CS 380 - GPU and GPGPU Programming
Lecture 27: Neural Shading on GPUs

(based on NVIDIA Siggraph course)

Markus Hadwiger, KAUST

Reading Assignment #15++

Read (optional):

- **SIGGRAPH 2025 course on Neural Shading, Bitterli et al. (NVIDIA)**

https://github.com/shader-slang/neural-shading-s25

- Random-Access Neural Compression of Material Textures, Vaidyanathan et al. (NVIDIA), 2023

https://arxiv.org/abs/2305.17105

- Real-Time Neural Materials using Block-Compressed Features, Weinreich et al. (Ubisoft), 2024

https://arxiv.org/abs/2311.16121

- Neural Texture Block Compression, Fujieda and Harada (AMD), 2024

https://gpuopen.com/download/2024_NeuralTextureBCCompression.pdf

- Filtering After Shading With Stochastic Texture Filtering, Pharr et al. (NVIDIA), 2024

https://arxiv.org/abs/2407.06107
https://research.nvidia.com/labs/rtr/publication/pharr2024stochtex/stochtex-slides.pdf

- Improved Stochastic Texture Filtering Through Sample Reuse, Wronski et al. (NVIDIA), 2025

https://arxiv.org/abs/2504.05562

- Real-Time Neural Appearance Models, Zeltner et al. (NVIDIA), 2024

https://research.nvidia.com/labs/rtr/neural_appearance_models/

- The Neural Light Grid: A Scalable Production-Ready Learned Irradiance Volume,
  Iwanicki et al. (Activision), 2024

https://www.activision.com/cdn/research/Neural_Light_Grid.pdf

- Decoding Light: Neural Compression of Global Illumination, Cao (Lightspeed Studios), 2025

https://gdcvault.com/play/1035521/Decoding-Light-Neural-Compression-for

Quiz #3: Dec 11

Organization

- First 30 min of lecture
- No material (book, notes, ...) allowed

Content of questions

- Lectures (both actual lectures and slides)
- Reading assignments
- Programming assignments (algorithms, methods)
- Solve short practical examples

Neural Shading

WHAT IS NEURAL SHADING?

SIGGRAPH 2025
Vancouver+ 10-14 August

- Usually: Involves Neural Networks
- Anything that's trainable

$\boxed{\text{Neural}}$ Shader

WHAT IS NEURAL SHADING?

- Usually: Involves Neural Networks
- Anything that's trainable

Neural $\boxed{\text{Shader}}$

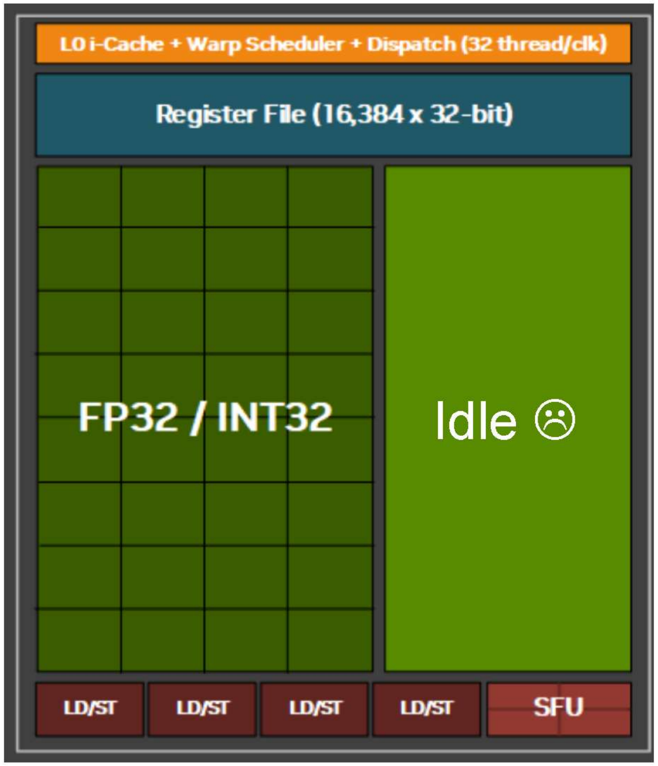
- Runs in the graphics pipeline
- Part of your normal shader code

HOW DO NEURAL SHADERS HELP?

- Consumer GPUs ship with neural network accelerators
- While you render, these sit at 0% utilization

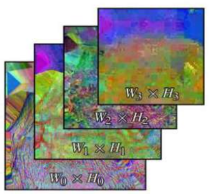
- These are untapped FLOPS!
- …and very efficient FLOPS

- Accessible in the graphics pipeline with Cooperative Vectors



HOW DO NEURAL SHADERS HELP?

[Fujieda and Harada, 2024]

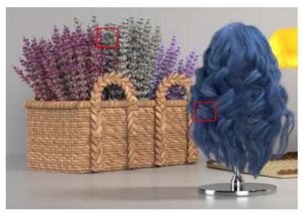


[Belcour and Benyoub, 2025]

[Vaidyanathan et al., 2023]

Compression

[Kuznetsov et al., 2021]



[Mullia et al., 2024]

[Zeltner et al., 2024]

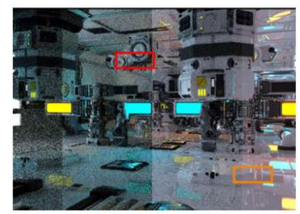
Materials

[Mildenhall et al., 2020]

[Müller et al., 2022]

[Kerbl et al., 2023]

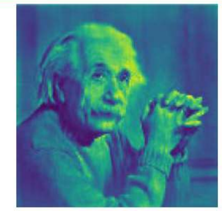
Geometry



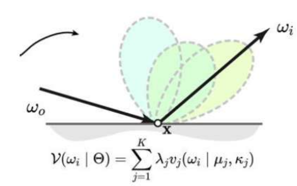
[Müller et al., 2021]

[Dereviannykh et al., 2024]

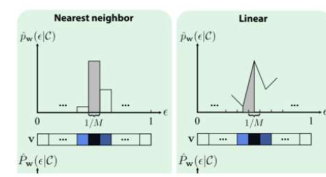
Caching



[Müller et al., 2019]



[Dong et al., 2023]



[Figueiredo et al., 2025]

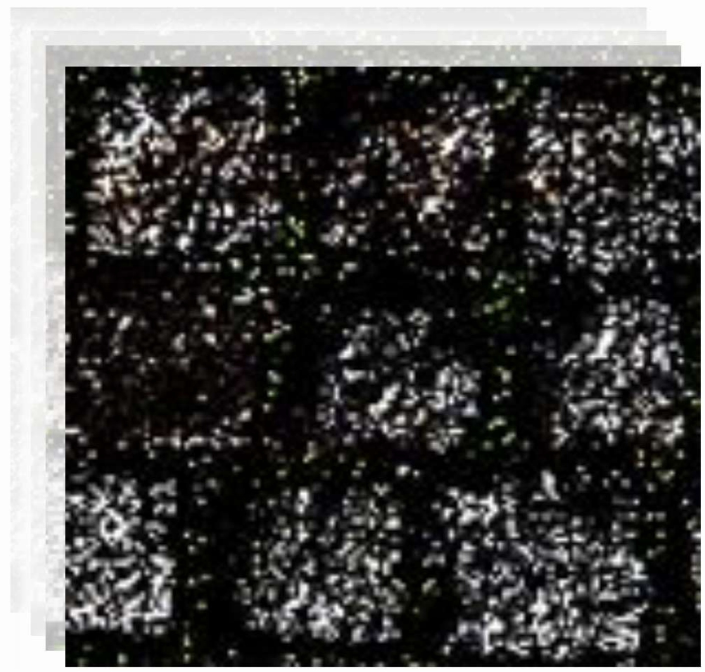
Guiding

CAN WE LEARN BETTER MIP MAPS?

