interpret virtual coordinates as Morton Codes on a Z-order Curve with aligned allocations* → Implicit QuadTree
    - Mip-Maps need to Round Up to Power of Two
    - Textures smaller than half a Page need to be upsampled!



What caused us to explore this problem? (3/3)

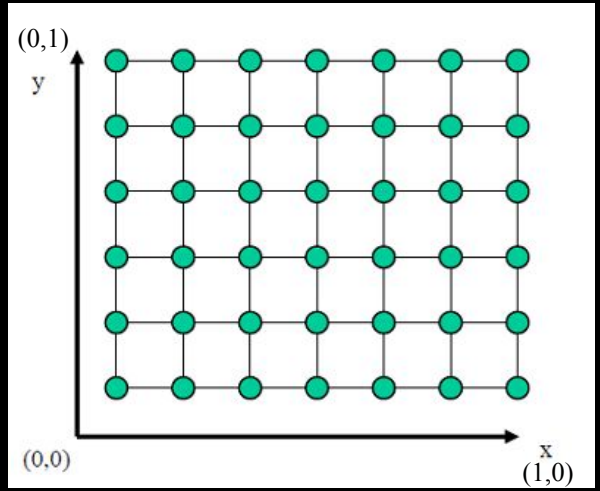
- Nvidia actually proposed Non Power of Two textures that ceil their mipmap resolutions, for OpenGL in early 2000s
  - Mipmaps must be recomputed from scratch and your favourite HQ mip generation software doesn't make these
    - *Regular Mip Chain: 1023 → 511 → 255 → 127 → 63 → 31 → 15 → 7 → 3 → 1*
    - *Our Mip Chain: 1023 → 512 → 256 → 128 → 64 → 32 → 16 → 8 → 4 → 2 → 1*
    - Our NPoT Textures always have one more mip level!
- Native Sparse Partially Resident Textures are still not great, so we might revive this VT implementation in Vulkan
  - Bind Commands are woefully **slow, especially on Windows**
  - Sparse Images have the same **Maximum Resolution Limits as non-Sparse** *(so no Megapixel per axis Textures)*
- **TL;DR** `nonUniformEXT` and Descriptor Indexing is a gift to humanity!

Image processing Recap

An image is usually represented as a set of texel values, aligned on a rectangular grid.

- Texels represent the value of a function at a particular sample location
- For simplicity of proof derivations later on in the talk, we will think of the functions' domains being the Normalized Texture Coordinates $[0,1]^2$
- Often the image is a sampling of an actual continuous signal with infinite frequencies (e.g. photograph, rendering, etc.)

Note that an image can also be 3D, then we need to add a Z axis.



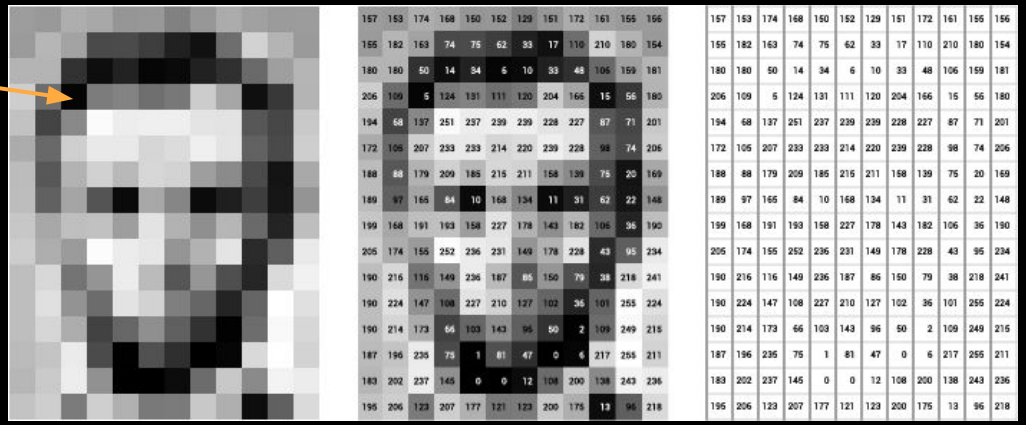
https://www.tutorialandexample.com/wp-content/uploads/2019/12/Image-Processing.png

An image can be considered as a function

- While you might think of an image A being an NxM discrete Matrix representing the 2D grid of an image, one can also define a 2D grayscale NxM image as, let $f : [0, 1]^2 \rightarrow R$ then

$$f(u,v) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} A_{n,m} \delta(u - \frac{2n+1}{2N}) \delta(v - \frac{2m+1}{2M}) \text{ which implies } \int\int_{-\infty}^{+\infty} |f(u,v)| \, du \, dv < \infty$$

Repeat After me:
"Texels are not Boxes!"



https://ai.stanford.edu/~syyeung/cvweb/Pictures1/imagematrix.png

Convolution Theorem Revisited (1/3)

- For continuous functions on an infinite domain we have

$$F\{f\}F\{g\} = f \circ g \quad \text{and} \quad fg = F\{f\} \circ F\{g\}$$

  Properties by corollary

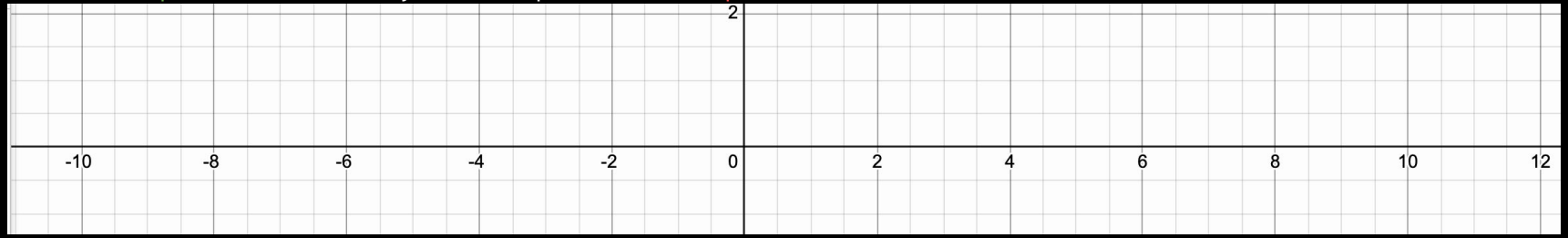
  - Shift $F\{f(t-t_0)\} = e^{-j\omega t_0}F(\omega)$
  - Linearity $F\{af+bg\} = aF\{f\} + bF\{g\}$
  - Stretch $g(x) = f(kx) \Leftrightarrow F\{g\}(w) = F\{f\}\left(\frac{w}{k}\right),\ k \in \Re$
  - Differentiation $(f \circ g)' = f' \circ g$

- For a continuous function with a domain [Min,Max], we can extend it to infinite by making it periodically repeating outside
  - This makes the Fourier Spectrum discrete (a Fourier Series) as non-integer-multiple frequencies will integrate to 0
    - *"Spawning periodic copies" is equivalent to Convolution with an Infinite Train of Dirac Deltas spaced Max-Min apart, by convolution theorem this is same as multiplication of the finite function's Spectrum with the Spectrum of the Dirac Train which is another Dirac Impulse Train but with* $\frac{1}{Max-Min}$ *spacings*
  - but not finite, as you can still have infinitely high frequency (fine) details

