Vulkanised 2025

The 7th Vulkan Developer Conference
Cambridge, UK | February 11-13, 2025

Practical Neural Texture Compression

Alexey Panteleev, NVIDIA

RTX Kit and Neural Rendering Initiative

Announced by NVIDIA at CES 2025

Neural Radiance Cache

DLSS 4

Neural Faces

Neural Materials

Neural Texture Compression

Tuscan Villa scene with BCn textures (6.5 GB VRAM)

Tuscan Villa scene with NTC textures (970 MB VRAM)

lla Brisby
hianti Classico
DOCG

Downscaled BCn textures (970 MB)

lla Brisby
Chianti Classico
D.O.C.G.

Full resolution NTC textures (970 MB)

BCn (970 MB)
lla Brisby
Chianti Classico
DOCG
NTC (970 MB)

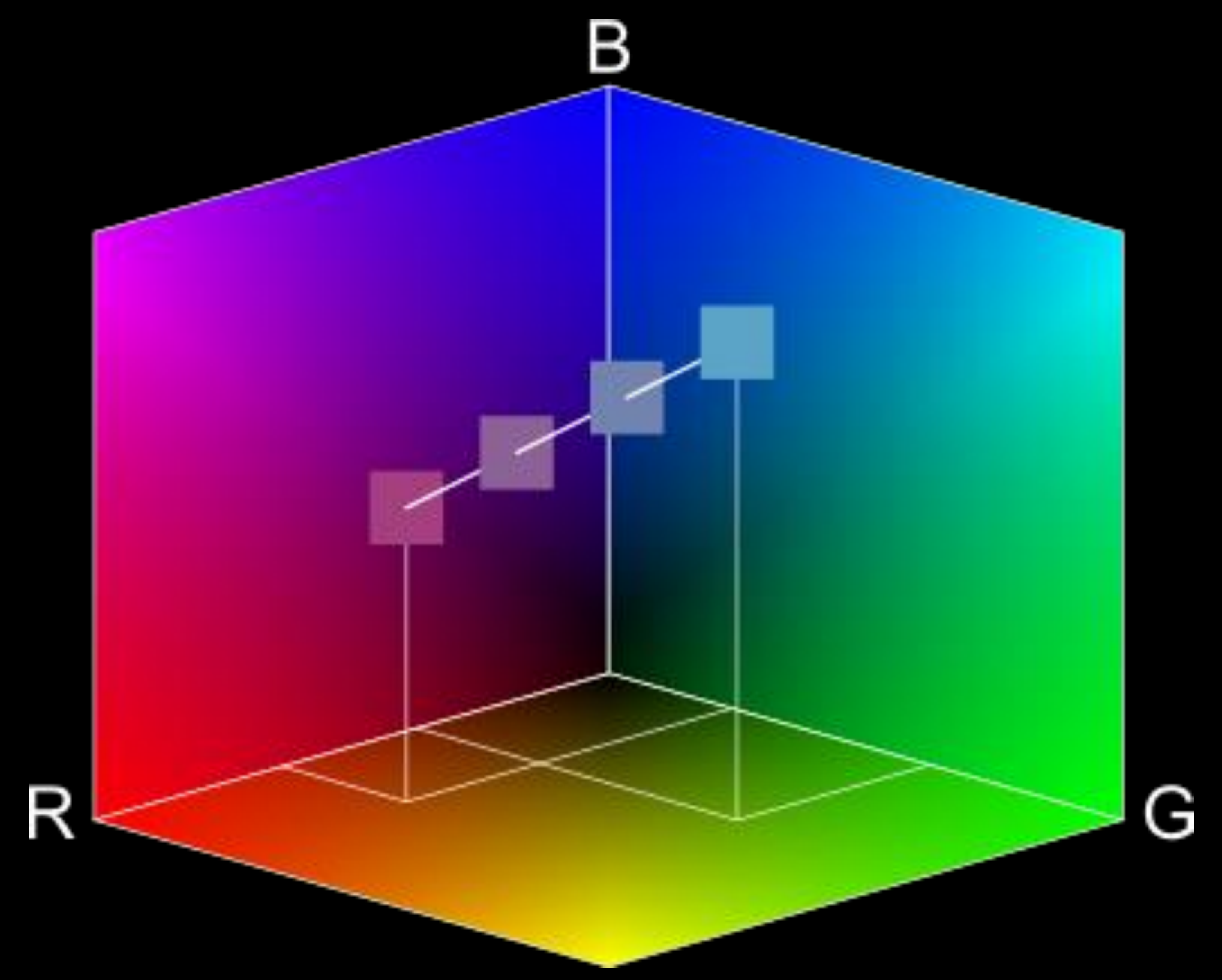
Neural vs Block Compression

How is it different?

Block Texture Compression (BC1-BC7)

- Operates on 1- to 4-channel images
- Fixed number of bits for each 4x4 block (1-2 bppc*)
- Most formats assume channels are correlated

- …But modern materials have more than 4 channels



https://www.reedbeta.com/blog/understanding-bcn-texture-compression-formats/

RTX Neural Texture Compression

- Operates on up to 16 channels at once
  - Base color, Normal, Metalness, Roughness, Emission…
- Adjustable compression rate (1/32 … 20 bppc*)
- Assumes channels are correlated

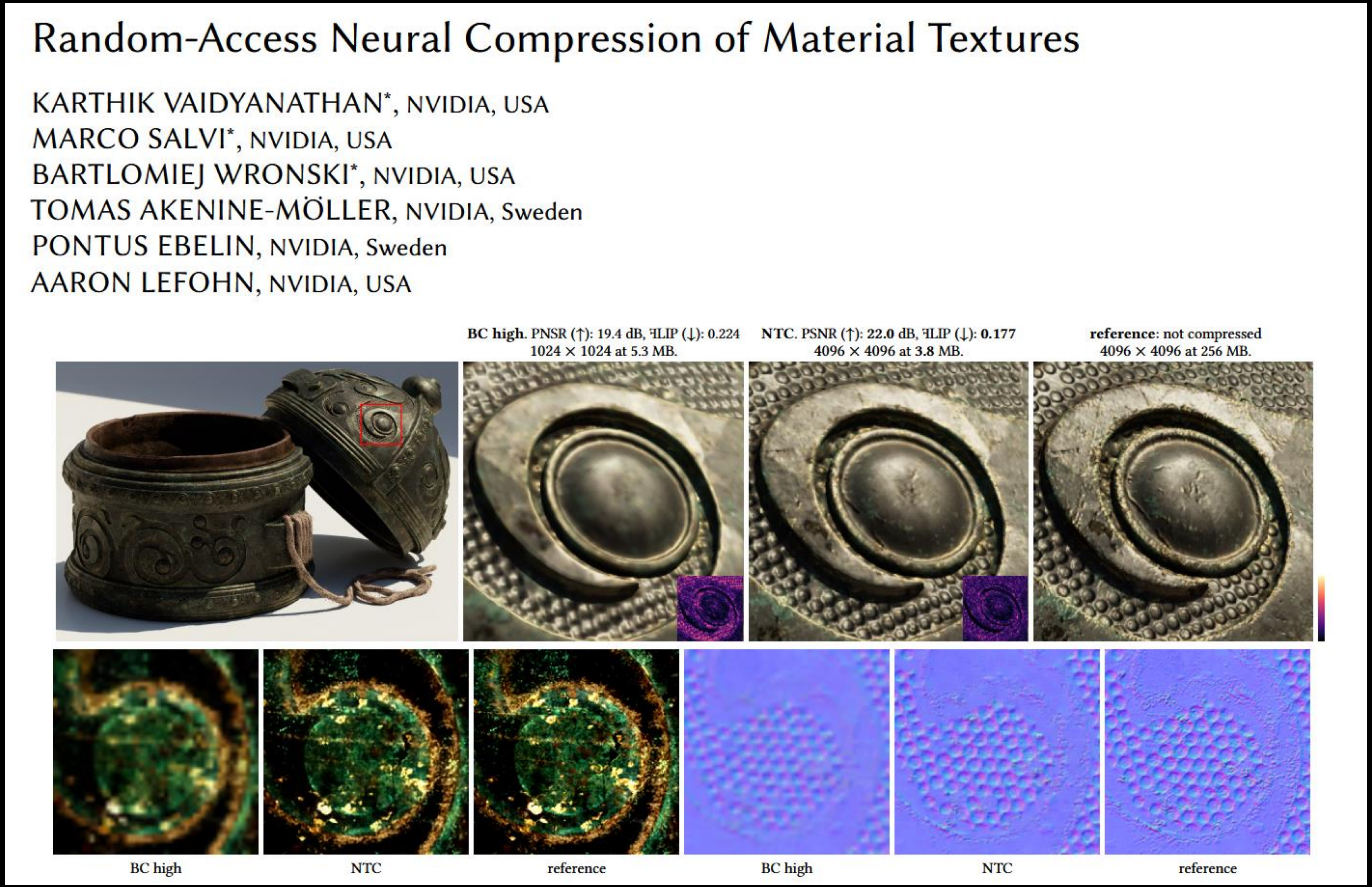
- Trains a very small neural network per material
  - Compression = training
  - No large training dataset or foundation models
  - No hallucinations
  - Network is overfitted to the material

- Compressed data:
  - Latent representation: 99% of space
  - Network weights: 12 KB or so

** bppc = bits per pixel per channel*

Neural Texture Compression

Research Recap



Random-Access Neural Compression of Material Textures

KARTHIK VAIDYANATHAN*, NVIDIA, USA
MARCO SALVI*, NVIDIA, USA
BARTLOMIEJ WRONSKI*, NVIDIA, USA
TOMAS AKENINE-MÖLLER, NVIDIA, Sweden
PONTUS EBELIN, NVIDIA, Sweden
AARON LEFOHN, NVIDIA, USA

BC high. PSNR (↑): 19.4 dB, ꟻLIP (↓): 0.224
1024 × 1024 at 5.3 MB.

NTC. PSNR (↑): **22.0** dB, ꟻLIP (↓): **0.177**
4096 × 4096 at **3.8** MB.