- Mipmaps reduce aliasing **and** improve coherency for distant surfaces
- Color maps downsample well using simple box filter
- Normal maps do not

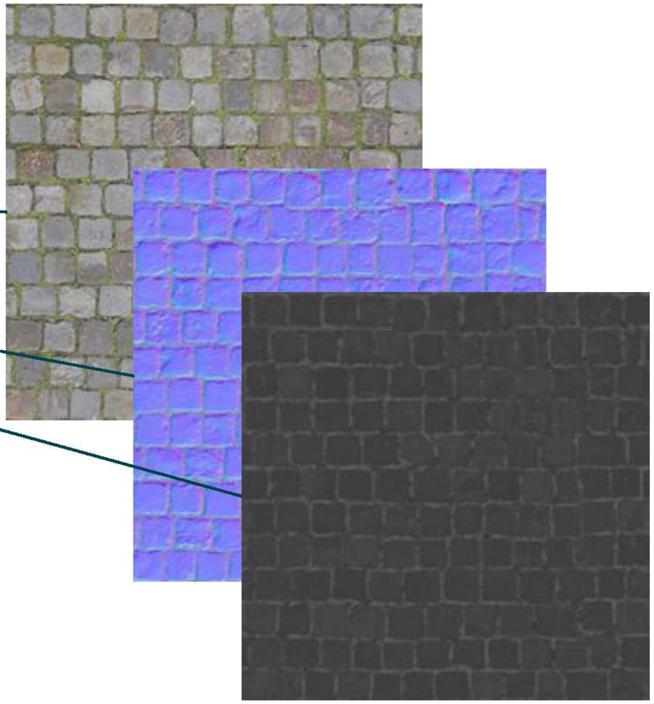
CAN WE LEARN BETTER MIP MAPS?

1. Render a beautiful PBR material
2. Generate + render low res with **Mip Maps**
3. Generate + render reference with **Supersampling**
4. Calculate the loss



BASIC SLANG PROGRAM


```
struct MaterialParameters
{
    Tensor<float3, 2> albedo;
    Tensor<float3, 2> normal;
    Tensor<float, 2> roughness;

    float3 get_albedo(int2 pixel)
    {
        return albedo.getv(pixel);
    }
    float3 get_normal(int2 pixel)
    {
        return normalize(normal.getv(pixel));
    }
    float get_roughness(int2 pixel)
    {
        return roughness.getv(pixel);
    }
};
```



BASIC SLANG PROGRAM

SIGGRAPH 2025
Vancouver+ 10-14 August

```
float3 render(int2 pixel, MaterialParameters material, float3 light_dir, float3 view_dir)
{
    // Bright white light
    float light_intensity = 5.0;

    // Sample very shiny BRDF (it rained today!)
    float3 brdf_sample = sample_brdf(
            material.get_albedo(pixel),  // albedo color
            normalize(light_dir),        // light direction
            normalize(view_dir),         // view direction
            material.get_normal(pixel),  // normal map sample
            0.05,                        // roughness
            0.0,                         // metallic (no metal)
            1.0                          // specular
            );

    // Combine light with BRDF sample to get pixel colour
    return brdf_sample * light_intensity;
}
```

BASIC PYTHON PROGRAM

SIGGRAPH 2025
Vancouver+ 10-14 August

```python
# Create the app and load the slang module.
app = App(width=2048, height=2048, title="Mipmap Example")
module = spy.Module.load_from_file(app.device, "nsc_basicprogram.slang")

# Load some materials.
albedo_map = spy.Tensor.load_from_image(app.device, "PavingStones070_2K.diffuse.jpg", linearize=True)
normal_map = spy.Tensor.load_from_image(app.device, "PavingStones070_2K.normal.jpg", scale=2, offset=-1)
roughness_map = spy.Tensor.load_from_image(app.device, "PavingStones070_2K.roughness.jpg", grayscale=True)

while app.process_events():
    # Allocate a tensor for output + call the render function
    output = spy.Tensor.empty_like(albedo_map)
    module.render(pixel = spy.call_id(),
                  material = {
                        "albedo": albedo_map,
                        "normal": normal_map,
                        "roughness": roughness_map,
                  },
                  light_dir = spy.math.normalize(spy.float3(0.2, 0.2, 1.0)),
                  view_dir = spy.float3(0, 0, 1),
                  _result = output)

    # Blit tensor to screen.
    app.blit(output)
    app.present()
```

RESULT!

SIGGRAPH 2025
Vancouver+ 10-14 August

Conveniently Shiny Cobblestones!

DOWNSAMPLING

SIGGRAPH 2025
Vancouver+ 10-14 August

```python
def downsample(source: spy.Tensor, steps: int) -> spy.Tensor:
    for i in range(steps):
        dest = spy.Tensor.empty(
                    device=app.device,
                    shape=(source.shape[0] // 2, source.shape[1] // 2),
                    dtype=source.dtype)
        module.downsample(spy.call_id(), source, _result=dest)
        source = dest
    return source
```

```
float3 downsample(
        int2 pixel,
        Tensor<float3, 2> source)
{
    float3 res = 0;
    res += source.getv(pixel * 2 + int2(0, 0));
    res += source.getv(pixel * 2 + int2(1, 0));
    res += source.getv(pixel * 2 + int2(0, 1));
    res += source.getv(pixel * 2 + int2(1, 1));
    return res * 0.25;
}
```

DOWNSAMPLED INPUTS

SIGGRAPH 2025
Vancouver+ 10-14 August

```
 # … downsampled output generation/render here

 # Quarter res rendered output BRDF from half res inputs.
 lr_output = spy.Tensor.empty_like(output)
 module.render(pixel = spy.call_id(),
               material = {
                     "albedo":    downsample(albedo_map, 2),
                     "normal":    downsample(normal_map, 2),
                     "roughness": downsample(roughness_map, 2),
               },
               light_dir = spy.math.normalize(spy.float3(0.2, 0.2, 1.0)),
               view_dir = spy.float3(0, 0, 1),
               _result = lr_output)

# … blit the ‘lr_output’ to screen here
```

RESULT!

SIGGRAPH 2025
Vancouver+ 10-14 August

Downscaled inputs

- 2x downsample (Mip level 2)
- 1 sample per map
- PBR function run once per pixel
- Optimal but noisy

Poor normal map downsampling gives nasty surface artifacts.