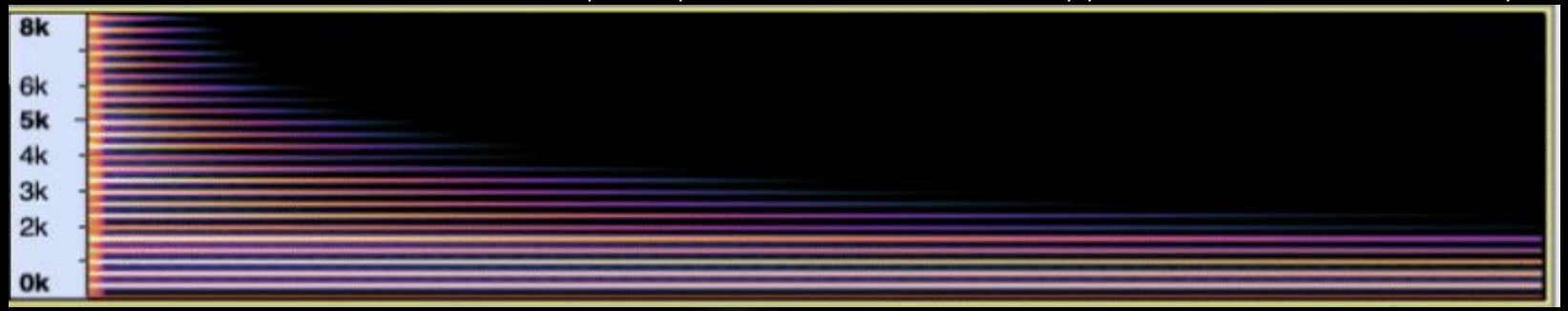
Convolution Theorem Revisited (2/3)

To go back to an a-periodic function with a finite domain and zero outside, we can construct its spectrum from a periodic version

- The a-periodic function is just a multiplication of the periodic function with a Box-Function window

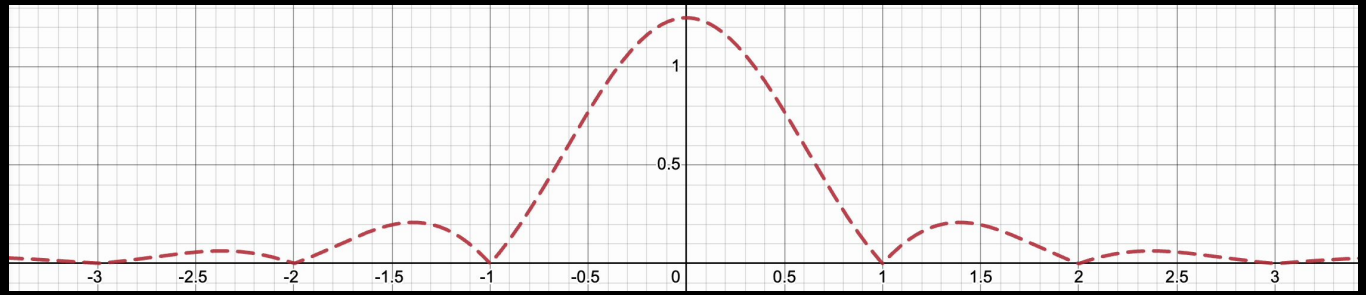


- The multiplication in the Spatial Domain is also a convolution in the Frequency Domain
  - The Spectrum of a Box function is the Sinc function with inversely proportional width
- Convolution of Sinc with a series of Delta Impulses produces streaks, which is why you can often see this in Audacity

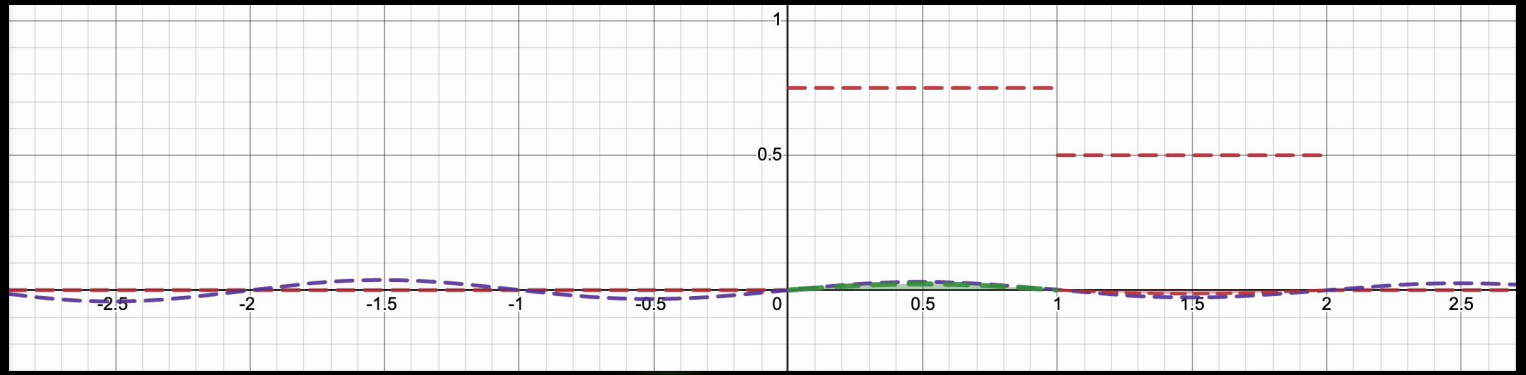


Convolution Theorem Revisited (3/5)

- We can also make our domain limited periodic function's Spectrum discrete by band limiting the function
  - Let us multiply the Spectrum of the function with a Box Function to cut off high frequencies

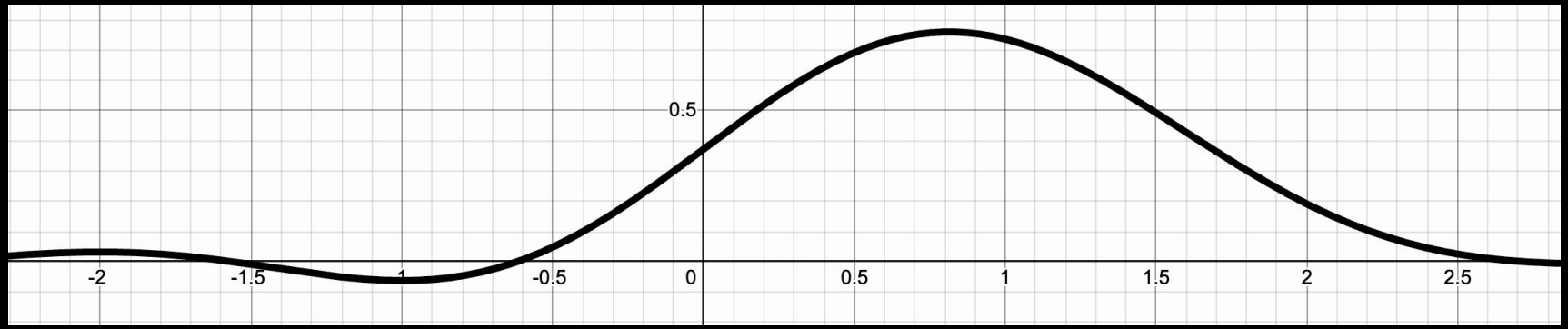


  - This is equivalent to convolving the original function with Sinc in the Spatial Domain (a Low-Pass Filter)