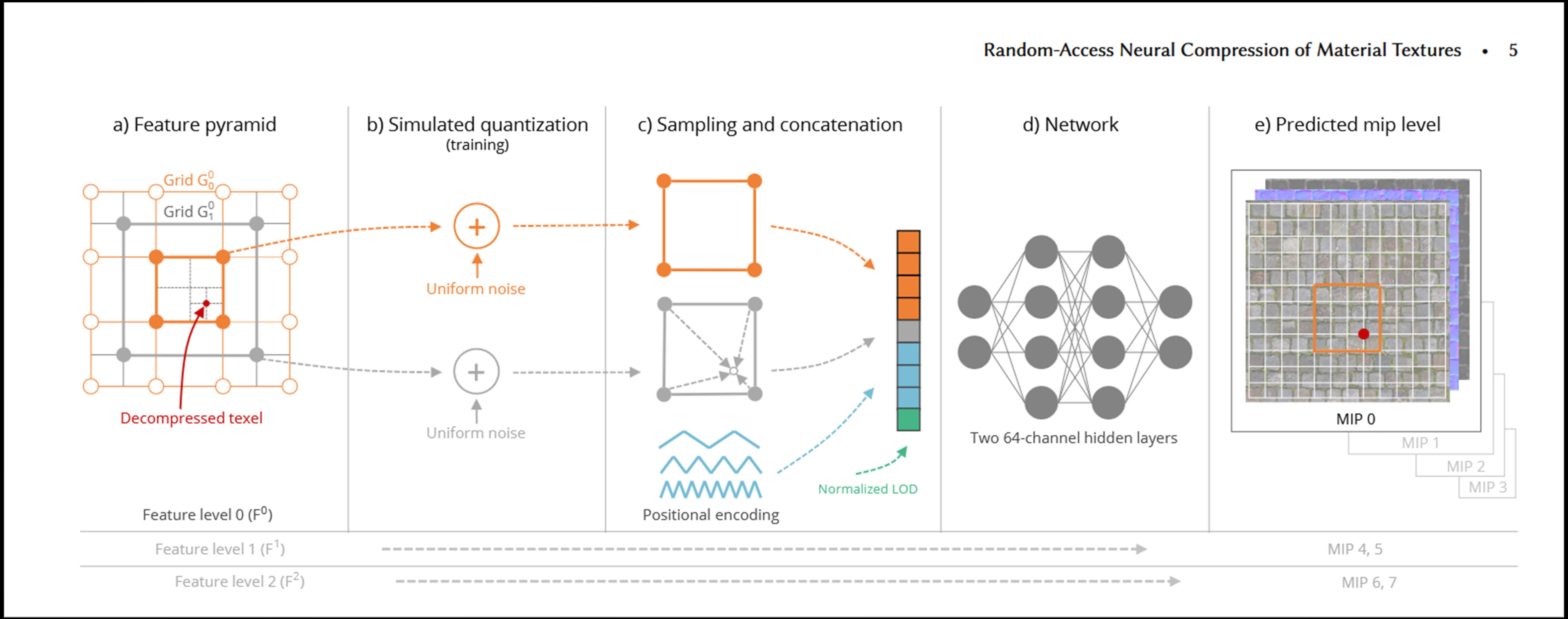
reference: not compressed
4096 × 4096 at 256 MB.

BC high | NTC | reference | BC high | NTC | reference

Neural Texture Compression

Algorithm Basics

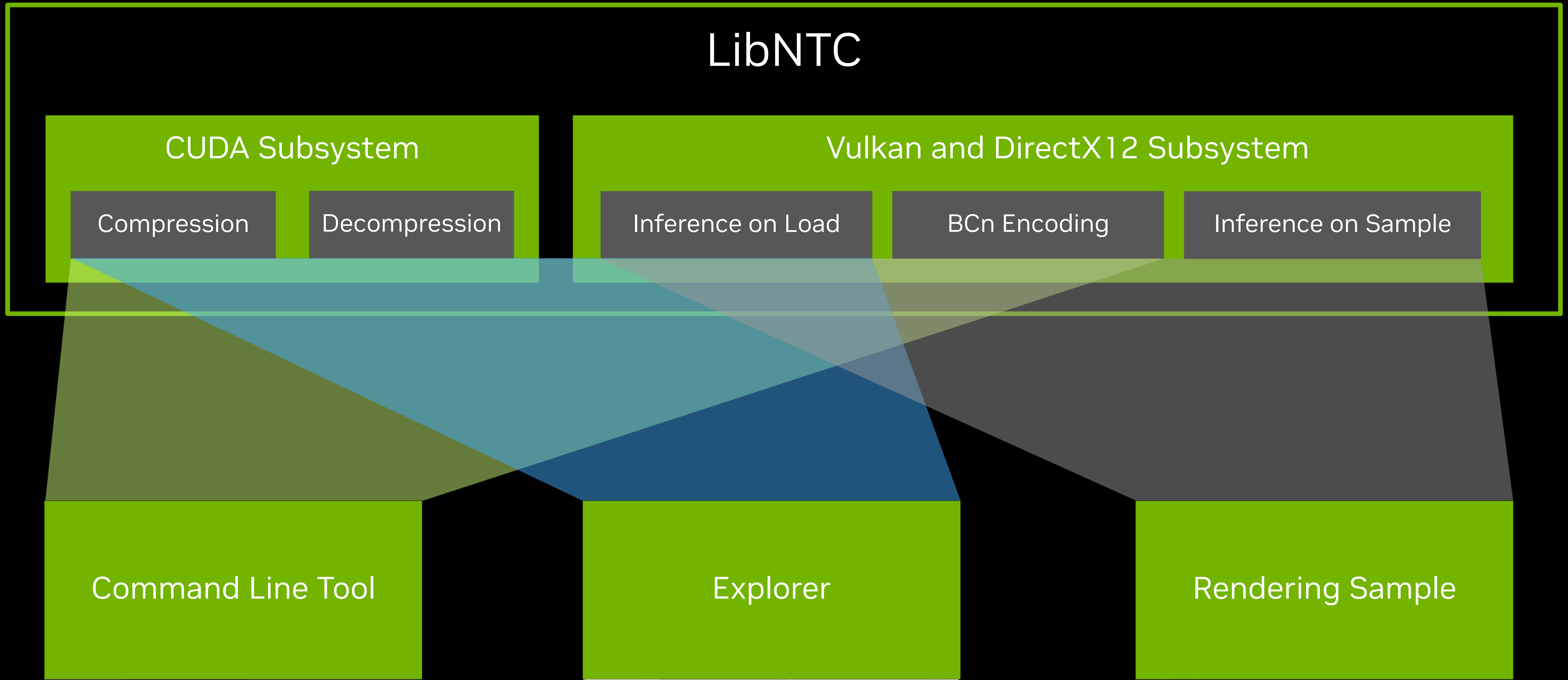
Compression: Train the network and features based on the reference images