DOWNSAMPLED OUTPUT

SIGGRAPH 2025
Vancouver+ 10-14 August

```
while app.process_events():
    # Allocate a tensor for output + call the render function
    output = spy.Tensor.empty_like(albedo_map)
    module.render(pixel = spy.call_id(),
                  material = {
                        "albedo": albedo_map,
                        "normal": normal_map,
                        "roughness": roughness_map
                 },
                  light_dir = spy.math.normalize(spy.float3(0.2, 0.2, 1.0)),
                  view_dir = spy.float3(0, 0, 1),
                  _result = output)

    # Downsample the output tensor to quarter res.
    output = downsample(output, 2)

    # Blit tensor to screen.
    app.blit(output, size=spy.int2(2048,2048))
    app.present()
```

RESULT!

SIGGRAPH 2025
Vancouver+ 10-14 August

Downscaled output

- 2x downsample (Mip level 2)
- 16 high res albedo, normal + roughness pixels sampled
- Full PBR function run 16 times
- Represents an **ideal** output

If our input mip maps were *perfect*, this is what we'd get

SIDE BY SIDE

SIGGRAPH 2025
Vancouver+ 10-14 August

The Ideal
Render at full res then downsample

The Reality
Downsample inputs and then render

CALCULATING THE LOSS

SIGGRAPH 2025
Vancouver+ 10-14 August

```
// Render the difference between the rendered pixel
a reference pixel.
float3 loss(
      int2 pixel,
      float3 reference,
      MaterialParameters material,
      float3 light_dir,
      float3 view_dir)
{
    float3 color = render(pixel, material,
                          light_dir, view_dir);
    float3 error = color - reference;
    return error * error; // Squared error
}
```

```
# Loss between downsampled full res output (the
# reference), and result from quarter res inputs.
loss_output = spy.Tensor.empty_like(output)
module.loss(pixel = spy.call_id(),
         material = {
               "albedo": downsample(albedo_map,2),
            "normal": downsample(normal_map,2),
            "roughness": downsample(roughness_map,2),
         },
         reference = output,
         light_dir = spy.math.normalize(
                   spy.float3(0.2, 0.2, 1.0)),
         view_dir = spy.float3(0, 0, 1),
         _result = loss_output)
```

CALCULATING THE LOSS

SIGGRAPH 2025
Vancouver+ 10-14 August

Reference

Loss

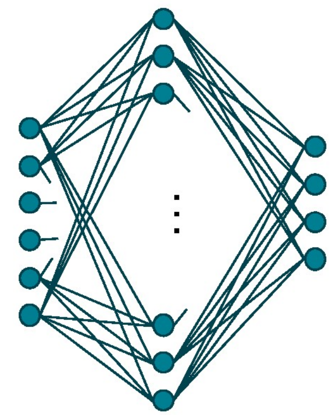
Render

Make mipmap better == Make the loss smaller

IMPLEMENTING OUR TOY NEURAL TEXTURE EXAMPLE

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NeuralTexture2D
{
    NetworkParameters<6, 32> layer1;
    NetworkParameters<32, 4> layer2;

    float4 sample(float2 uv, float2 du, float2 dv)
    {
        float input[6] = {uv.x, uv.y, du.x, du.y, dv.x, dv.y};
        let latentResult = layer1.eval(input);
        let output = layer2.eval(latentResult);
        return float4(output[0], output[1], output[2], output[3]);
    }
}
```



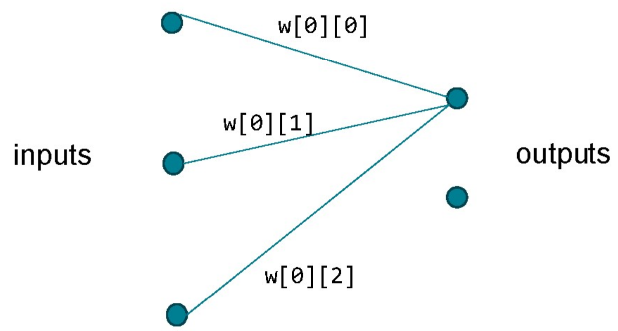
COMPUTATION OF A 3-INPUT, 2-OUTPUT LAYER

SIGGRAPH 2025
Vancouver+ 10-14 August

```
outputs[0] = biases[0] +
             inputs[0] * w[0][0] +
             inputs[1] * w[0][1] +
             inputs[2] * w[0][2];
outputs[0] = max(0.0, outputs[0]);
```

Weighted Sum of Inputs

Activation



COMPUTATION OF A 3-INPUT, 2-OUTPUT LAYER

SIGGRAPH 2025
Vancouver+ 10-14 August

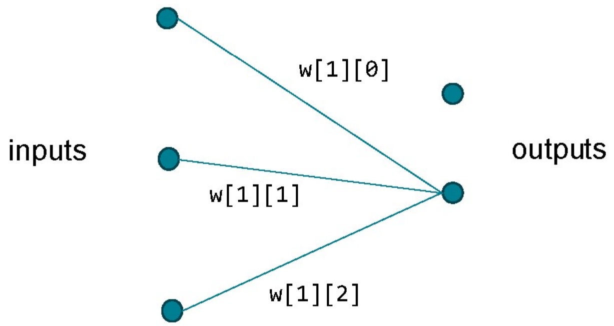
```
outputs[0] = biases[0] +
             inputs[0] * w[0][0] +
             inputs[1] * w[0][1] +
             inputs[2] * w[0][2];
outputs[0] = max(0.0, outputs[0]);

outputs[1] = biases[1] +
             inputs[0] * w[1][0] +
             inputs[1] * w[1][1] +
             inputs[2] * w[1][2];
outputs[1] = max(0.0, outputs[1]);
```

Weighted Sum of Inputs

Activation

Weighted Sum of Inputs

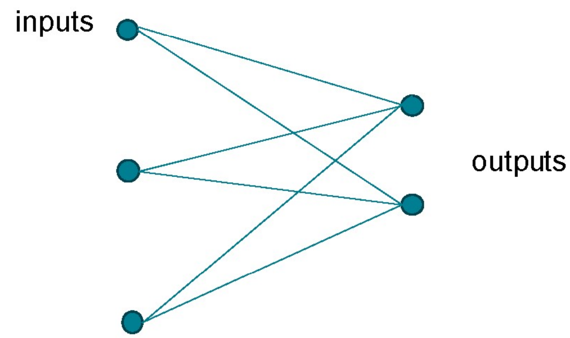
Activation



COMPUTATION OF A 3-INPUT, 2-OUTPUT LAYER


```
outputs[0] = biases[0] +
        inputs[0] * w[0][0] +
        inputs[1] * w[0][1] +
        inputs[2] * w[0][2];
outputs[0] = max(0.0, outputs[0]);

outputs[1] = biases[1] +
        inputs[0] * w[1][0] +
        inputs[1] * w[1][1] +
        inputs[2] * w[1][2];
outputs[1] = max(0.0, outputs[1]);
```

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} + \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{bmatrix} \begin{bmatrix} i_0 \\ i_1 \\ i_2 \end{bmatrix}$$



SIMPLIFIED VIEW OF A FEED-FORWARD LAYER

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    float weights[InputSize][OutputSize];
    float biases[OutputSize];
    Array<float, OutputSize> eval(float input[InputSize])
    {
        float output[OutputSize] = biases;
        output += MatMulVec(weights, input);
        for (int i = 0; i < OutputSize; ++i)
            output[i] = max(output[i] * 0.01f, output[i]); // ReLU
        return output;
    }
}
```