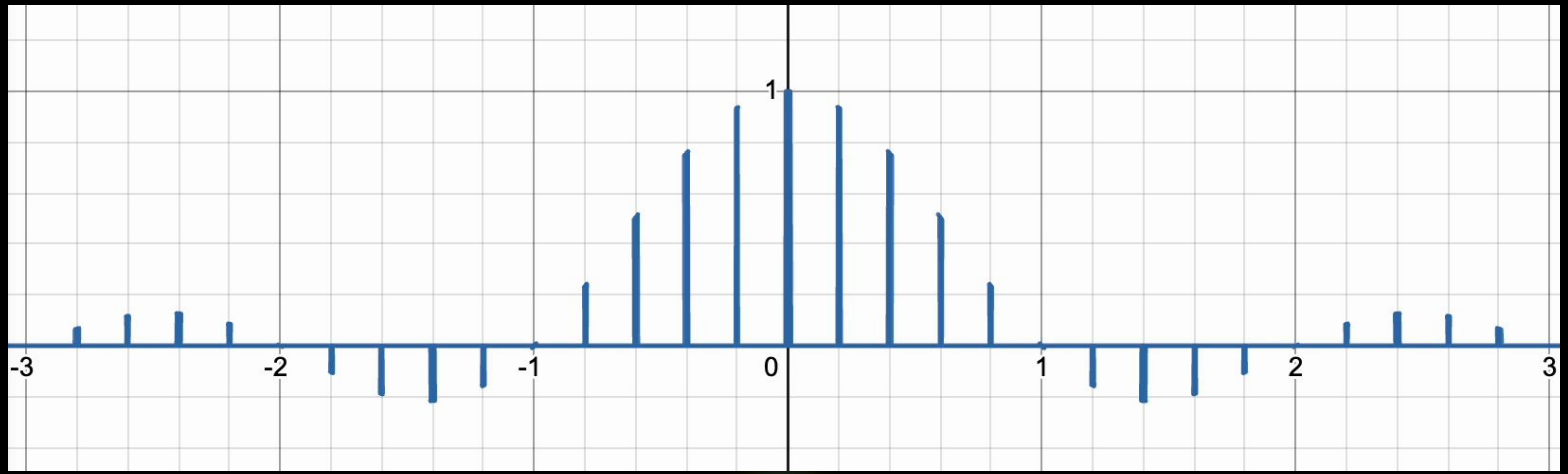
Convolution Theorem Revisited (4/5)

- Now we can study what happens to the Spectrum of the Domain and Bandwidth Limited Function as we sample it
  - Spatial Domain Multiplication with a Dirac Comb a.k.a Shah Function samples the function in a ~~roundabout~~ $\frac{1}{Period}$ way
  - The Spectrum of a Dirac Comb $III_T(x) = T\sum_i \delta(x - Ti)$ is a Dirac Comb with inversely spaced impulses $III_{\frac{1}{T}}(x) = \frac{1}{T}\sum_i \delta(x - \frac{i}{T})$



Convolution Theorem Revisited (5/5)

- This "spawns copies" of the original spectrum $\frac{1}{Period}$ frequency units apart
  - Which means the Spectrum of your Sampled Function is Periodic
  - And if you didn't Bandlimit your function before sampling, the copies overlap causing ALIASING!
- Always think of your image's Spectrum and Spatial representations being Discrete and Periodic (repeating)

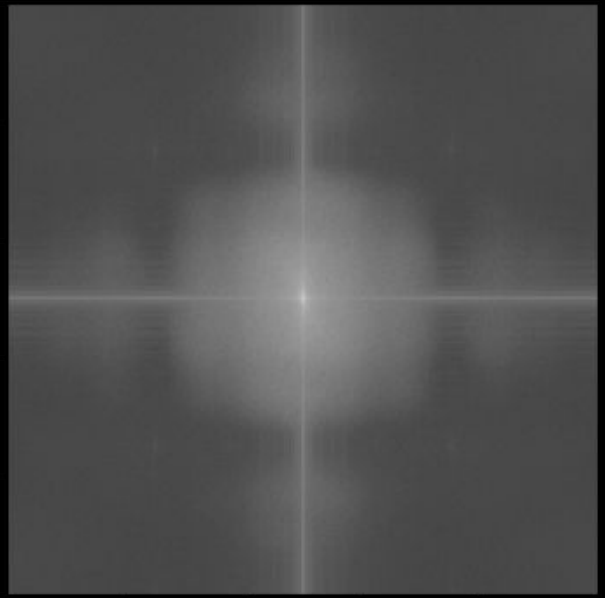


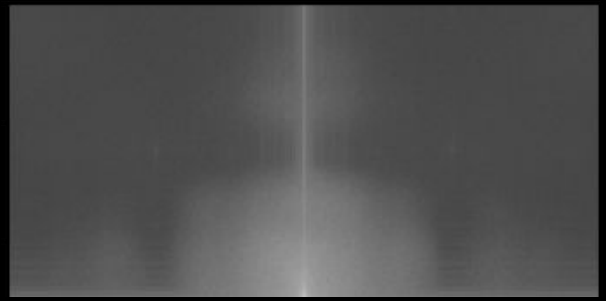
WARNING: Power Spectra Diagrams are Lies!

A Fourier Transform produces Complex Numbers, and a Discrete Fourier Transform produces N outputs for N inputs.

Conservation of Entropy suggests we can't have dimensional inflation, so for a Real valued Input, only half of the Outputs should be independent of each other.

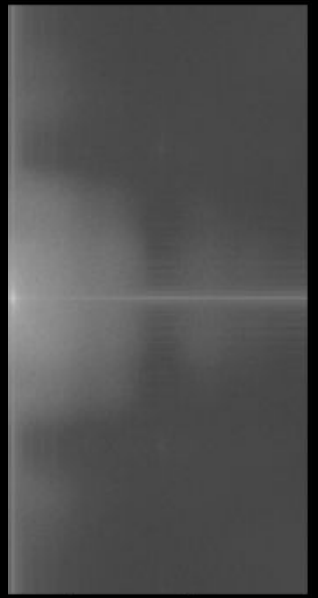
Which is actually the case, as a Fourier Transform of a Real Signal is an Even-Symmetric Function.





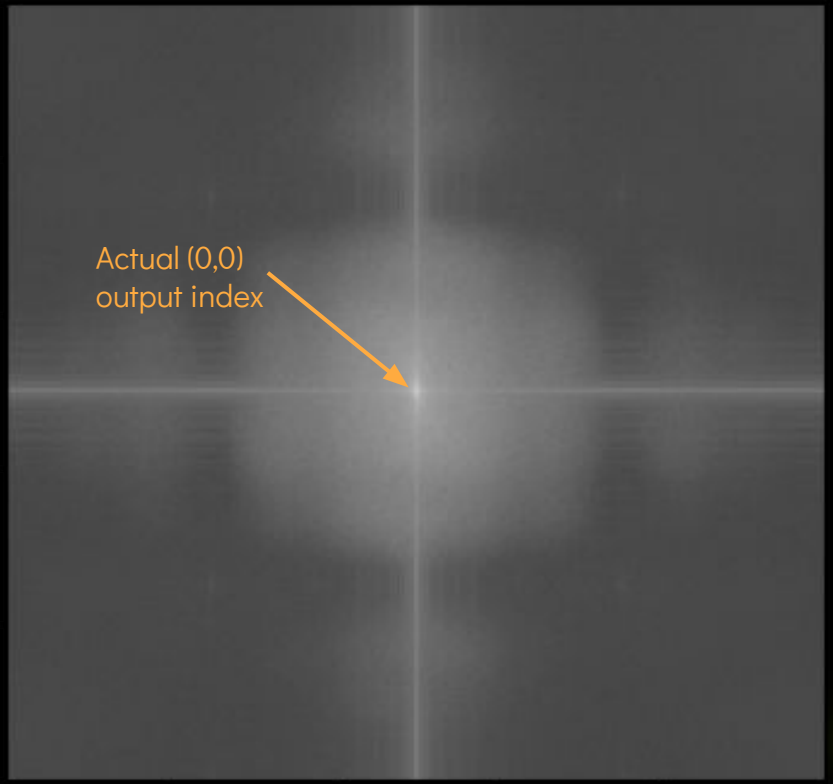
This is not what a DFT spits out if you try to visualize your output array as an image.