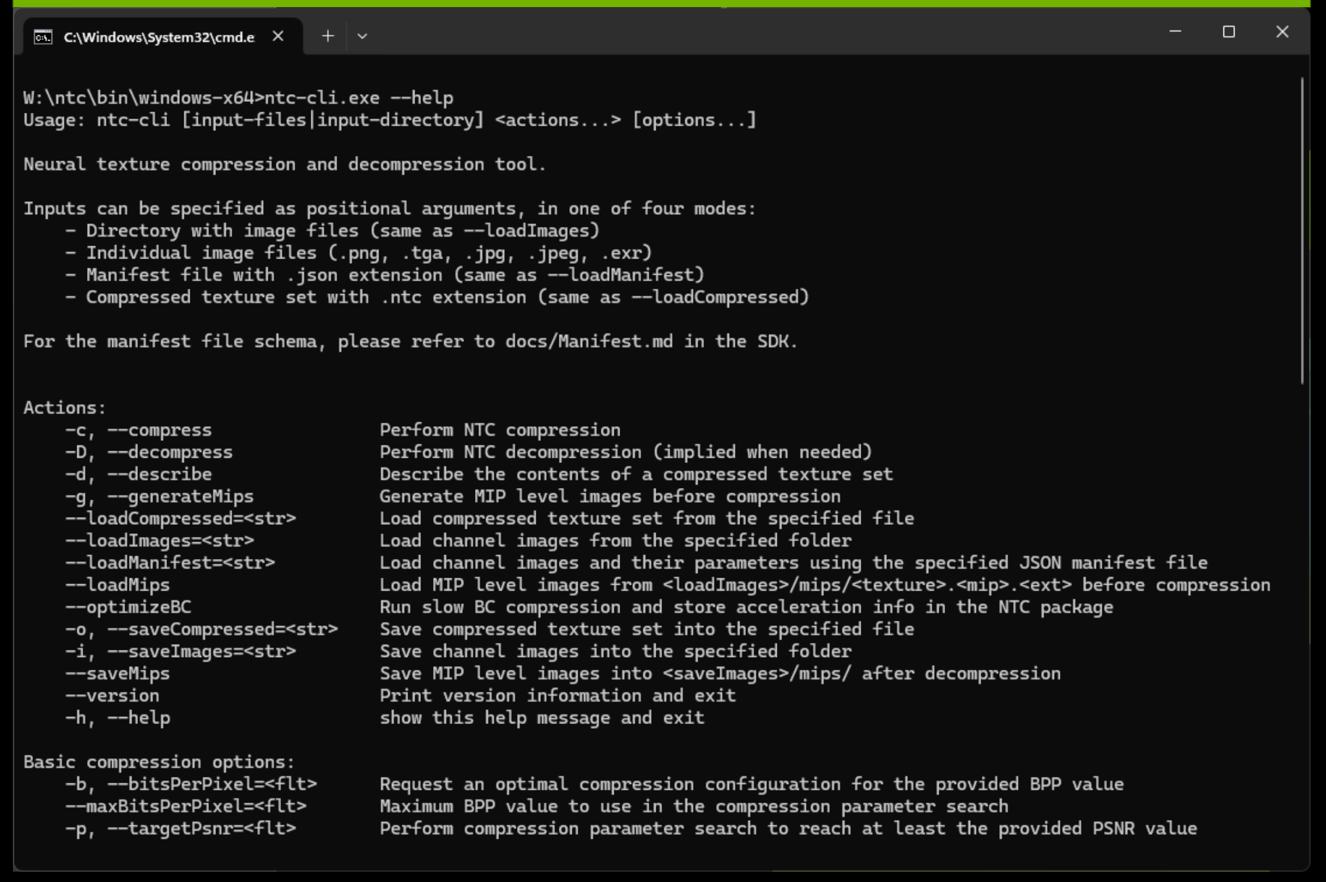


Decompression: Sample the features and pass them through the network

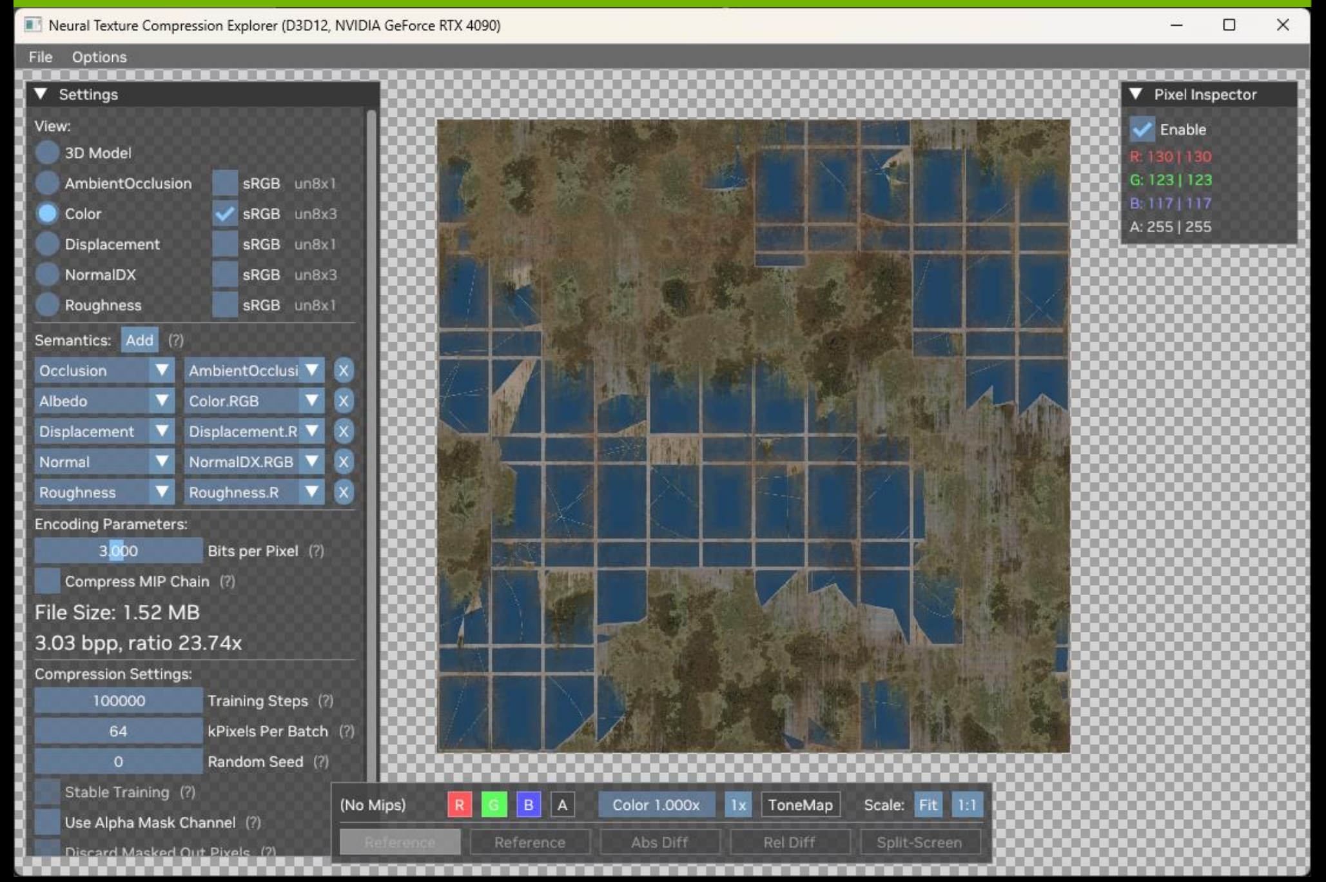
RTX Neural Texture Compression SDK

Key Component Overview





```
W:\ntc\bin\windows-x64>ntc-cli.exe --help
Usage: ntc-cli [input-files|input-directory] <actions...> [options...]

Neural texture compression and decompression tool.

Inputs can be specified as positional arguments, in one of four modes:
    - Directory with image files (same as --loadImages)
    - Individual image files (.png, .tga, .jpg, .jpeg, .exr)
    - Manifest file with .json extension (same as --loadManifest)
    - Compressed texture set with .ntc extension (same as --loadCompressed)

For the manifest file schema, please refer to docs/Manifest.md in the SDK.


Actions:
    -c, --compress                Perform NTC compression
    -D, --decompress              Perform NTC decompression (implied when needed)
    -d, --describe                Describe the contents of a compressed texture set
    -g, --generateMips            Generate MIP level images before compression
    --loadCompressed=<str>        Load compressed texture set from the specified file
    --loadImages=<str>            Load channel images from the specified folder
    --loadManifest=<str>          Load channel images and their parameters using the specified JSON manifest file
    --loadMips                    Load MIP level images from <loadImages>/mips/<texture>.<mip>.<ext> before compression
    --optimizeBC                  Run slow BC compression and store acceleration info in the NTC package
    -o, --saveCompressed=<str>    Save compressed texture set into the specified file
    -i, --saveImages=<str>        Save channel images into the specified folder
    --saveMips                    Save MIP level images into <saveImages>/mips/ after decompression
    --version                     Print version information and exit
    -h, --help                    show this help message and exit

Basic compression options:
    -b, --bitsPerPixel=<flt>      Request an optimal compression configuration for the provided BPP value
    --maxBitsPerPixel=<flt>       Maximum BPP value to use in the compression parameter search
    -p, --targetPsnr=<flt>        Perform compression parameter search to reach at least the provided PSNR value
```



View:
- 3D Model
- AmbientOcclusion — sRGB un8x1
- [✓ Color] — [✓ sRGB] un8x3
- Displacement — sRGB un8x1
- NormalDX — sRGB un8x3
- Roughness — sRGB un8x1

Semantics: Add (?)

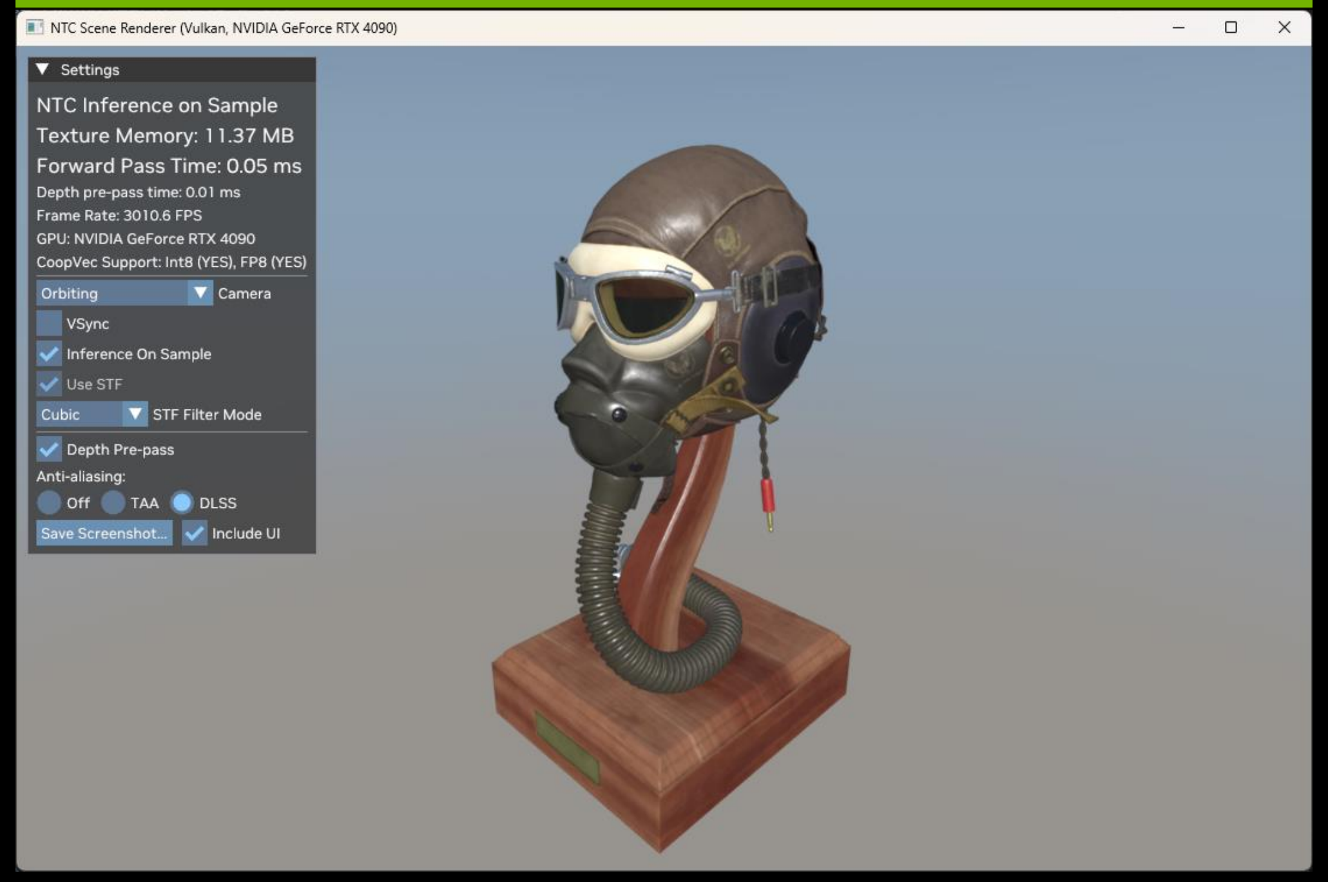
Occlusion	AmbientOcclusi
Albedo	Color.RGB
Displacement	Displacement.R
Normal	NormalDX.RGB
Roughness	Roughness.R

Encoding Parameters:
3.000 Bits per Pixel (?)
Compress MIP Chain (?)
File Size: 1.52 MB
3.03 bpp, ratio 23.74x

Compression Settings:
100000 Training Steps (?)
64 kPixels Per Batch (?)
0 Random Seed (?)
Stable Training (?)
Use Alpha Mask Channel (?)
Discard Masked Out Pixels (?)

(No Mips) R G B A Color 1.000x 1x ToneMap Scale: Fit 1:1

Pixel Inspector: [✓ Enable]
R: 130 | 130
G: 123 | 123
B: 117 | 117
A: 255 | 255

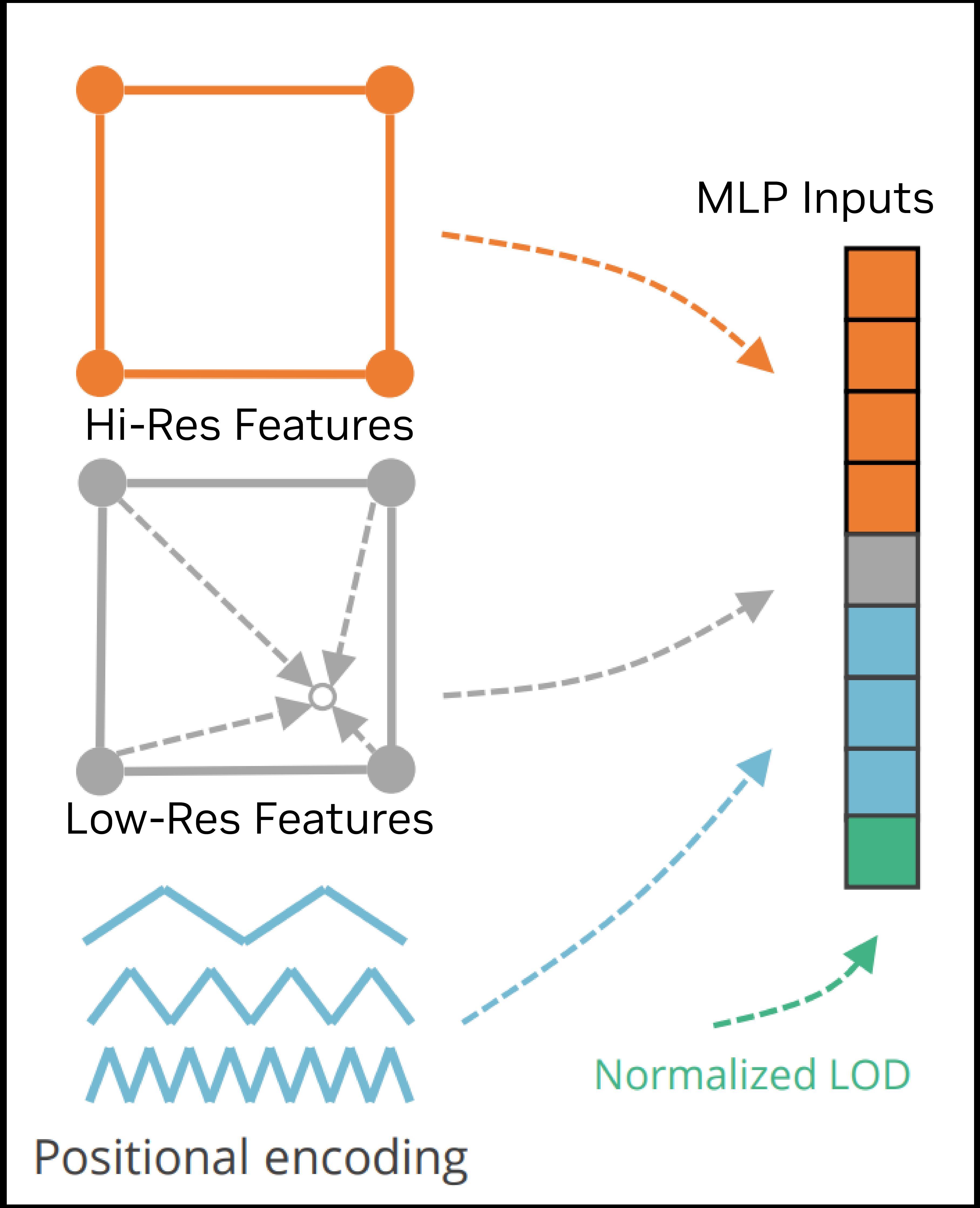


NTC Inference on Sample
Texture Memory: 11.37 MB
Forward Pass Time: 0.05 ms
Depth pre-pass time: 0.01 ms
Frame Rate: 3010.6 FPS
GPU: NVIDIA GeForce RTX 4090
CoopVec Support: Int8 (YES), FP8 (YES)
Orbiting — Camera
VSync
[✓ Inference On Sample]
[✓ Use STF]
Cubic — STF Filter Mode
[✓ Depth Pre-pass]
Anti-aliasing: Off TAA [✓ DLSS]
[✓ Include UI]