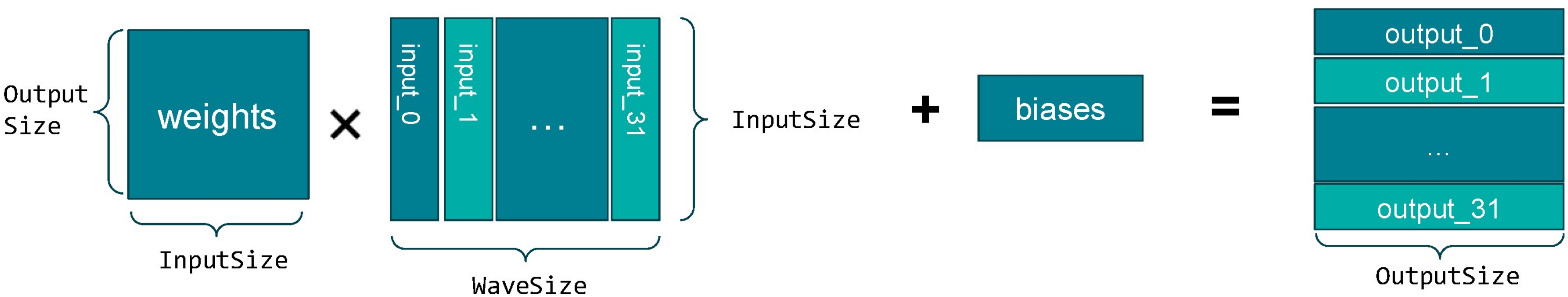
EACH SUBGROUP IS COMPUTING A MATRIX-MATRIX MULTIPLICATION

SIGGRAPH 2025
Vancouver+ 10-14 August

```
Array<float, OutputSize> eval(float input[InputSize])
{
    float output[OutputSize] = biases;
    output += MatMulVec(weights, input);
    ...
    return output;
}
```

Thread	Computation
0	MatMulVec(weights, input_0)
1	MatMulVec(weights, input_1)
2	MatMulVec(weights, input_2)
3	MatMulVec(weights, input_3)
…	
31	MatMulVec(weights, input_31)

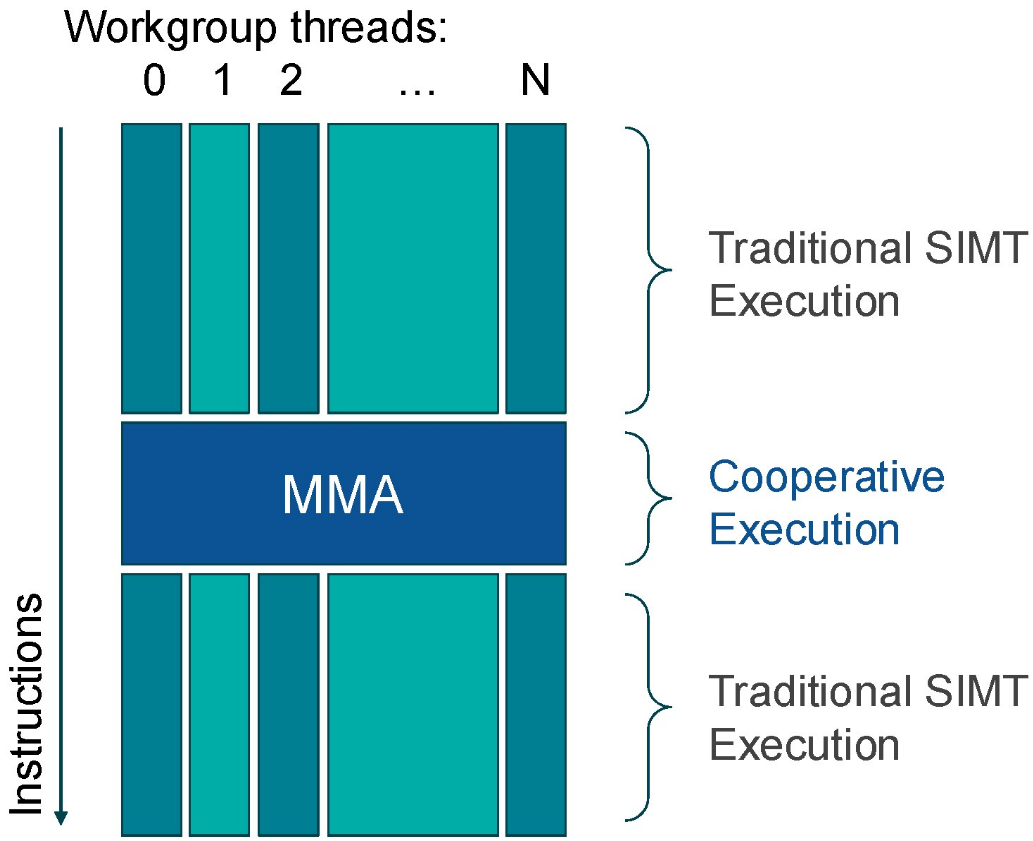
Computation done by the entire subgroup: **M**atrix **M**ultiply **A**ccumulate (a.k.a. **GE**neral **M**atrix **M**ultiply)



- *Wave == *Subgroup(Vulkan)* == *Warp* (CUDA) == *SIMD Group* (Metal)

MODERN GPUS HAVE HARDWARE SUPPORT FOR MATRIX-MULTIPLY-ACCUMULATION

SIGGRAPH 2025
Vancouver+ 10-14 August



MMA EXPOSED AS INTRINSICS IN VARIOUS APIS

DirectX® 12 — WaveMatrix

Vulkan™ — VK_KHR_cooperative_matrix

VK_NV_cooperative_matrix2

SIMD Group Matrix

MMA INTRINSICS RESTRICTIONS

- Matrix operands in a Matrix-Multiply-Accumulation operation must be the same across all threads in a subgroup.

- Input vectors must be explicitly packed into a matrix and loaded in one operation.
  - Often require explicit use of group shared memory, which isn't available outside compute shaders.