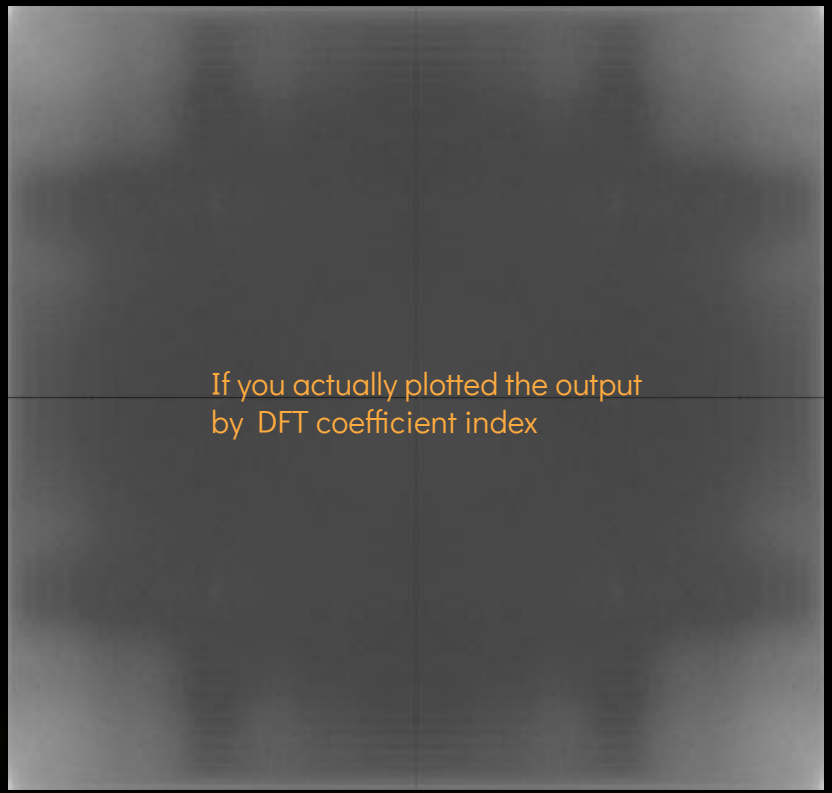
Any 180 degree section around the center can be used to obtain the other half with conjugate symmetry.



Spectrums are usually illustrated with a cyclic shift

Take this into account for when we're talking about borders, padding, and zooming on the next slides

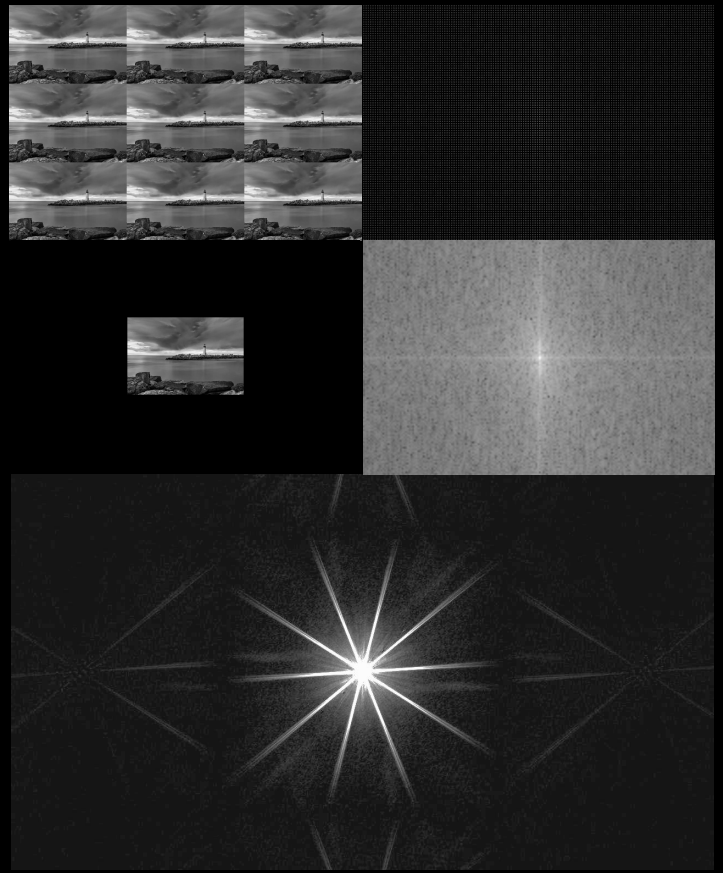




Spectral Domain Manipulation: Zooming

Here we mean warping w.r.t. the normalized UV coordinates, the Stretch Property of the Fourier Transform applies.

- Zoom out, resolution increases → careful choice of new padding pixels outside original Spatial Domain required
  - REPEAT equivalent to naive Spectrum Resampling →
  - CLAMP_TO_BORDER is the above + Sinc convolution → *(you basically get an interpolated spectrum)**
- Zoom in, borders get cropped → original wavelengths migrate closer to the center (frequency 0), and Spectrum needs fewer samples, so again, resampling the Spectrum!
  - Cropping is a Spatial Domain multiplication with Box, so again a Spectral Domain convolution with Sinc!
  - *Results in a "Bandlimited" Spectrum, lossless to naively resample as original can be reconstructed.*



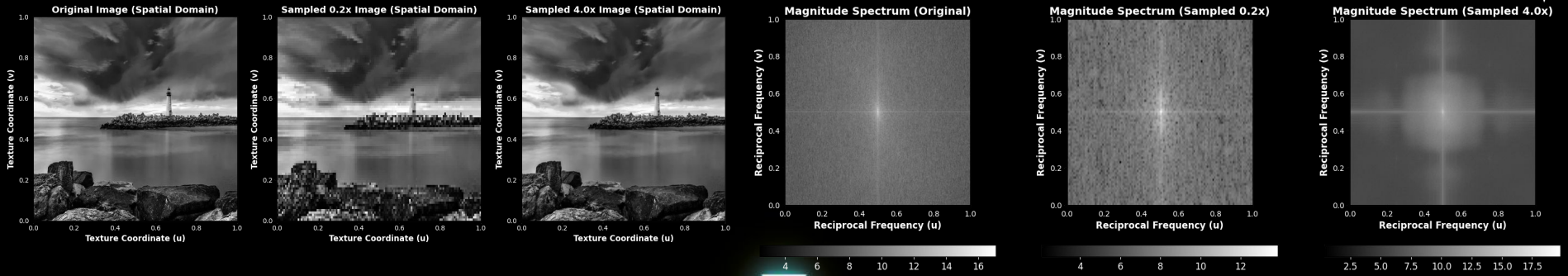
**Our FFT Bloom Implementation actually uses this to only do the FFT of the Lens Kernel once and adjusts its relative size freely. Notice the ringing in Spatial Domain due to upsampling the Spectrum with a Tent Filter (HW Bilinear) instead of Sinc. Contrast and Exposure artificially increased to showcase ringing.*

Spectral Domain Manipulation: Resampling

Period Length in UV space **won't change** unless resolution does!