Compression (CUDA Subsystem)

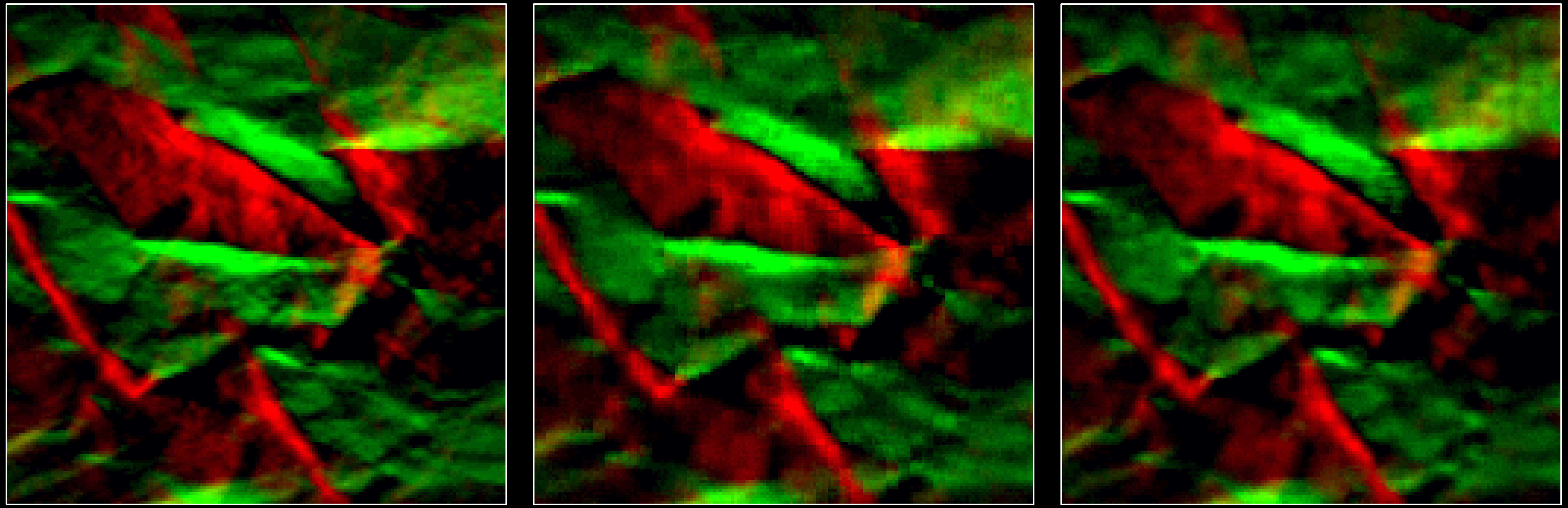
Algorithmic Improvements #1

Weighted Sampled Features



- Original paper:
  - Hi-res features copied to input vector verbatim
  - Low-res features go through bilinear interpolation
- Improved version:
  - Hi-res features are multiplied by their bilinear weights
- Benefit:
  - No more blocky artifacts at 1/4 feature grid scale!

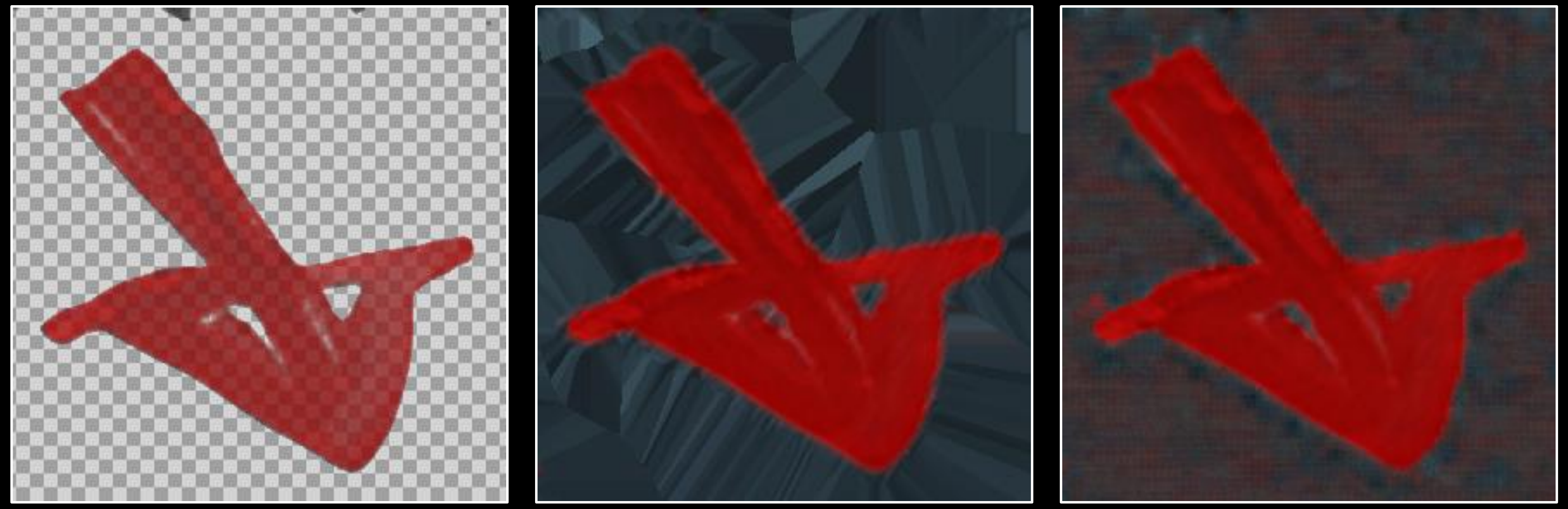
RG channels of a normal map with enhanced contrast:

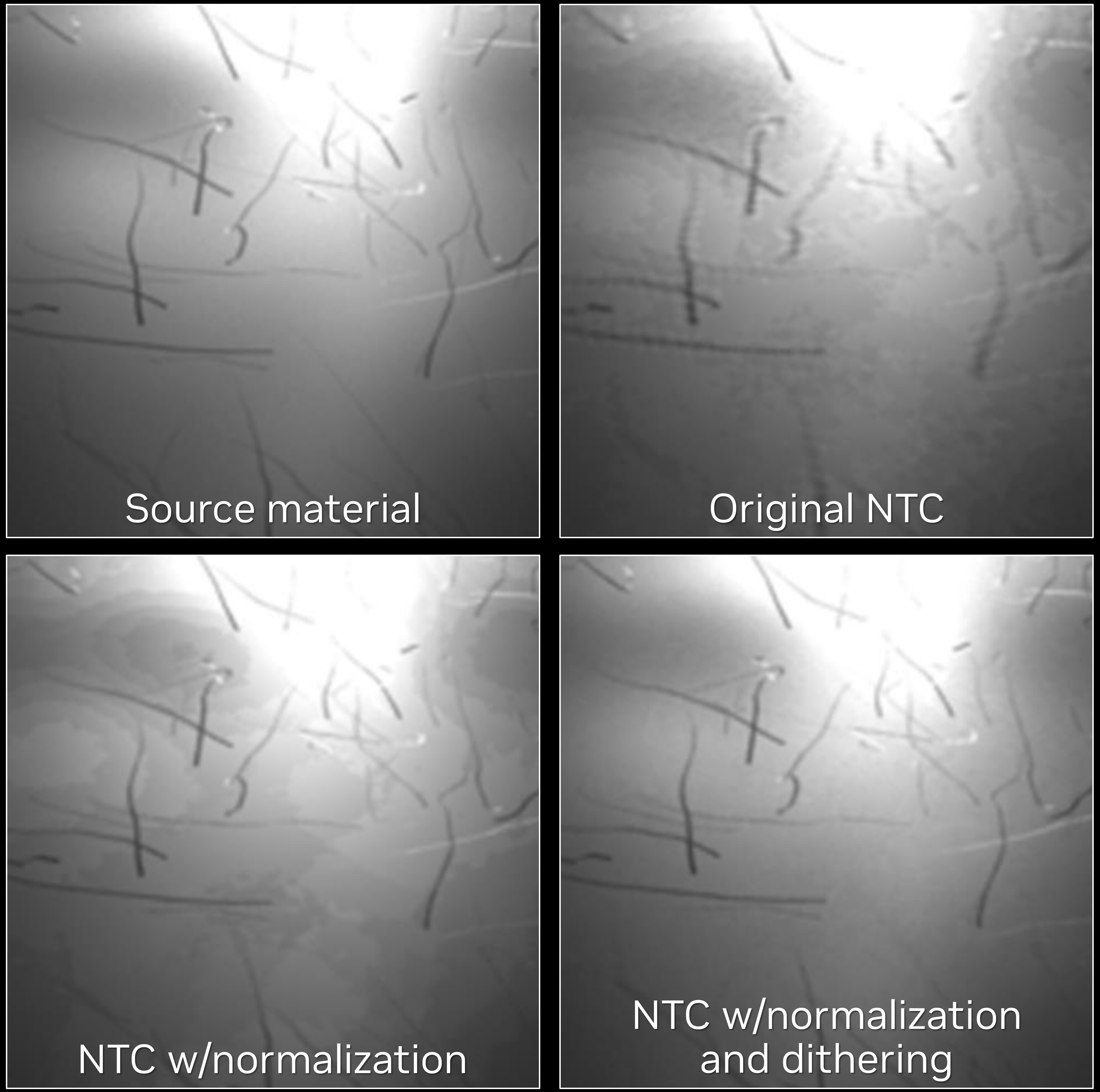


Algorithmic Improvements #2

Other things that improve image quality

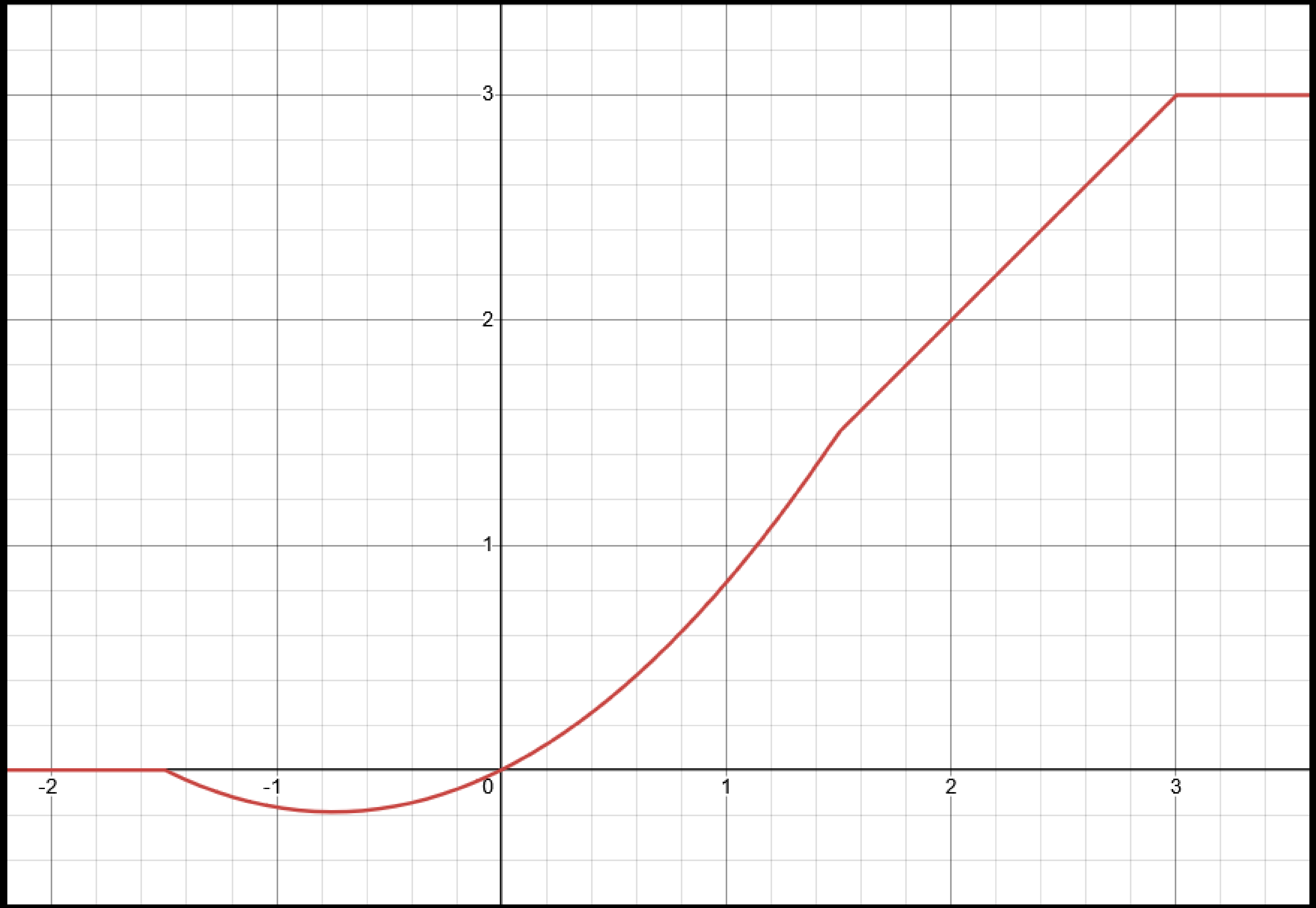
- Normalize each texture channel to [0, 1] before compression
  - Big quality improvement for channels with very small variance
  - New details can leak into non-correlated channels and reduce overall PSNR
- Dither the results on output to RGBA8 textures
  - NTC output has higher precision than 8 bits per channel
- Compress HDR images to Hybrid Log-Gamma color space
- Use alpha mask to ignore regions where mask is 0
  - Let the network focus on learning important areas


Blended Texture 42.2 dB 43.5 dB



Algorithmic Improvements #3

INT8 and FP8 Quantization Support



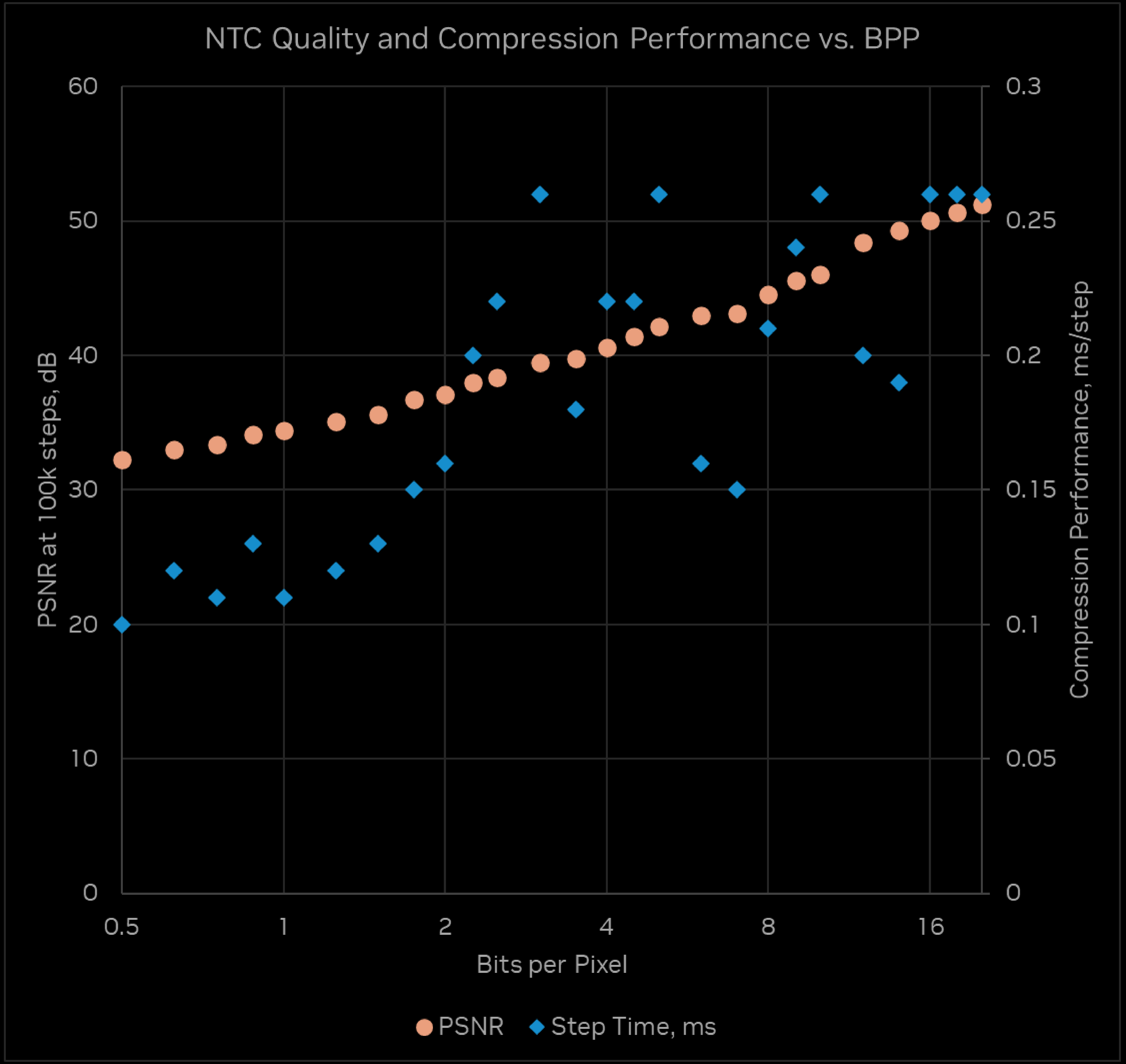
$$\max(x, 3) \cdot clamp(\frac{x}{3} + \frac{1}{2}, 0, 1)$$

- Original paper:
  - hardGELU activation function
- Improved version:
  - hardGELU activation clamped to [-3/16, 3.0]
- Benefit:
  - Can scale the layer inputs to fill the [-128, 127] range
- Quantized data flow:
  - Layers 1-3 evaluated in FP8 for performance
  - Layer 4 evaluated in INT8 for output precision
  - All INT8 version available for GPUs that don't support FP8

Compression Performance

Highly optimized CUDA kernels

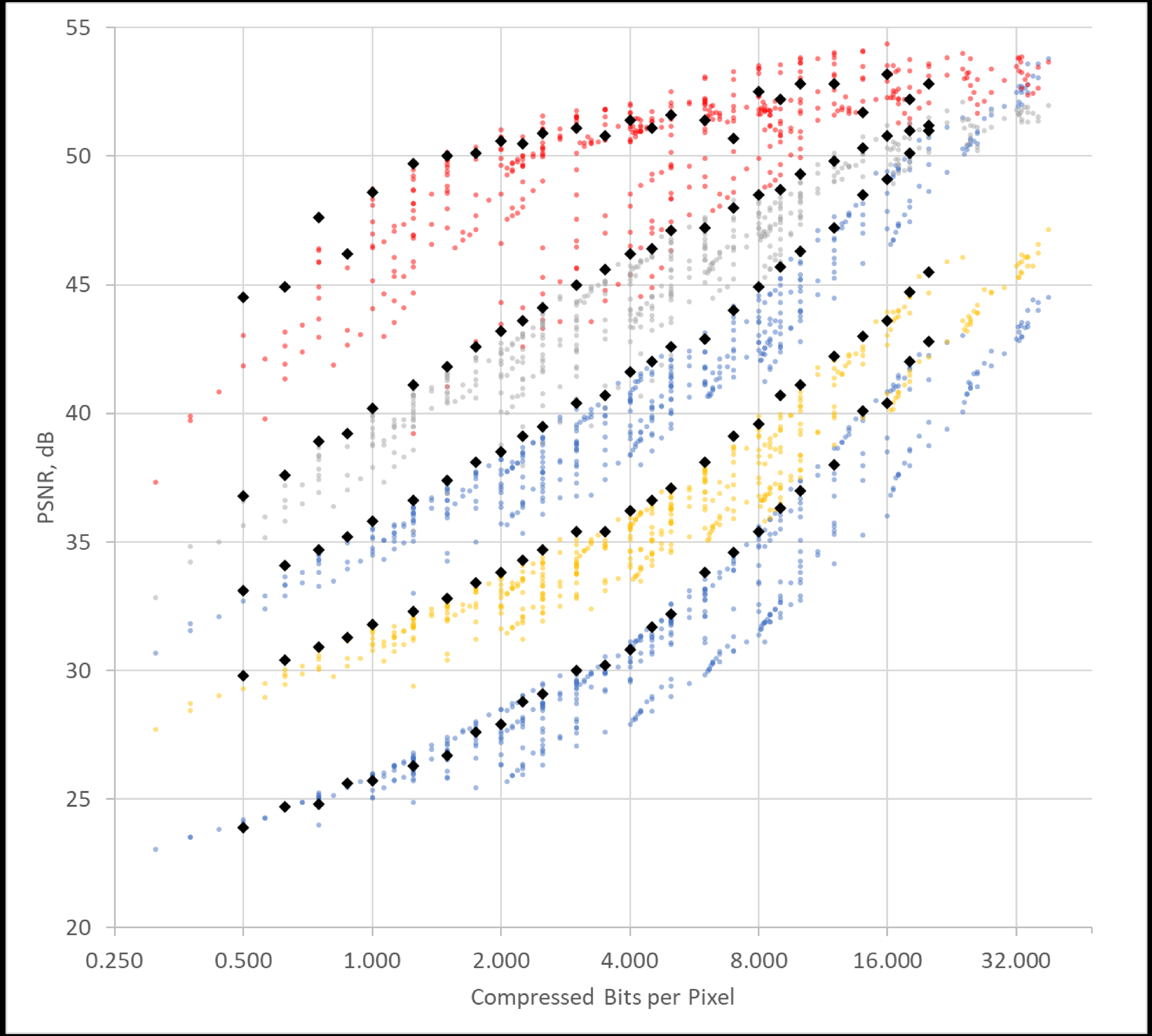
- Entire NTC evaluation pipeline is fused into one kernel
  - No extra video memory round-trips
  - One thread evaluates all channels of a single texel
  - Forward and backprop in the same pass, optimizer is separate
- Compression has been optimized since the paper version
  - 25-30x perf improvement
  - 100k steps provides good compression quality
    - Well under a minute on RTX 4090
- Many optimizations combined, no single breakthrough
  - Optimized tensor core usage
  - Improved occupancy
  - Custom RNG
  - FP16 gradients
  - Removed register indexing, …



Compression Configuration Space

Lots of available "latent shapes" or BPP settings

- Available latent shape parameters:
  - High-res latent grid size: 1/2 or 1/4 of the texture size
  - High-res and low-res feature counts: 4, 8, 12, 16
  - High-res and low-res bits per feature: 1, 2, 4, 8
  - Total 2*4*4*4*4 = 512 configurations – too many options
- Introducing a simple BPP setting
  - A single number between 0.5 and 20.0 bits per pixel
  - Optimal latent shape picked automatically
  - Based on experimental data from all 512 configurations →
- Adaptive compression ratio mode
  - User provides a target PSNR value
  - Tool makes 4-5 compression runs to find the optimal BPP
  - Can take a few minutes, depending on GPU and settings
  - Use fixed BPP mode for quick iteration



Experimental data from 5 materials compressed with every latent shape. Black diamonds show the picked optimal configs.