COOPERATIVE VECTORS: USING MMA HARDWARE FROM SIMT CODE

DirectX® 12

Shader Model 6.9

Vulkan™

VK_NV_cooperative_vector

Metal 4
Machine Learning

SIMPLIFIED VIEW OF A FEED-FORWARD LAYER

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    float weights[InputSize][OutputSize];
    float biases[OutputSize];
    Array<float, OutputSize> eval(float input[InputSize])
    {
        float output[OutputSize] = biases;
        output += MatMulVec(weights, input);
        for (int i = 0; i < OutputSize; ++i)
            output[i] = max(output[i] * 0.01f, output[i]); // ReLU
        return output;
    }
}
```

FEED-FORWARD LAYER USING COOPERATIVE VECTORS

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    half* weights;
    half* biases;
    CoopVec<half, OutputSize> eval(CoopVec<half, InputSize> input)
    {
        let output = coopVecMatMulAdd<half, OutputSize>(
            input, CoopVecComponentType.Float16, // input and format
            weights, CoopVecComponentType.Float16, // weights and format
            biases, CoopVecComponentType.Float16, // biases and format
            CoopVecMatrixLayout.RowMajor, // matrix layout
            false, // transpose matrix before multiply?
            sizeof(half) * InputSize); // matrix stride
        return max(output * 0.01h, output); // Leaky ReLU activation
    }
}
```

Using 16-bit floats because 32-bit float is not supported by HW.

TRAINING WITH COOPERATIVE VECTORS

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    half* weights;
    half* biases;

    CoopVec<half, OutputSize> eval(CoopVec<half, InputSize> input)
    {
        let output = coopVecMatMulAdd<half, OutputSize>(
            input, CoopVecComponentType.Float16, // input and format
            weights, CoopVecComponentType.Float16, // weights and format
            biases, CoopVecComponentType.Float16, // biases and format
            CoopVecMatrixLayout.RowMajor, // matrix layout
            false, // transpose matrix before multiply?
            sizeof(half) * InputSize); // matrix stride
        return max(output * 0.01h, output); // Leaky ReLU activation
    }
}
```

1. Where should we store the gradients of the weights?

2. How to make this differentiable?

STORING THE GRADIENTS OF THE WEIGHTS

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    half* weights;
    half* biases;
    half* weightsGrad;
    half* biasesGrad;

}
```

1. Where should we store the gradients of the weights?

TRAINING WITH COOPERATIVE VECTORS

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    ...

    CoopVec<half, OutputSize> eval(CoopVec<half, InputSize> input)
    {
        let output = coopVecMatMulAdd<half, OutputSize>(
            input, CoopVecComponentType.Float16, // input and format
            weights, CoopVecComponentType.Float16, // weights and format
            biases, CoopVecComponentType.Float16, // biases and format
            CoopVecMatrixLayout.RowMajor, // matrix layout
            false, // transpose matrix before multiply?
            sizeof(half) * InputSize); // matrix stride
        return max(output * 0.01h, output); // Leaky ReLU activation
    }
}
```

How to make `eval` differentiable?

- `CoopVec` type is non-differentiable.
- `coopVecMatMulAdd` intrinsic is non-differentiable.

WRAPPING COOPERATIVE VECTOR IN A DIFFERENTIABLE TYPE

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct MLVec<int N> : IDifferentiable
{
    CoopVec<half, N> data;
    typealias Differential = This;

    static Differential dadd(Differential d0, Differential d1)
    {
        return {d0.data + d1.data};
    }
    static Differential dmul<U:__BuiltinRealType>(U s, Differential d)
    {
        return {d.data * __realCast<half>(s)};
    }
    static Differential dzero()
    {
        return {};
    }

}
```

Wrap CoopVec in a MLVec type, that is declared to be Differentiable, with Differential type being itself.

UPDATE EVAL TO USE MLVEC TYPE

SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    ...
    half* weightsGrad;
    half* biasesGrad;

    MLVec<OutputSize> eval(MLVec<InputSize> input) { ... }


}
```

MAKE EVAL DIFFERENTIABLE WITH CUSTOM DERIVATIVE


```
struct NetworkParameters<int InputSize, int OutputSize>
{
    ...
    half* weightsGrad;
    half* biasesGrad;

    MLVec<OutputSize> eval(MLVec<InputSize> input) { ... }

    [BackwardDerivativeOf(eval)]
    void evalBwd(
        inout DifferentiablePair<MLVec<InputSize> input,
        MLVec<OutputSize> resultGrad)
    {
         // What goes here??
    }
}
```

Task: propagate gradients from each output element (`resultGrad`) to:

- weight and bias parameters (to store in `weightsGrad` and `biasesGrad`).
- Each input element (return via `input.d`)

GRADIENT PROPAGATION FOR MATRIX-VECTOR MULTIPLICATION

SIGGRAPH 2025
Vancouver+ 10-14 August

Given matrix-vector multiplication

$$\boldsymbol{u} = \mathrm{M}\boldsymbol{v} + B$$

We have:

$$d\mathrm{M} = d\mathbf{u} \otimes_{\mathrm{outer}} \boldsymbol{v}$$

$$d\mathrm{B} = d\mathbf{u}$$

$$d\mathbf{v} = \mathrm{M}^{T}\, d\mathbf{u}$$

$\otimes_{\mathrm{outer}}$: outer vector product

ACCUMULATING GRADIENTS ACROSS THREADS

Given matrix-vector multiplication

$$\boldsymbol{u} = \mathrm{M}\boldsymbol{v} + B$$

We have:

$$d\mathrm{M} = d\mathbf{u} \otimes_{\text{outer}} \boldsymbol{v}$$

$$d\mathrm{B} = d\mathbf{u}$$

$$d\mathbf{v} = \mathrm{M}^T \, d\mathbf{u}$$

$\otimes_{\text{outer}}$: outer vector product

M

dM

B

dB

Thread	Computation
0	u = malMulVecAdd(M, v0, B)
1	u = malMulVecAdd(M, v1, B)
2	u = malMulVecAdd(M, v2, B)
…	
N	u = malMulVecAdd(M, vN, B)

Each thread accesses different v, but the same M and B

ACCUMULATING GRADIENTS ACROSS THREADS

SIGGRAPH 2025
Vancouver+ 10-14 August

Given matrix-vector multiplication

$$\boldsymbol{u} = \mathrm{M}\boldsymbol{v} + B$$

We have:

$$d\mathrm{M} \mathrel{+}= d\mathbf{u} \otimes_{\text{outer}} \boldsymbol{v}$$

$$d\mathrm{B} \mathrel{+}= d\mathbf{u}$$

$$d\mathbf{v} = \mathrm{M}^{T}\, d\mathbf{u}$$

M

dM

B

dB

Thread	Computation
0	u = malMulVecAdd(M, v0, B) dM += outerProduct(…) dB += dU
1	u = malMulVecAdd(M, v1, B) dM += outerProduct(…) dB += dU
2	u = malMulVecAdd(M, v2, B) dM += outerProduct(…) dB += dU
…	
N	u = malMulVecAdd(M, vN, B) dM += outerProduct(…) dB += dU

Needs to be atomic to prevent race conditions!

INTRINSICS TO SPEEDUP ATOMIC GRADIENT ACCUMULATION

SIGGRAPH 2025
Vancouver+ 10-14 August

Given matrix-vector multiplication

$$\boldsymbol{u} = \mathrm{M}\boldsymbol{v} + B$$

We have:

$$d\mathrm{M} \mathrel{+}= d\mathbf{u} \otimes_{\text{outer}} \boldsymbol{v}$$

$$d\mathrm{B} \mathrel{+}= d\mathbf{u}$$

$$d\mathbf{v} = \mathrm{M}^{T}\, d\mathbf{u}$$

Each step has its corresponding coop-vector intrinsic:

`coopVecOuterProductAccumulate`

`coopVecReduceSumAccumulate`

`coopVecMatMul`

BACK-PROP GRADIENTS THROUGH EVAL

SIGGRAPH 2025
Vancouver+ 10-14 August