- Upsampling, zooms out on the Spectrum → allows "higher frequency" content, empty border regions in Spectrum
  - Choice of Spatial interpolation affects the border contents; Sinc leads to zeros, Nearest Neighbour to ringing
- Downsampling, zooms in on the Spectrum → migrating the frequencies "higher", outside of the original Domain
  - They don't just disappear, if you don't band-limit, they'll wrap-around onto the lower ones = Classical Aliasing
    - *Sad news: Ideal Band-limiting kernel (Sinc) rings in practice because of Gibbs Phenomenon*

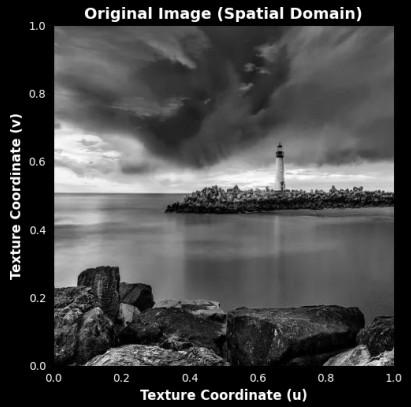
I hope you're starting to see a pattern, Zooming in Spatial is equivalent to Resampling in Frequency and vice versa!

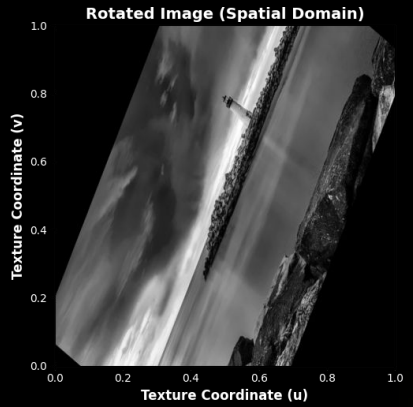


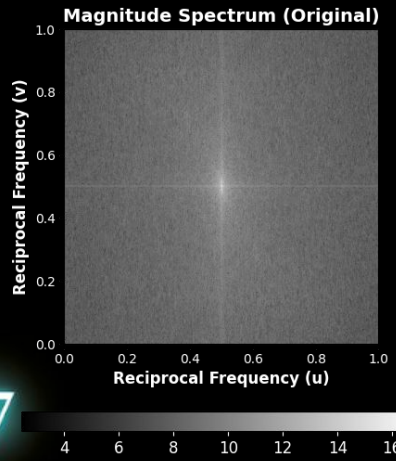
Spectral Domain Manipulation: nD Rotation (n>1)

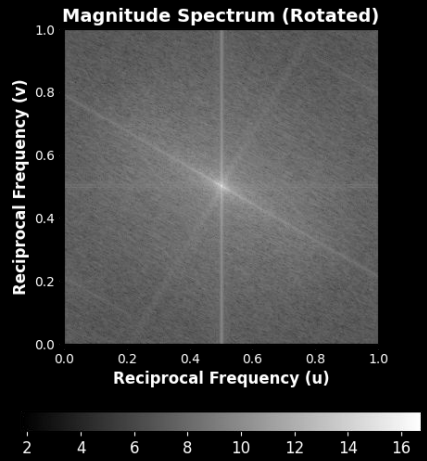
- A continuous unbounded spectrum can be simply rotated, but a Grid's samples won't re-align for most rotations
- Artefacts introduced by unwanted frequencies being removed or added:
  - Original Corners may be clipped outside domain
  - New Corners may be left uncovered by the original image inside the domain

So applying a rotation to a Discrete Spectrum will not produce an IDFT which equals the rotated discrete function.









Why go into so much detail?

It's obvious we don't resample images with DFTs, and there are reasons not to do so:

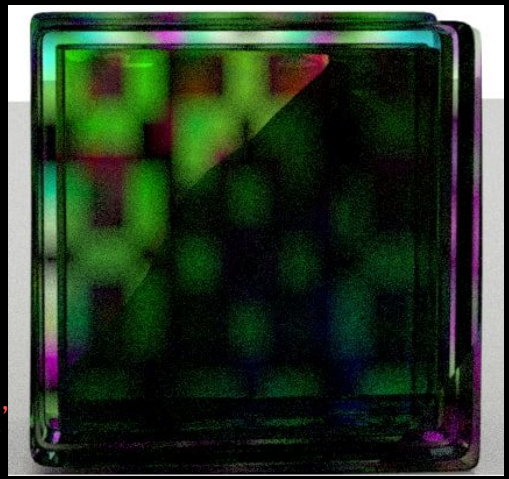
- most nD kernels you'd want to use are separable anyway *(why we only use FFT Convolution for Bloom/Flare)*
- log(n) is still huge in comparison to a separable kernel's gather footprint (a.k.a. support)
- CPU / GPU caches love naive sequential access patterns, while FFT scrambles outputs like a Van der Corput sequence

We already shipped the Path Tracer using Nabla in 2021 with the Virtual Texturing using a CPU Software Blit Image Filter to generate the custom Mip Pyramid with arbitrary 1D Resampling Kernels:

- Dirac
- Box
- Triangle / Tent
- Gaussian / Normal Distribution
- Mitchell
- Kaiser Windowed Sinc (assumed non-windowed)

Then one day we encountered this little bug →

Remember "Textures smaller than half a Page need to be upsampled to Page Size!"
This is what Upsampling of a 1x1 images to 64x64 with a Mitchell kernel looks like



What went wrong (1/2)?

When you are doing a numerical convolution of an image f with a kernel k you expect their spectra to multiply to produce an image $g = f \circ k$ especially if the sampler used for out-of-domain taps of f is REPEAT (*periodic*).

So let's apply the rest of "Convolution Theorem":

- f is periodic but discrete, so the Spectrum of f Discrete but, it is Periodic and not Finite because it's sampled
    - Spectrum of f has a period of width $Input\ Resolution$
- k is usually not periodic or discrete, but is an even and decaying function with width proportional to $\frac{1}{OutputResolution}$
    - Spectrum of k has width proportional to $OutputResolution$ is Continuous, A-Periodic, and probably Infinite*
- Spectrum of $f \circ k$ is Discrete, not Periodic, and probably Infinite*

What went wrong (2/2) ?

- but when you're changing the sample locations there's an implicit multiply $g = (f \circ g) III_{\frac{1}{OutputResolution}}$
- which is a convolution in the Frequency Domain $F\{g\} = (F\{f\} F\{k\}) \circ III_{OutputResolution}$
  - You now have copies of $F\{f \circ k\} == F\{f\} F\{k\}$ spaced $OutputResolution$ Hz apart in the Spectrum
  - Unless k has Sinc with width less than $\frac{1}{InputResolution}$ convolved into it, copies will overlap → **Aliasing**!

Let's think about what we actually want

One could say we get fluctuations in the output because we have gaps in the input.