Inference (Graphics Subsystem)

Inference on Sample

The original NTC solution

- Upload NTC latents and weights to the GPU
- Decompress each texel when needed
  - Includes an evaluation of the MLP
- Apply Stochastic Texture Filtering (STF) instead of hardware filtering
  - Direct evaluation of trilinear/aniso would be way too slow
  - Enables higher-order filters like cubic or Gaussian for free

- Benefits:
  - Minimal VRAM footprint, up to 8x lower than BCn
- Drawbacks:
  - Adverse performance effects on more complex shaders
    - MLP evaluation needs a lot of registers
    - Cooperative Vectors and SER only help to a degree
    - Could drop 20-30% of whole frame performance
  - STF is a requirement
    - Need to filter the results with DLSS
    - Denoising becomes more complicated
    - Visuals will be different vs. HW filtering

```
#include "libntc/shaders/Inference.hlsli"
#include "STFSamplerState.hlsli"

typedef NtcNetworkParams<NETWORK_VERSION> NtcParams;

ConstantBuffer<NtcTextureSetConstants> g_NtcMaterial;
ByteAddressBuffer t_InputFile;
ByteAddressBuffer t_WeightBuffer;

// Sample the texture with STF
STF_SamplerState sampler = STF_SamplerState::Create(randomVector);
float3 samplePos = sampler.Texture2DGetSamplePos(width, height, mipLevels, uv);

// Get the integer texel position and MIP level
int mipLevel = int(samplePos.z);
int2 texelPosition = int2(floor(samplePos.xy * mipSize));

// Decompress the texel and get all the channels
float channels[NtcParams::OUTPUT_CHANNELS];

NtcSampleTextureSet<NETWORK_VERSION>(g_NtcMaterial, t_InputFile, 0,
    t_WeightBuffer, 0, texelPosition, mipLevel, true, channels);

// Use the decompressed color channels as needed
if (NtcTextureSetHasChannels(g_NtcMaterial, CHANNEL_BASE_COLOR, 3))
{
    float3 baseColor = float3(
        channels[CHANNEL_BASE_COLOR + 0],
        channels[CHANNEL_BASE_COLOR + 1],
        channels[CHANNEL_BASE_COLOR + 2]);

    // ...
}
```

Inference on Load

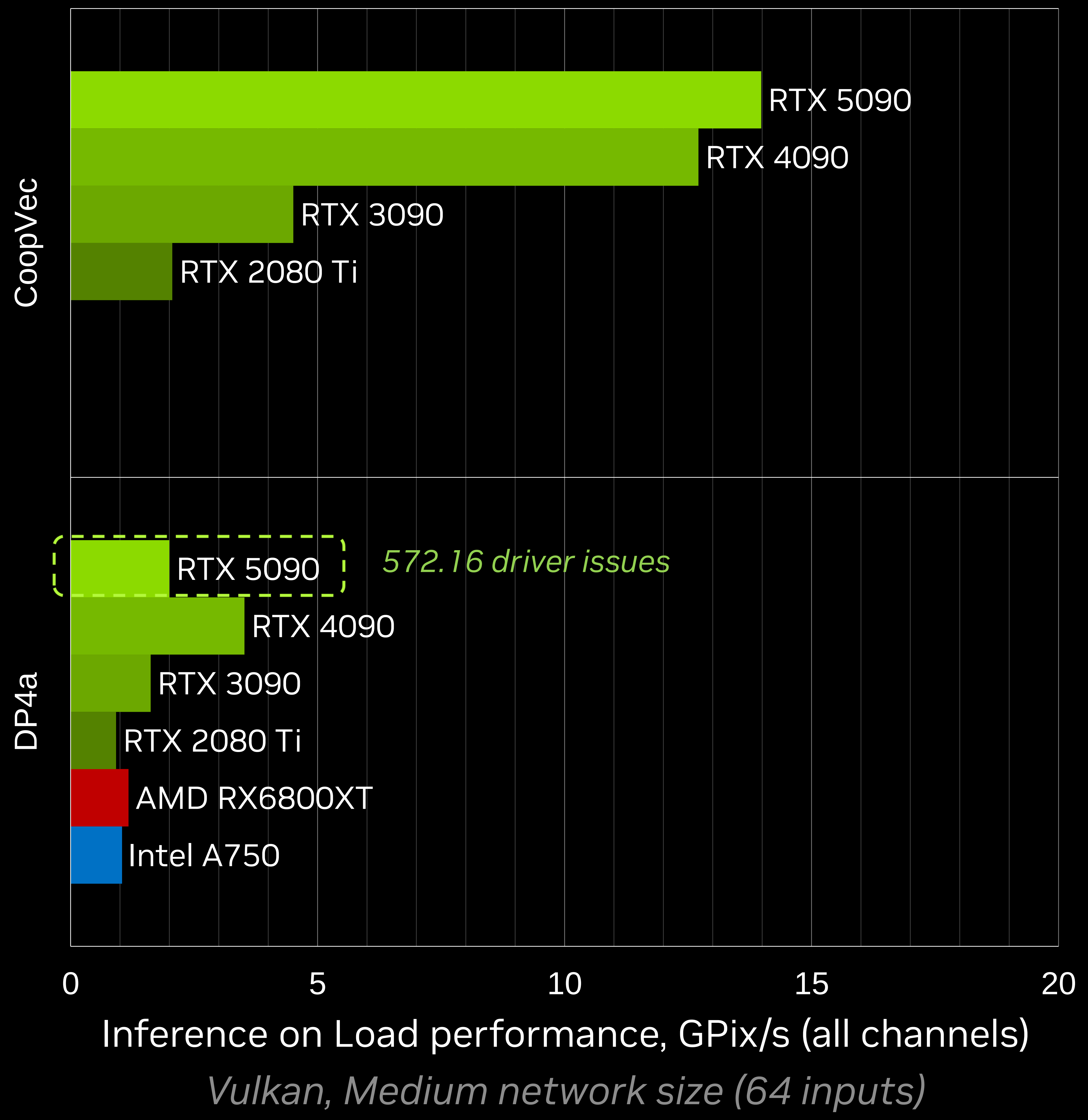
Compatibility Mode

- Decompress texture sets into color pixels at load time
- Compress into BCn right after that

- Why decompress on load?
  - Inference on Sample is impractical on all but the latest NV GPUs
  - Inference on Load works everywhere and it's not performance-critical
  - Game can be shipped with NTC textures only and work on all GPUs

- How fast is it?
  - See chart on the right
  - All textures for a 4K material decompressed in a few milliseconds

- Decompression is done entirely on the GPU
  - Minimal CPU parsing of NTC files and uploading of raw data
  - Tiled decompression is possible but a bit more involved



Transcoding to BCn

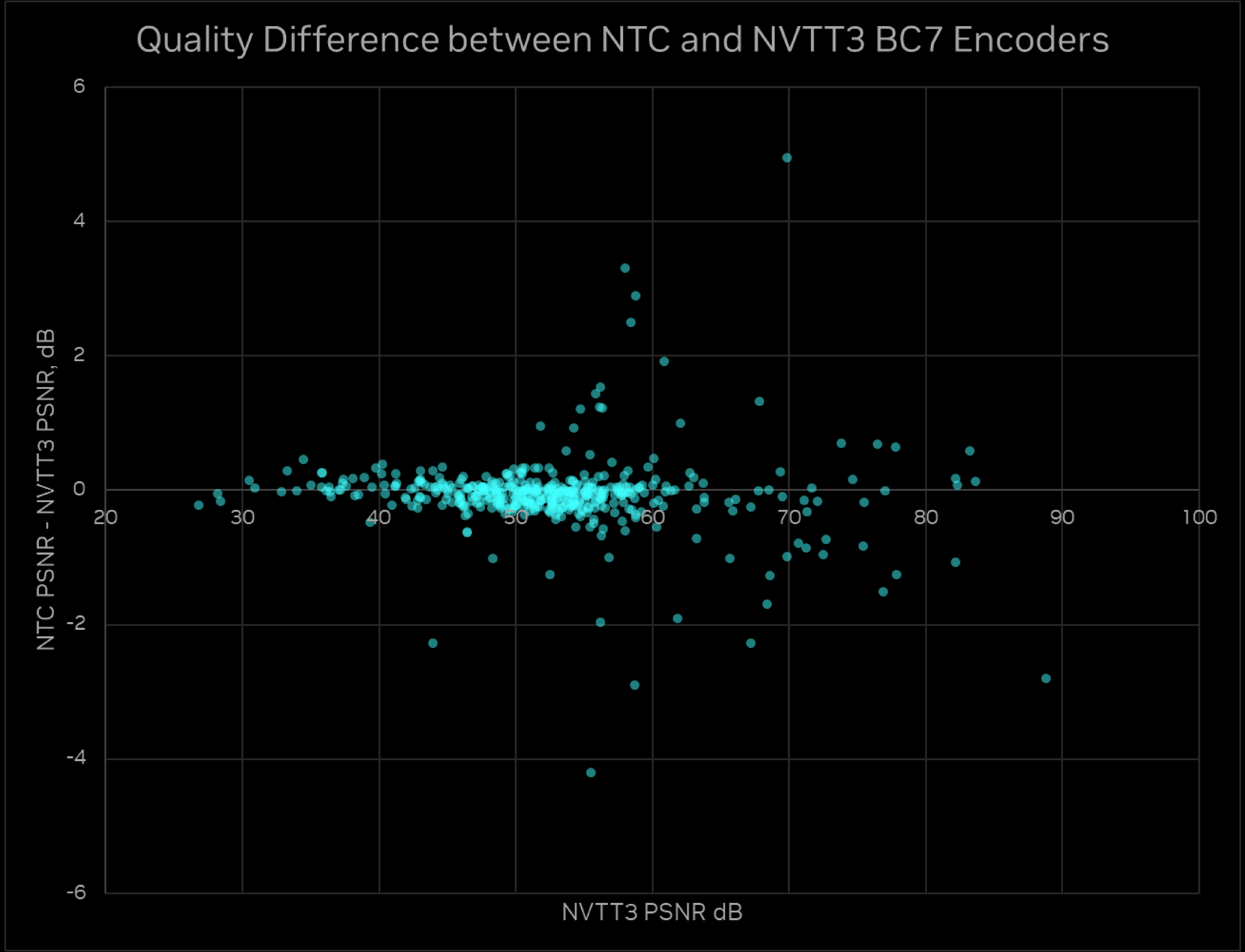
It's no longer slow

- LibNTC provides HLSL-based GPU encoders for all BC modes:
  - Custom, fast encoders for BC1-BC5
  - BC6 encoder from knarkowicz/GPURealTimeBC6H
  - Custom BC7 encoder with some tricks