```
[BackwardDerivativeOf(eval)]
void evalBwd(
    inout DifferentialPair<MLVec<InputSize>> input,
    MLVec<OutputSize> resultGrad)
{
    let fwd = eval(input.p);
    // Back-prop resultGrad through activation.
    for (int i = 0; i < OutputSize; i++)
        if (fwd.data[i] < 0.0)
            resultGrad.data[i] *= 0.01h;

    // Back-prop gradients to the weights matrix.
    coopVecOuterProductAccumulate(
        resultGrad.data,
        input.v.data,
        weightsGrad, 0,
        CoopVecMatrixLayout.TrainingOptimal, CoopVecComponentType.Float16);

    // Back-prop gradients to the biases vector.
    coopVecReduceSumAccumulate(resultGrad.data, biasesGrad, biasesOffset);
```

`coopVecOuterProductAccumulate` **requires** `weightsGrad` to be stored in `TrainingOptimal` layout.

$$d\mathrm{M} \mathrel{+}= d\mathbf{u} \otimes_{\mathrm{outer}} \boldsymbol{v}$$

$$d\mathrm{B} \mathrel{+}= d\mathbf{u}$$

BACK-PROP GRADIENTS THROUGH EVAL

SIGGRAPH 2025
Vancouver+ 10-14 August

```
[BackwardDerivativeOf(eval)]
void evalBwd(
    inout DifferentialPair<MLVec<InputSize>> input,
    MLVec<OutputSize> resultGrad)
{
    ...
    // Back-prop gradients to the input vector.
    let dInput = coopVecMatMul<half, InputSize>(
        resultGrad.data, CoopVecComponentType.Float16,
        weights, CoopVecComponentType.Float16,
        CoopVecMatrixLayout.ColumnMajor, false, sizeof(half)*InputSize);
    input = {input.p, {dInput}};
}
```

$$d\mathbf{v} = \mathrm{M}^T\, d\mathbf{u}$$

Compute
i.e. `transpose(weights) * resultGrad`
by setting the layout of weights to **`ColumnMajor`**.

CALLERS OF FEED-FORWARD LAYER CAN NOW BE AUTODIFF'ED

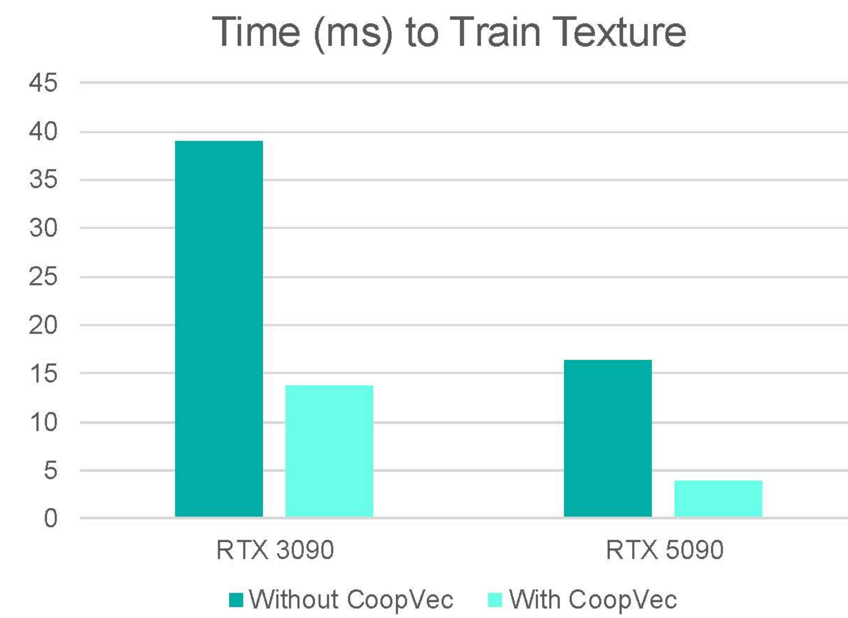
SIGGRAPH 2025
Vancouver+ 10-14 August

```
struct NeuralTexture2D
{
    NetworkParameters<6, 32> layer1;
    NetworkParameters<32, 4> layer2;

    [Differentiable]
    float4 sample(float2 uv, float2 du, float2 dv)
    {
        float input[6] = {uv.x, uv.y, du.x, du.y, dv.x, dv.y};
        let latentResult = layer1.eval(input);
        let output = layer2.eval(latentResult);
        return float4(output[0], output[1], output[2], output[3]);
    }
}
```

PERFORMANCE IMPROVEMENTS WITH COOP-VECTOR

- **Benchmark:** Neural Texture Training
- **Input:** **100MB** PNG

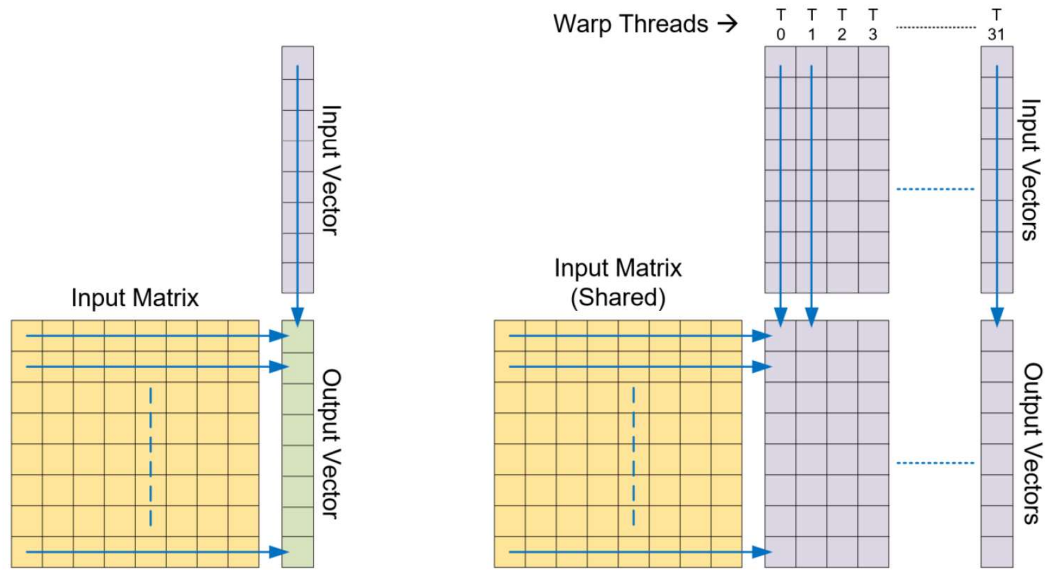
Over **3x** speedup compared to highly optimized code without coop-vec.



BATCHED MATRIX-VECTOR MULTIPLICATION

SIGGRAPH 2025
Vancouver+ 10-14 August

- Hardware Tensor Cores:
  - Matrix-Matrix multiplication using entire wave/warp
  - Low precision (FP16, FP8, INT8, even FP4)
- Cooperative Vector API:
  - Matrix-Vector multiplication in each thread
- Mapping CoopVec onto Tensor Cores:
  - Combine M-V multiplications from all threads in a wave
  - Divergence becomes a problem (more on that later)



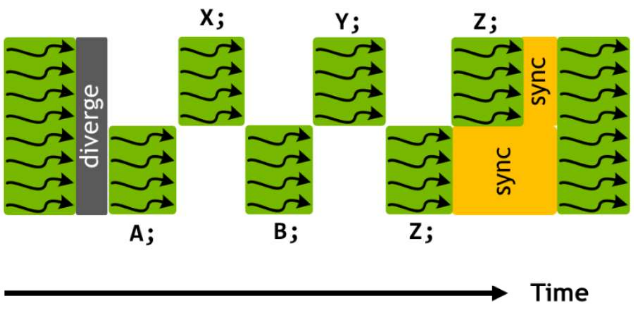
Single Thread:
Matrix-Vector Multiplication

Many Threads:
Matrix-Matrix Multiplication
Maps to Tensor Cores

REFRESHER ON GPU SIMT ARCHITECTURE

- NVIDIA GPUs group execution threads into 32-thread “warps”
  - Other vendors have similar concepts with different thread counts
  - We use the term “wave” throughout this course for consistency
- All threads in a wave execute one instruction on different data
  - Some may be inactive
  - Following different code branches in a wave causes execution divergence
  - Divergent code branches are serialized into synchronous groups
- Memory accesses work best when addresses are packed and aligned
  - Not following optimal address patterns causes data divergence
  - Divergent memory accesses are serialized into optimal groups