S. Guthe and P. Heckbert in "Non-power-of-two mipmap creation" actually address this, by having a separate Reconstruction and Resampling kernel

$$g = ((f \circ r_{construct}) \circ r_{sample}) III_{\frac{1}{OutputResolution}}$$

The Reconstruction has a width of $\frac{1}{InputResolution}$ and the Resampling has a width of $\frac{1}{OuputResolution}$

Implemented in Nvidia Texture Tools but Tent Filter hardcoded for both, while we can present the full theory:

- Convolution is Associative, so you can write $g = f \circ (r_{construct} \circ r_{sample}) III_{\frac{1}{OutputResolution}}$
- Spectrum of $r_{construct} \circ r_{sample}$ get "double windowed"

  so width roughly proportional to $\min(InResolution, OutResolution)$
  - *Can be forced to have a finite Domain & zero outside* $InputResolution$ *by making* $r_{construct}$ *a Sinc*
- Now your Spectrum copies after sampling are far less pronounced → far less Aliasing!
  - No copies at all if you make $r_{construct}$ a Sinc, but might introduce faint ringing... still okay for LDR

TL;DR Lots of theory for a simple fix

On our legacy Path Tracing branch we fixed the original Single Convolution Blit by using a g which has a width of $\frac{1}{\max(InputResolution, OutputResolution)}$

TL;DR Lots of theory for a simple fix

On our legacy Path Tracing branch we fixed the original Single Convolution Blit by using a g which has a width of $\frac{1}{\max(InputResolution, OutputResolution)}$

That discussion of Fourier Spectra was really needed...

TL;DR Lots of theory for a simple fix

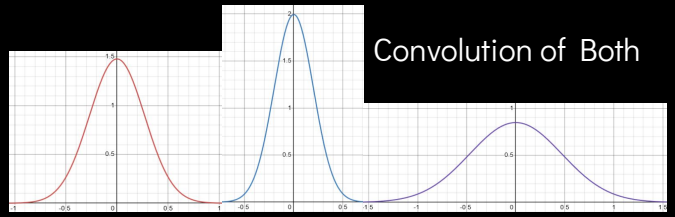
On our legacy Path Tracing branch we fixed the original Single Convolution Blit by using a g which has a width of $\frac{1}{\max(InputResolution, OutputResolution)}$

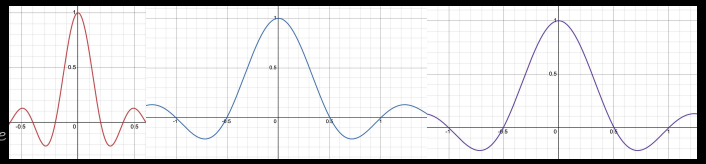
That discussion of Fourier Spectra was really needed....

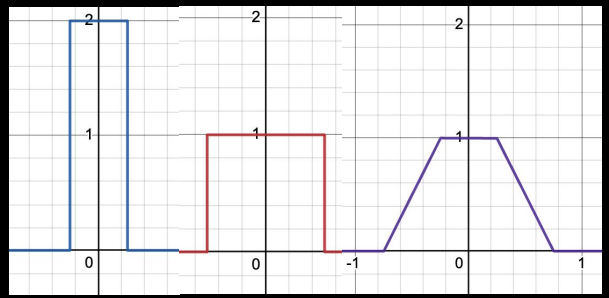
Actually we did implement a proper "high-performance" implementation eventually, where we can use a g which is convolution of two arbitrary width functions that can be analytically computed:

- *Gauss on Gauss* → *Single Gaussian with* $\sigma_g = \sigma_{r_{construct}} + \sigma_{r_{sample}}$
- *Sinc on Sinc* → $Sinc_{g(x)} = Sinc(\min(A, B)x)$
- *Box on Box* → *a weird trapezium*

But then we went further and actually implemented the thing properly so you can mix'n'match the kernels and the above analyticals got faster as well!







The Image API in Nabla

- Take an input ICPUImage and produce output image given the filter type, it's state, and if required, preallocated scratch memory for intermediate operations
  - a recipe for creating VkImage and filling its contents with copies of regions from ICPUBuffer
  - being an asset makes its operations and mutations independent of any GPU present on the system
  - sparse friendly contents specced as a series of region copied from ICPUBuffers
  - all 150+ non-planar VkFormats decode and encode implemented in software, flexible runtime texture converting
  - format conversion with template & runtime Swizzling and Dithering available

The Image Filter API in Nabla

- Various filter types supported
  - CCopyImageFilter, CPaddedCopyImageFilter, CBufferToImageCopyImageFilter copy filters
  - CSwizzleAndConvertImageFilter filter with optional and custom dithering (Tileable 2D Blue Noise as default)
  - CBlitImageFilter convolution filter with Polyphase LUT optimization
  - CMipMapGenerationImageFilter, CNormalMapToDerivativeFilter, CSummedAreaTableImageFilter and more composed or derived by filters mentioned above
- Flexible templated state struct containing IO image data

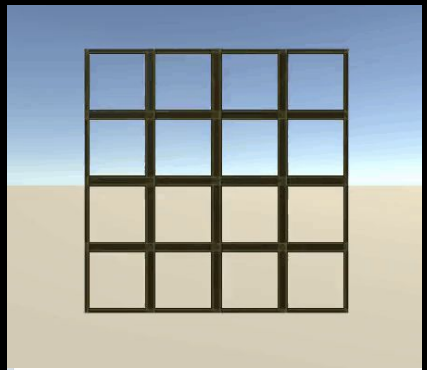
Nabla's CBlitImageFilter

- Composes the Swizzle, Dither and Convert filters
  - Output can have different format than input
- Does a separate pass over each axis
  - Can even do convolutions in 3D!
  - Intermediate result stored after each step to a scratch buffer
  - Parallelized with `std::for_each(std::execution::par_unseq`
  - Local accumulation of convolution in doubles
  - Clamp control
  - Leverages Static Polymorphism for speed, e.g.:
    - Input and Output format can be provided as template parameters, encode/decode get inlined into hot loops
    - Swizzles and Dithers can be compile time too
- accounts for when alpha being used as coverage

Careful handling of the Alpha Channel in Mip Mapping

Consider N pixels rendered with super sampling, then averaging them to a single one. The result we expect depends on the semantics:

- Alpha Testing $(SRC - DST) \cdot (\overline{\alpha > \sigma \; ? \; 1 \; : \; 0}) \; + \; DST$
  - Distribution/Histogram changes so does the number of pixels passing the alpha test with subsequent mip levels.
  - So scale the mip level's alpha up/down to keep coverage percentage constant
- Blending Against Background $\frac{1}{N}(Background(1 - \sum \alpha_i) + \sum color_i \alpha_i)$ equals
  - Premultiplied $\overline{DST} \cdot (1 - \overline{\alpha}) + (\overline{SRC \cdot \alpha})$
  - Not-Premultiplied $\overline{DST} \cdot (1 - \overline{\alpha}) + \frac{(\overline{SRC \cdot \alpha})}{\overline{\alpha}} \cdot \overline{\alpha}$
    - Horrible singularities and precision issues due to division by Average Alpha
    - Just to multiply the prefiltered RGB value by Average Alpha when rendering



1 level mipmap

100% 100% 100% 100% 66% 50% 33% 25%
100% 100% 100% 66% 50% 33% 25% 0%
100% 100% 100% 50% 33% 25% 0% 0%
100% 66% 50% 100% 25% 0% 0% 0%
66% 50% 33% 25% 100% 50% 0% 0%
50% 33% 25% 0% 50% 100% 50% 0%
33% 25% 0% 0% 0% 50% 100% 50%
25% 0% 0% 0% 0% 0% 50% 100%

2 level mipmap

100% 91.5% 49.75% 20.75%
91.5% 75% 20.75% 0%
49.75% 20.75% 75% 12.5%
20.75% 0% 12.5% 75%

3 level mipmap

89.5% 22.8125%
22.8125% 43.75%

4 level mipmap

44.71875%

Let's check how many pixels for each mipmap level passed the alpha test (they were not discarded).