- LibNTC BC7 encoder tricks:
  - We know the texture contents ahead of time
  - Pre-encode after NTC compression with all BC7 modes enabled
  - Count the most frequently used modes and partitions
  - Only use the necessary modes and partitions at transcoding time

- Encoding performance (on RTX 4090):

Format	Performance	Avg PSNR vs. NVTT3
BC1	60-70 GPix/s	-0.3 dB
BC5	30-40 GPix/s	+0.055 dB
BC7 (all modes)	1.5-1.8 GPix/s	-0.055 dB
BC7 (optimized modes)	3-12x faster	N/A (-0.2 dB for each)



Based on a set of 460 material textures

Inference on Tiled Texture Streaming

Work in progress

- Track which texture MIPs and tiles are needed
- Upload NTC latents for those tiles to the GPU
- Decompress the tiles and transcode them to BCn

- Benefits compared to other NTC modes:
  - No performance impact on the rendering passes
  - No need for Stochastic Texture Filtering
  - Extends existing texture streaming systems
- Benefits compared to existing solutions:
  - Reduced on-disk size of game assets
  - Potentially faster streaming (depends on system specs)

- Drawbacks:
  - More VRAM needed for the same texel count

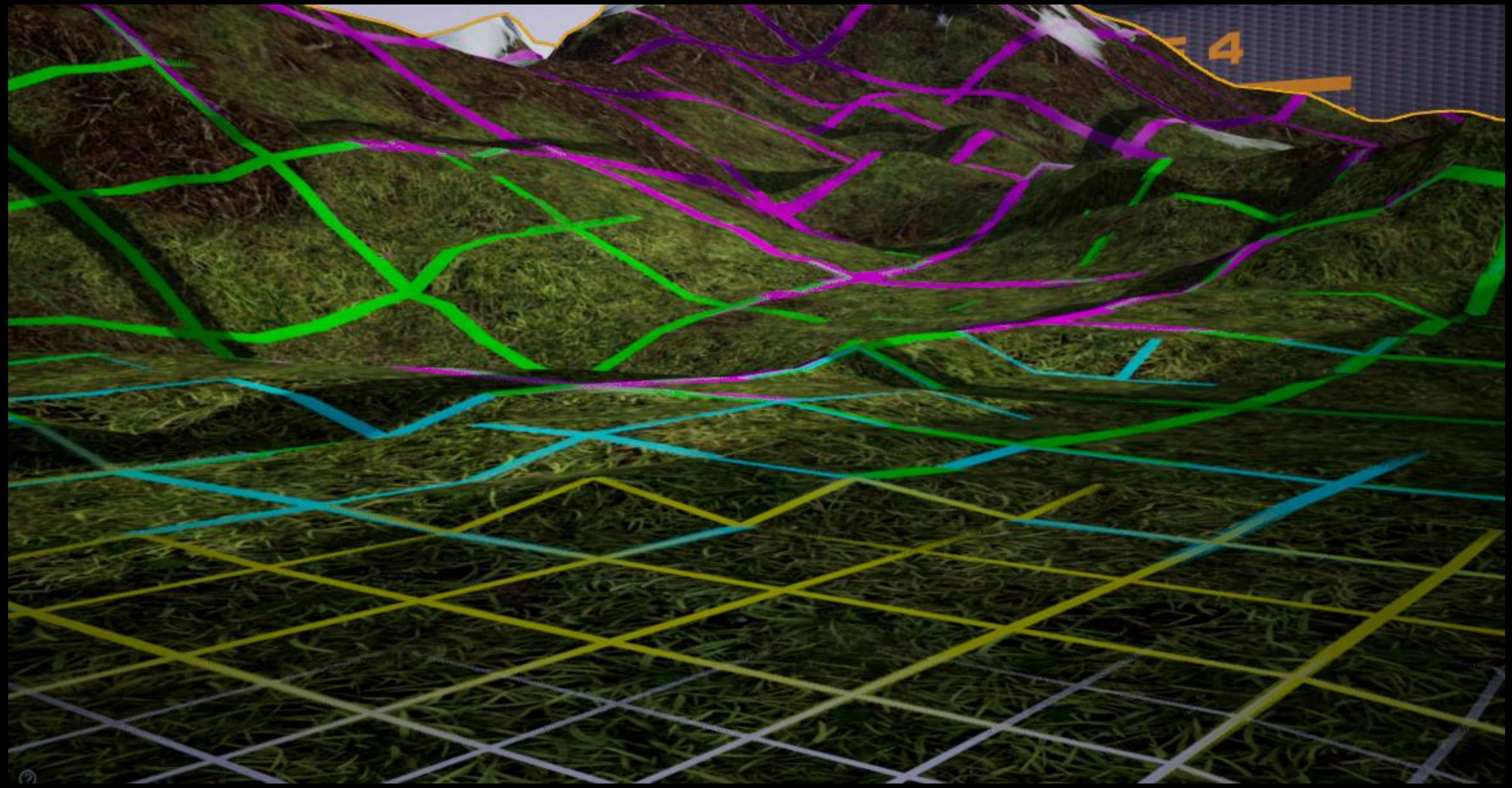


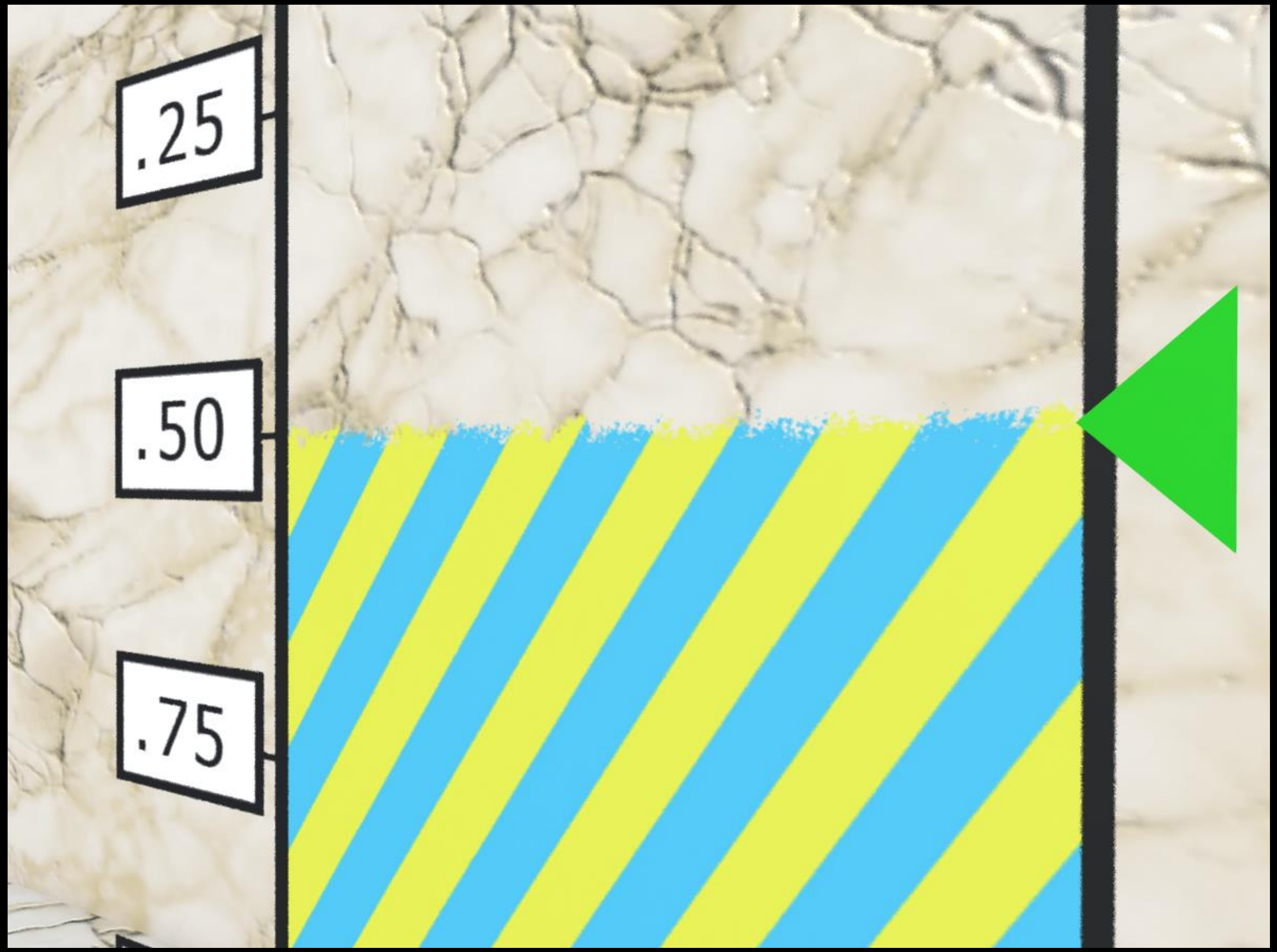
Image: Streaming Virtual Texturing in Unreal Engine 5

Practical Considerations

Contents of NTC Texture Sets

What to include and what to leave out

- BxDF inputs: include
  - This is what NTC was made for
- Alpha mask: it’s complicated
  - Too slow for inference in Z pre-pass or any-hit shaders
  - Too noisy for smooth gradients with alpha test →
  - OK quality for leaves and similar cutouts
  - Can improve NTC compression quality
  - Overall: Include but transcode to a separate BC4 texture on load
- Displacement: exclude
  - Not needed at the same time as the other channels
  - Displacement precision could be critical
  - Store separately as BC4 or BC6H
    - BC6H wastes 2 channels but is ½ the size of FP16