```
if (threadIdx.x < 4) {
    A;
    B;
} else {
    X;
    Y;
}
Z;
__syncwarp()
```

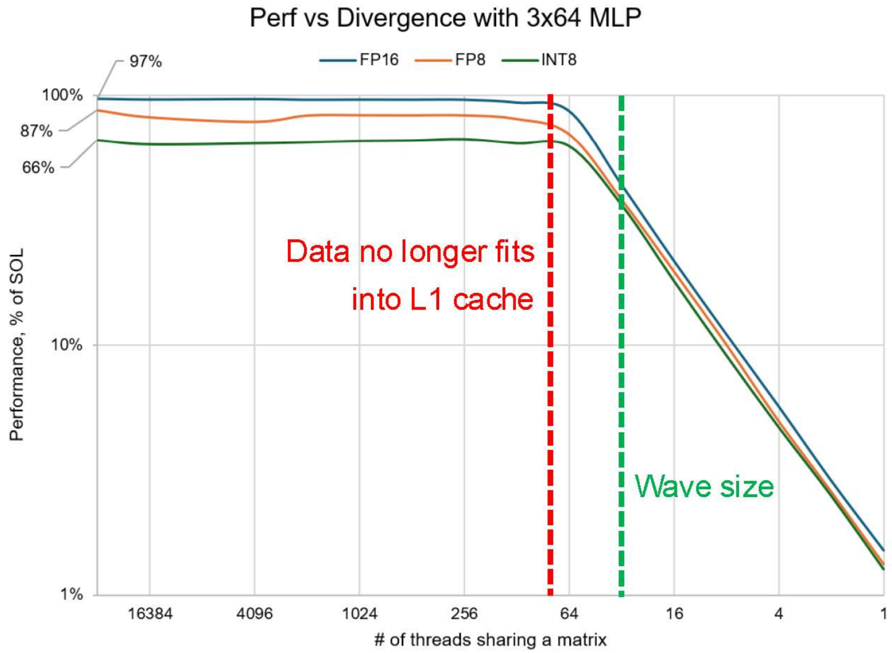


Source: Volta architecture whitepaper

TENSOR CORES AND DIVERGENCE

SIGGRAPH 2025
Vancouver+ 10-14 August

- Tensor Cores compute a single MMA using the entire wave
  - MMA = Matrix Multiply Accumulate
  - We can wake up inactive threads if needed
- Wave-wide coopVecMatMul[Add] might be incompatible:
  - Vector inputs are trivially combined into a matrix
  - Matrix inputs must be the same for all threads
  - Programming model allows matrix inputs to be different per-thread
- Solution: Serialize divergent matrix operations
  - Handled transparently by the driver
  - Has severe performance effects
- Avoid CoopVec operations with divergent matrices
  - Group draw calls by material
  - Sort threads manually or use Shader Execution Reordering (SER)



Measured on RTX 4090 using a directed test – Log/Log scale

https://github.com/jeffbolznv/vk_cooperative_vector_perf

TENSOR CORE MATRIX LAYOUT PROBLEM

- Tensor cores use custom matrix layouts
  - Components of matrices are shuffled between threads
  - Specific layout depends on the GPU and data types
- Need to shuffle data before and after MMAs
  - Weight matrix must be pre-shuffled in memory
  - DX12 and Vulkan provide functions to shuffle the weights
  - Input and output shuffling is handled transparently by the driver
- Output of one MMA can be used as input of another MMA without shuffle
  - Redundant shuffles can be eliminated

MMA computation 1

Row\ Col	0	1	2	3	4	5	6	7
0	T0 : { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
3	T3: { c0, c1, c2, c3, c4, c5, c6, c7 }							
4	T16: { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
7	T19: { c0, c1, c2, c3, c4, c5, c6, c7 }							

MMA computation 3

Row\Col	0	1	2	3	4	5	6	7
0	T8 : { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
3	T11: { c0, c1, c2, c3, c4, c5, c6, c7 }							
4	T24: { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
7	T27: { c0, c1, c2, c3, c4, c5, c6, c7 }							

MMA computation 2

Row\Col	0	1	2	3	4	5	6	7
0	T4 : { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
3	T7: { c0, c1, c2, c3, c4, c5, c6, c7 }							
4	T20: { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
7	T23: { c0, c1, c2, c3, c4, c5, c6, c7 }							

MMA computation 4

Row\Col	0	1	2	3	4	5	6	7
0	T12 : { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
3	T15: { c0, c1, c2, c3, c4, c5, c6, c7 }							
4	T28: { c0, c1, c2, c3, c4, c5, c6, c7 }							
..								
7	T31: { c0, c1, c2, c3, c4, c5, c6, c7 }							

Output matrix layout for an FP16 MMA operation (maps to 4 instructions)
https://docs.nvidia.com/cuda/parallel-thread-execution/index.html

WEIGHT MATRIX LAYOUT CONVERSIONS

SIGGRAPH 2025
Vancouver+ 10-14 August

- Query shuffled matrix size from VK/DX12
  - Could be larger than the optimally packed data
- Upload row- or column-major matrix to GPU
- Convert layout (perform the shuffle) on the GPU
  - Inference Optimal
  - Training Optimal
- Use converted matrix in CoopVec operations
  - Specify layout as `coopVecMatMulAdd` argument

FEED-FORWARD LAYER USING COOPERATIVE VECTORS