Mipmap level	Pixel passing alpha test
1	50%
2	37.5%
3	25%
4	0%

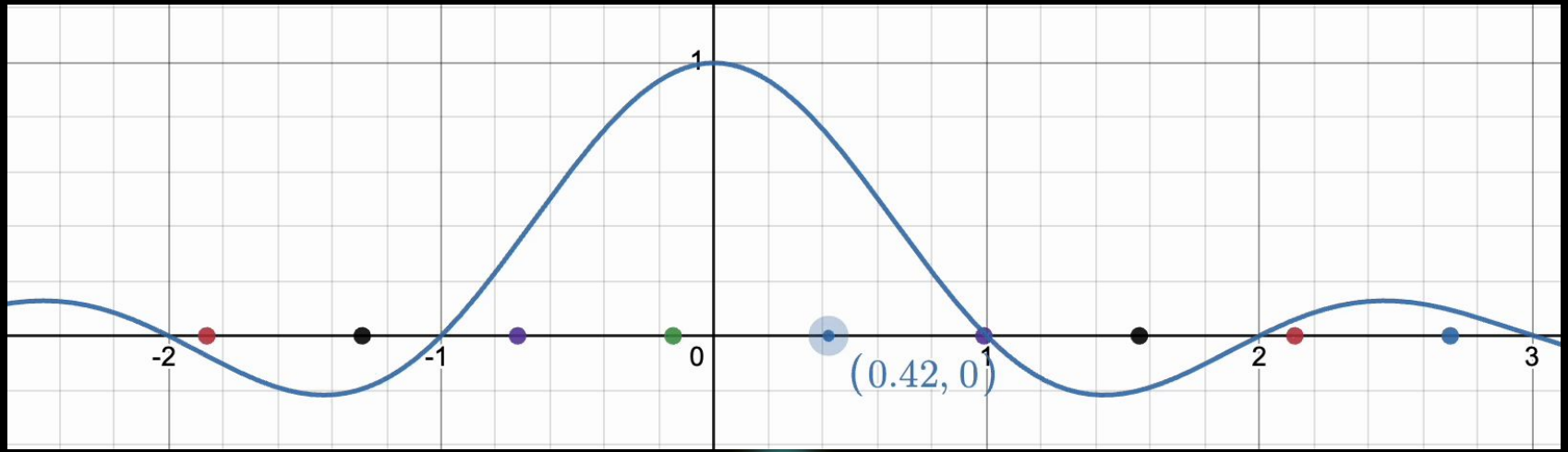
Source : https://asawicki.info/articles/alpha_test.php5

The Polyphase LUT Optimization

Due to Rational Scaling, shifting a pixel in the output makes the input pixels align slightly differently relative to the output which leads to distinct "phases".

Every P samples the phase repeats itself.

This allows a Look-Up-Table (LUT) **[P]x[ceil(support)]** per Convolution Kernel per Axis to be precomputed on CPU, then get used by the filter implementation.



The Polyphase LUT Optimization

- P can be as large as $\max(InputResolution,\ OutputResolution)$, e.g. when one is 1023 and other 1024. But because we usually convolve 2D and 3D images:
  - the LUT consumes very little extra memory and can fit into L2 CPU or L1 GPU cache
  - some kernels like Sinc are expensive to evaluate, so fetching value from LUT is faster
    - *Especially true for Kaiser windowed Sinc - Bessel function is expensive*
  - we can afford the Kernel to be a convolution of arbitrary Reconstruction and Resampling functions, because there are comparatively few values in the LUT and we can compute them through Numerical Integration
    - The previously covered special cases of Gauss on Gauss, etc. are handled via C++ Template Specialization
- MAX_SUPPORT_SIZE is computed by taking the support of the Convolution Kernel whose width has been scaled to match the output sampling interval, then converting it to input image integer coordinates (not normalized) and ceiling

Considerations for a GPU Implementation

The considerations for a GPU implementation are different:

- Do not want a separate dispatch per axis
  - batches of Compute Blit dispatches can be very large (i.e. recompute all mipmaps in a scene at once)
  - allocating and synchronizing per-axis-pass intermediate output global scratch memory, limits parallelism
  - would need pipeline barrier between every pass on the same image (Vulkan Event/Split Barrier spam)
  - GPUs are much more parallel than CPUs, might actually have more resident invocations than image scanlines
  - do all 3 axes at once in a Single Dispatch instead, store directly to output, but keep the Convolution separable
  - normalizations (like Alpha Coverage Adjustment) require scratch memory and 1 or 2 extra dispatches, however:
    - Histograms are fixed size per image (depend on value bucket count)
      - No separate prefix sum dispatch needed, small enough to be done redundantly in a work group
    - Transient high-precision (fp32, etc.) pre-quant and dither storage size is proportional to output resolution
- Avoid FP64 like the plague, leverage "free" format conversions, so access the LUT as Uniform Texel Buffer

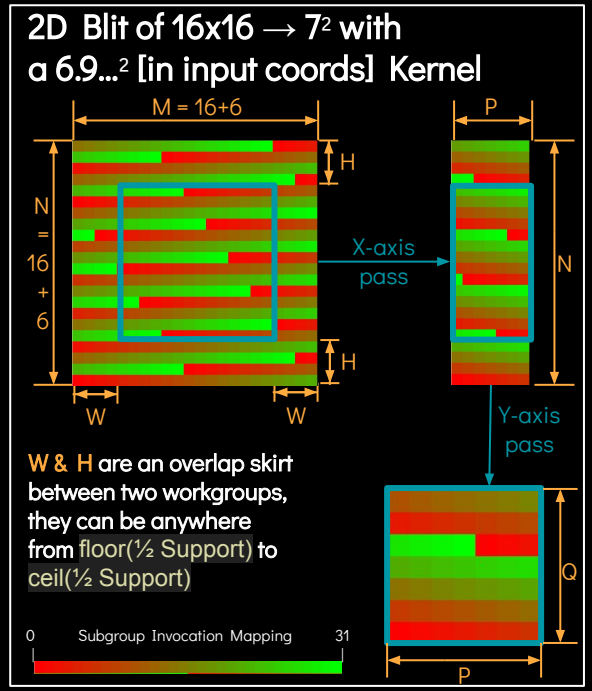
Details of a GPU Implementation (1/2)

Persistent invocations mapped in snake-order to Shared Memory indices

1. Preload an *MxNxO* input texel patch into Work Group's Shared Memory (SMem) which will produce an *PxQxR* output patch
2. Issue a Work Group execution barrier
3. Remap invocations to output texels of an axis-pass and compute the convolution as a gather operation
   a. If there will be another pass, store results back in SMem
   b. Otherwise store straight to output image

There are "wasted" loads on the border of the cube where WGs overlap, as PxQxR need to have have the kernel supports subtracted.