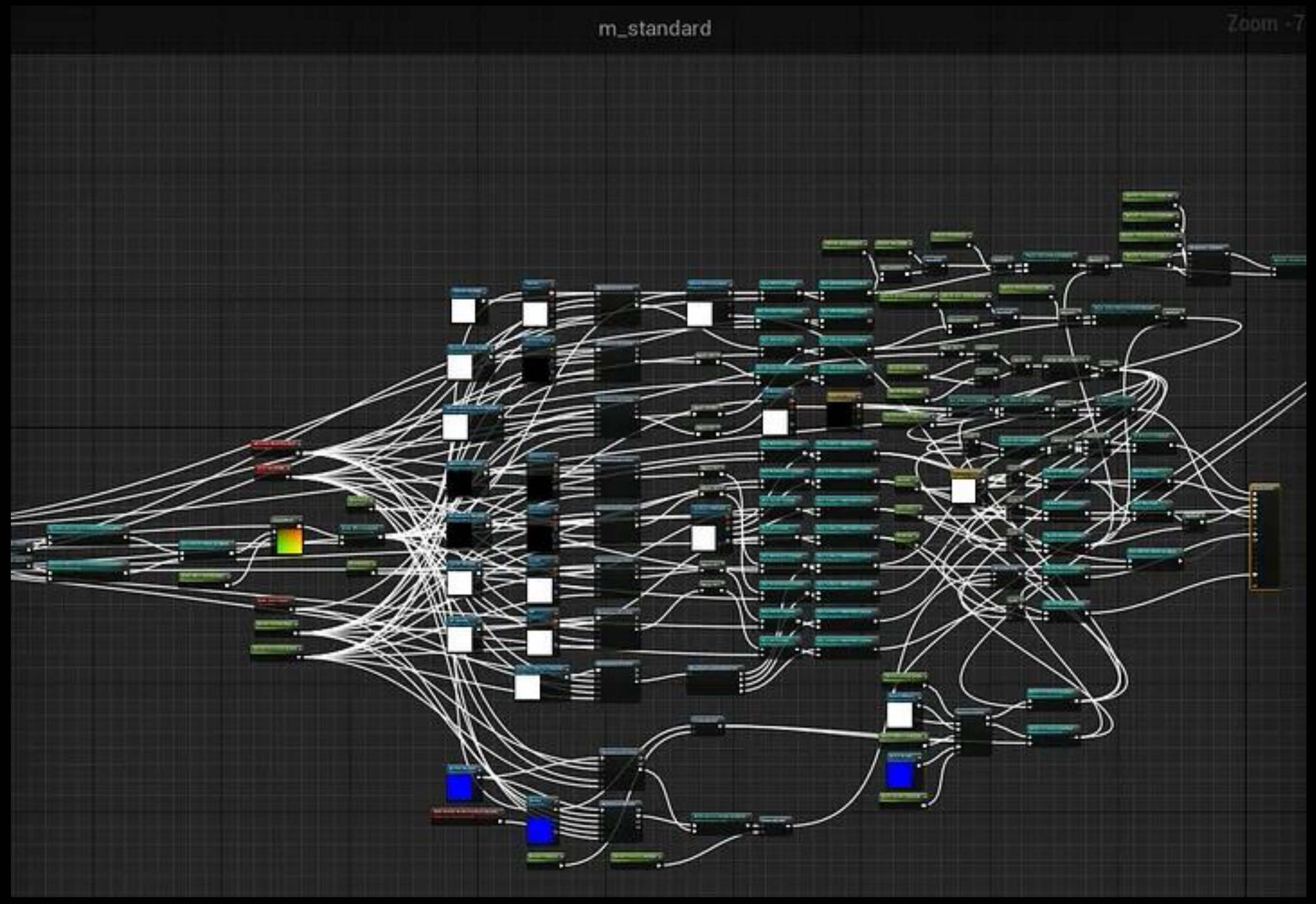


Fuzzy alpha mask in the GLTF Alpha Blend Mode Test

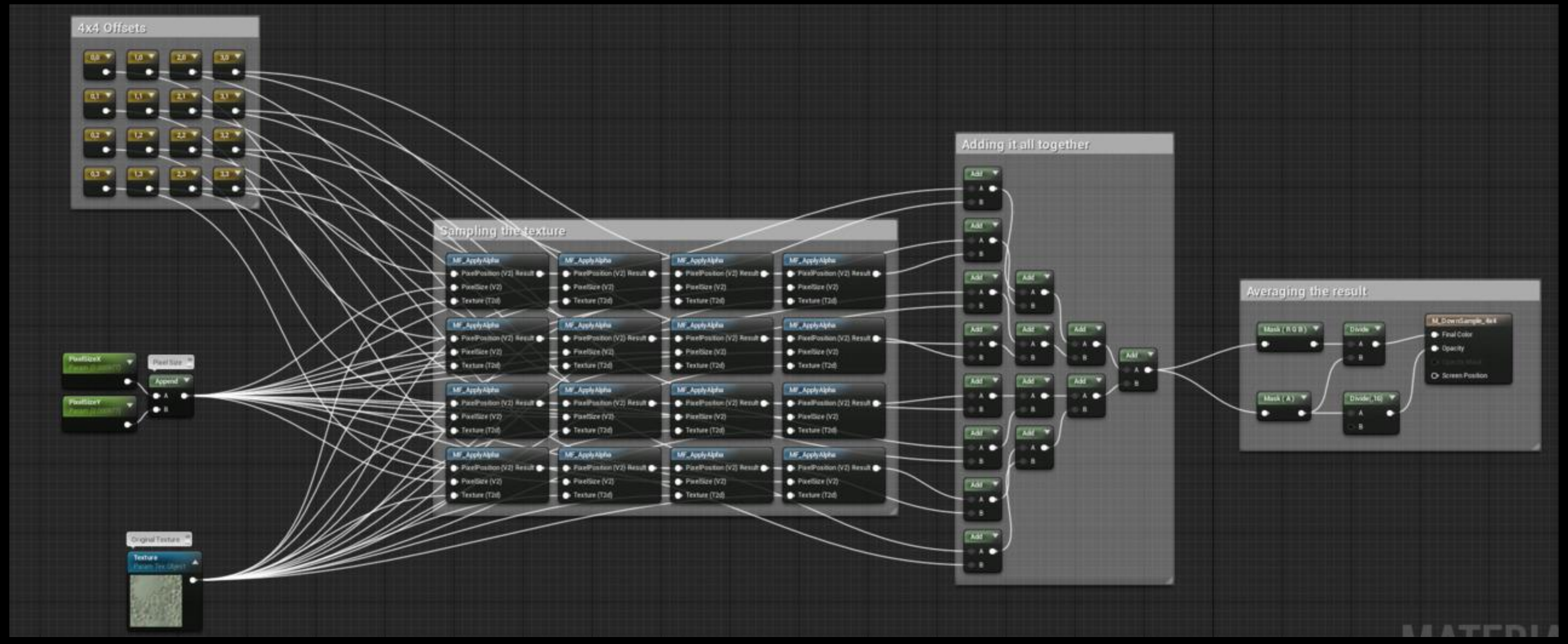
Integration Complexity

How difficult is NTC to implement in an engine?

- We don't have much data about this yes
- Asset pipeline: Easy to Difficult
  - Texture conversion tools provided in the SDK
  - Easy for fixed material models like glTF
  - Difficult to detect texture bundles in material graph systems
- Inference on Load: Easy
  - Injects a transcoding step into material loading phase
- Inference on Sample: Moderate to Difficult
  - Affects all shaders where materials are sampled
- Inference on Tiled Streaming: Difficult
  - Tight integration into the streaming system



https://medium.com/engineering-joy/materials-in-unreal-engine-4-28417504acd2



https://www.tumblr.com/blueprintsfromhell

Tech and API Readiness

When can I ship a game with NTC?

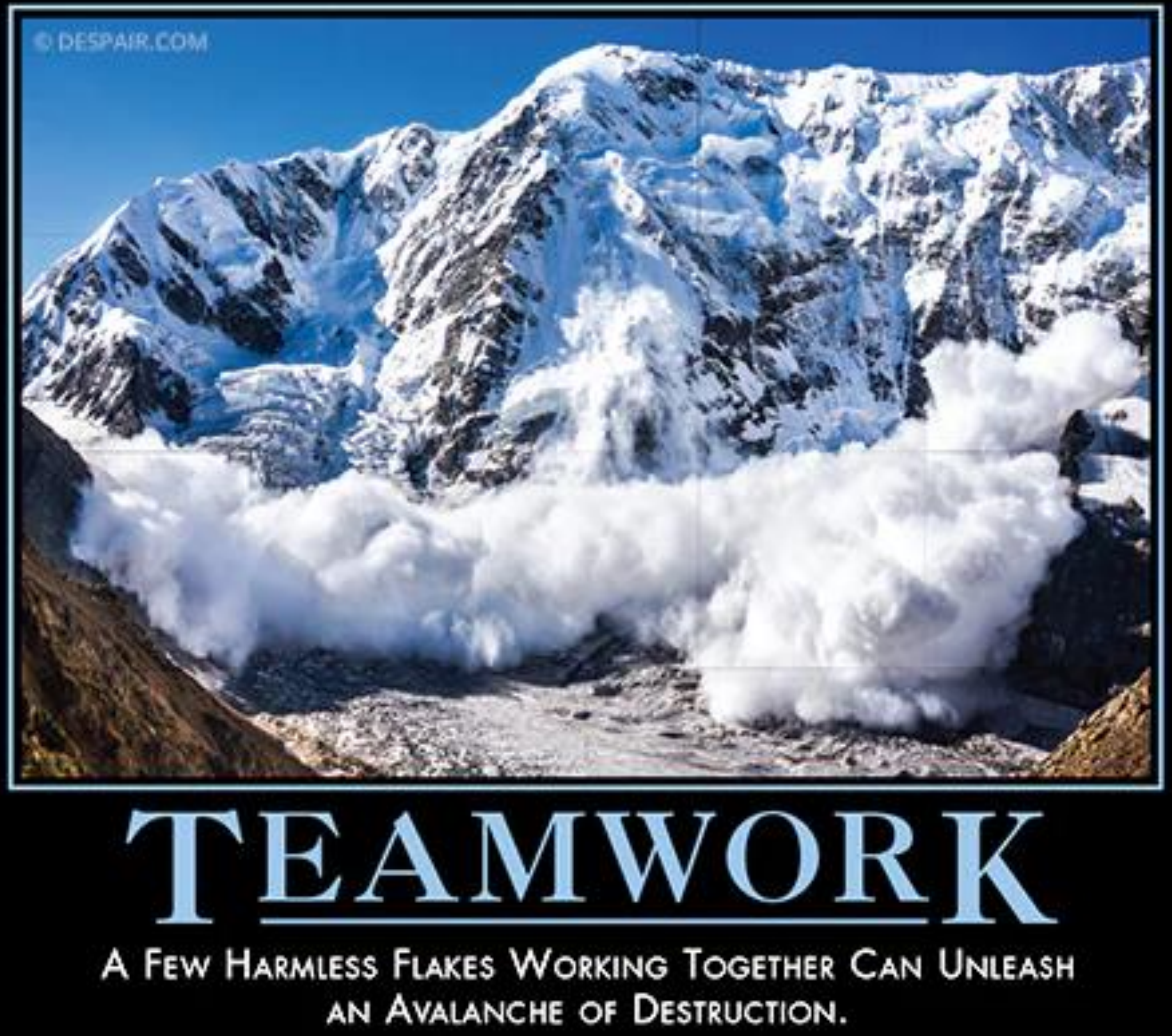
- Compression: Ready
- Inference on Load: Ready except…
  - VK_NV_cooperative_vector extension is final and stable
  - DP4a and other fallback versions are good
  - Don't ship with DX12 Cooperative Vectors until final API is released by Microsoft
    - Sometime mid-2025 maybe?
- BCn transcoding: Ready
- Inference on Sample: Ready except…
  - Performance in complex shaders could be a challenge
  - Same note about DX12 Cooperative Vectors applies

Conclusion

Let’s figure out the future of texture storage together!

- We provide basic tools for implementing NTC
  - Compression tools and samples
  - Inference and BCn transcoding code that works everywhere
- There are open practical questions better addressed by engine developers
  - Inference on tiled texture streaming
  - Texture bundles
- There are open research questions, too
  - Can we use smaller MLPs to improve performance?
  - Can we make Inference on Sample more efficient in complex shaders?
  - Can we improve compression quality without sacrificing performance?
  - Should we use a different quality metric instead of PSNR?
- …But what we already have is quite impressive, try it out!

  https://github.com/NVIDIA-RTX/RTXNTC


© DESPAIR.COM
TEAMWORK
A Few Harmless Flakes Working Together Can Unleash an Avalanche of Destruction.

Questions?

alpanteleev@nvidia.com

𝕏 @more_fps