The Kernels' supports needs to be small and have an actual range limit



Details of a GPU Implementation (2/2)

There are some fun potential optimizations we're about to try:

- Register file is underutilized, can halve SMem usage by using local private variables to perform the swap of SMem contents for the price of one extra WG execution barrier
- Vulkan StorageImageWriteWithoutFormat feature lets you have a single pipeline for all the blitting of images of identical SPIR-V dimension
    - *Maybe alias different dimension images to the same descriptor binding?*
- Bonus: Pick the first *(2D Blit)* and second *(3D Blit)* pass axes to minimize intermediate output storage
    - *Only matters for upscale, otherwise the initial input patch is larger than any intermediate result*
- Output patch can be larger if each color channel is processed separately or float16_t is used for intermediate storage (*textureGather looks interesting here)*

Static Polymorphism in HLSL

We want to provide the Blit as a single header library and not as a single shader, and HLSL 2021 (mostly templates) enables:

- Decoupling Input/Output/Parameter Storage from Descriptor and Pipeline Layouts, Descriptor Binding

```
template<typename PassedPixelsAccessor, typename InSamplerAccessor>
void alpha_test(
        NBL_REF_ARG(PassedPixelsAccessor) passedPixels, NBL_CONST_REF_ARG(InSamplerAccessor) inSampler,
        const uint32_t3 globalInvocationID, const uint32_t3 workGroupID, const float32_t referenceAlpha
) {
                const float32_t alpha = inSampler.get(globalInvocationID, workGroupID.z).a;
                if (alpha > referenceAlpha)
                        passedPixelsAccessor.atomicAdd(workGroupID.z, uint32_t(1));
}
```

- Aggressive Inlining and Constant Expression Elimination without Global Identifiers (more than 1 instance possible)

```
template<uint32_t Dimension> struct dim_to_image_properties
{
        using combined_sampler_t = Texture1DArray<...>;
        using image_t = RWTexture1DArray<...>;

        template<typename T>  static vector<T,2> getIndexCoord(const vector<T,3> coords, const uint32_t layer);
};

template<typename ConstevalParameters> struct compute_blit_t;

#pragma unroll // later on in `compute_blit_t::execute<...>(...)`
for (uint32_t virtualInvocation=localInvocationIx; virtualInvocation<virtualInvocations; virtualInvocation+=ConstevalParameters::WorkGroupSize)
```

Future: Shared Source Software Decode/Encode

The C++ functions for transcoding Texel Blocks have been written many years ago:

```
template<asset::E_FORMAT fmt, typename T> requires (std::is_arithmetic_v<T> && sizeof(T)==8)
inline void decodePixels(const void* _pix[MAX_PLANES], T* _output, uint32_t _blockX, uint32_t _blockY);

template<asset::E_FORMAT fmt, typename T> requires (std::is_arithmetic_v<T> && sizeof(T)==8)
inline void encodePixels(void* _pix, const T* _input);

// runtime format argument versions exist for all the E_FORMAT templated functions
template<typename T> requires (std::is_arithmetic_v<T> && sizeof(T)==8)
inline bool encodePixels(asset::E_FORMAT _fmt, void* _pix, const T* _input);
```

And they are missing some features:

- Block Compressed format encoding, due to:
  - Unexpired patents → only S3TC is expired so far
  - Local Optimums in encoding error are not best → might want to prioritize continuity between individual blocks
  - Large search space even when encoding for a local optimum
- Planar Formats → we never used them

Want a single header HLSL implementation which compiles as C++, but:

- Need to avoid float64_t, uint64_t, and int64_t as transient formats unless truly necessary
- Low priority goal, will probably start with porting Format enums and Format traits (ranges, precision, etc)

Bonus Round: HQ Normal and Derivative Maps

Remember the Derivative Property? We can smack a derivative operation in the middle of our resampling.

$g = \nabla f \circ (r_{construct} \circ r_{sample}) \cdot Shah_{\frac{1}{OutputResolution}}$

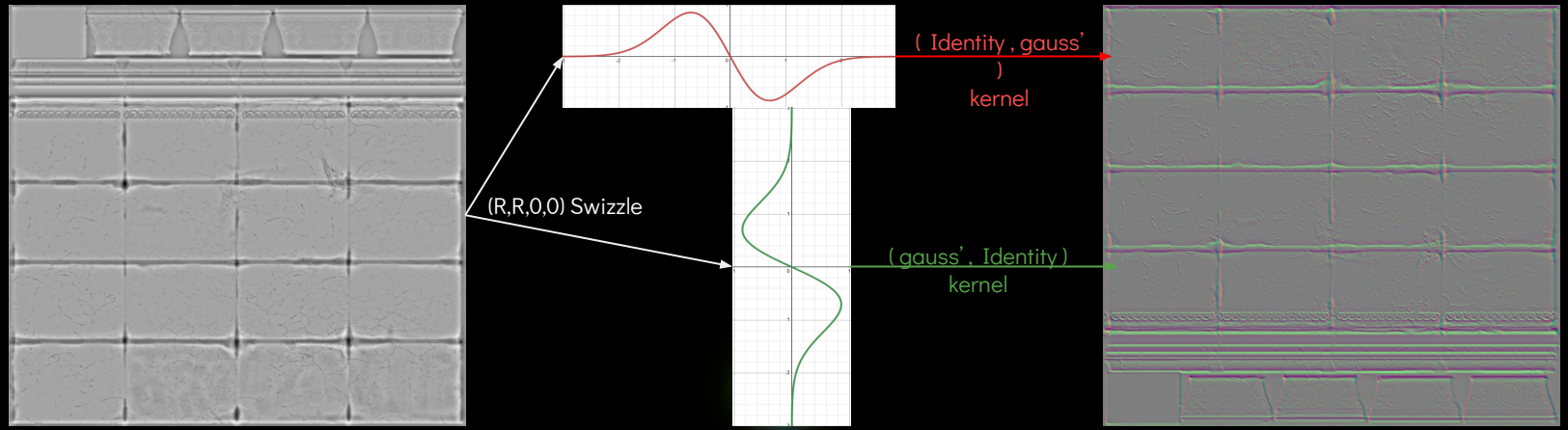
Actually Convolution is Commutative as well, so the above is equivalent to a Single Convolution Blit of f with k where

$k = \nabla(r_{construct} \circ r_{sample})$ or $k = (r_{construct} \circ \nabla r_{sample})$

we have implemented the Derivative Property to our templated Gauss on Gauss, Sinc on Sinc, and other combos.

Note that Box and Dirac are not C1 and the Blit filter used for differentiation must use at least one C1 kernel.

No longer feel uneasy about a 4096^2 Bump Map **NOT** producing a 4095^2 Output Map, or about using central differences.



Questions ?

Credits:

- Initial Image Filter API → Mateusz Kielan
- Flatten, Copy, Dither, SAT Filters → Arkadiusz Lachowicz
- Virtual Texturing and Padded Copy Filter → Krzysztof Szenk
- Improvements to Format Convert Filters → Erfan Ahmadi
- FFT Bloom/Flare → Erfan Ahmadi
- Unoptimized CPU Blit Filter → Mateusz Kielan
- Polyphase LUT and Double Convolution Blit → Achal Pandey
- GLSL Compute Blit → Achal Pandey
- WIP : HLSL SPIR-V 1.6 and Vulkan 1.3 Port → Mihail Mladenov