```
struct NetworkParameters<int InputSize, int OutputSize>
{
    half* weights;
    half* biases;
    CoopVec<half, OutputSize> eval(CoopVec<half, InputSize> input)
    {
        let output = coopVecMatMulAdd<half, OutputSize>(
            input, CoopVecComponentType.Float16,
            weights, CoopVecComponentType.Float16,
            biases, CoopVecComponentType.Float16,
            CoopVecMatrixLayout.InferencingOptimal,
            false, // is matrix transposed
            sizeof(half) * InputSize); // matrix stride
        return max(output * 0.01h, output);
    }
}
```

PREVIOUS SECTION

MATRIX LAYOUT CONVERSION APIS

SIGGRAPH 2025
Vancouver+ 10-14 August

Vulkan (VK_NV_cooperative_vector)

- `vkConvertCooperativeVectorMatrixNV`
  - CPU conversion / size query
- `vkCmdConvertCooperativeVectorMatrixNV`
  - GPU conversion

DX12 (Agility SDK 717 Preview)

- `Device::GetLinearAlgebraMatrixConversion…`
  `DestinationInfo`
  - Size query
- `CommandList::ConvertLinearAlgebraMatrix`
  - GPU conversion
  - Note: No CPU conversion function!

MAPPING AN MLP ONTO TENSOR CORES 1/3

SIGGRAPH 2025
Vancouver+ 10-14 August

ORIGINAL CODE

```
coopVecMatMul(layer0, input, weightBuffer, offset0);
layer0 = max(layer0, 0);

coopVecMatMul(layer1, layer0, weightBuffer, offset1);
layer1 = max(layer1, 0);

coopVecMatMul(output, layer1, weightBuffer, offset2);
```

Expensive shuffles

ADD SHUFFLES AROUND MMA OPS

```
shflinput = shuffle(input);
coopVecMatMul(temp0, shflinput, weightBuffer, offset0);
layer0 = unshuffle(temp0);

layer0 = max(layer0, 0);

shfllayer0 = shuffle(layer0);
coopVecMatMul(temp1, shfllayer0, weightBuffer, offset1);
layer1 = unshuffle(temp1);

layer1 = max(layer1, 0);

shfllayer1 = shuffle(layer1);
coopVecMatMul(temp2, shfllayer1, weightBuffer, offset2);
output = unshuffle(temp2);
```

Baseline implementation – functional but slow

MAPPING AN MLP ONTO TENSOR CORES 2/3

SIGGRAPH 2025
Vancouver+ 10-14 August

```
shflinput = shuffle(input);
coopVecMatMul(temp0, shflinput, weightBuffer, offset0);
layer0 = unshuffle(temp0);

layer0 = max(layer0, 0);

shfllayer0 = shuffle(layer0);
coopVecMatMul(temp1, shfllayer0, weightBuffer, offset1);
layer1 = unshuffle(temp1);

layer1 = max(layer1, 0);

shfllayer1 = shuffle(layer1);
coopVecMatMul(temp2, shfllayer1, weightBuffer, offset2);
output = unshuffle(temp2);
```

REMOVE UNSHUFFLE / SHUFFLE PAIRS

```
shflinput = shuffle(input);
coopVecMatMul(temp0, shflinput, weightBuff, offset0);
temp0_1 = max(temp0, 0);

coopVecMatMul(temp1, temp0_1, weightBuff, offset1);
temp1_1 = max(temp1, 0);

coopVecMatMul(temp2, temp1_1, weightBuff, offset1);
output = unshuffle(temp2);
```

The shuffle removal is called "Layer fusion"

Sometimes it fails

MAPPING AN MLP ONTO TENSOR CORES 3/3

SIGGRAPH 2025
Vancouver+ 10-14 August

```
shflinput = shuffle(input);
coopVecMatMul(shfllayer0, shflinput, weightBuff, offset0);
temp0_1 = max(temp0, 0);

coopVecMatMul(temp1, temp0_1, weightBuff, offset1);
temp1_1 = max(temp1, 0);

coopVecMatMul(temp2, temp1_1, weightBuff, offset1);
output = unshuffle(temp2);
```

PEEL THE DIVERGENT MATRICES

```
shflinput = shuffle(input);

foreach (unique combination of offsets)
{
    coopVecMatMul(temp0, shflinput, weightBuff, offset0);
    temp0_1 = max(temp0, 0);

    coopVecMatMul(temp1, temp0_1, weightBuff, offset1);
    temp1_1 = max(temp1, 0);

    coopVecMatMul(temp2, temp1_1, weightBuff, offset2);

    mergeThreadResults(temp2);
}

output = unshuffle(temp2);
```

DIAGNOSING LAYER FUSION FAILURES

- No publicly available diagnostic tool at this time
- Experimental approach:
  - Measure performance of the original code
  - Remove all code between coopVecMatMul[Add] operations
  - See if performance increases dramatically
  - Re-introduce pieces of that code looking for perf cliffs

Nsight Graphics is a great tool,

but it doesn’t fully support CoopVec yet

PREVENTING LAYER FUSION FAILURES 1/3

SIGGRAPH 2025
Vancouver+ 10-14 August

Avoid elementwise operations on CoopVec's between layers

BAD

```
coopVecMatMul(layer0, input, weightBuffer, offset0);

for (int i = 0; i < 64; ++i)
{
    // Leaky ReLU
    if (layer0[i] < 0)
        layer0[i] *= 0.01;
}

coopVecMatMul(layer1, layer0, weightBuffer, offset1);
```

GOOD

```
coopVecMatMul(layer0, input, weightBuffer, offset0);

// Use vector operations to express the same math
layer0 = max(layer0, layer0 * 0.01);

coopVecMatMul(layer1, layer0, weightBuffer, offset1);
```

Useful vector intrinsics: min, max, clamp, **step**

Look in "hlsl.meta.slang" for more

PREVENTING LAYER FUSION FAILURES 2/3

SIGGRAPH 2025
Vancouver+ 10-14 August

Use vector load and store operations instead of elementwise ones

BAD

```
coopVecMatMul(layer0, input, weightBuffer, offset0);

CoopVec<half, 64> bias;
for (int i = 0; i < 64; ++i)
{
    bias[i] = biasBuffer.Load<half>(biasOffset0
        + i * sizeof(half));
}

layer0 += bias;

layer0 = max(layer0, 0); // ReLU

coopVecMatMul(layer1, layer0, weightBuffer, offset1);
```

BETTER

```
coopVecMatMul(layer0, input, weightBuffer, offset0);

let bias = CoopVec<half, 64>.load(biasBuffer, biasOffset0);
layer0 += bias;

layer0 = max(layer0, 0); // ReLU

coopVecMatMul(layer1, layer0, weightBuffer, offset1);
```

PREVENTING LAYER FUSION FAILURES 3/3

SIGGRAPH 2025
Vancouver+ 10-14 August

Use vector load and store operations instead of elementwise ones

BETTER

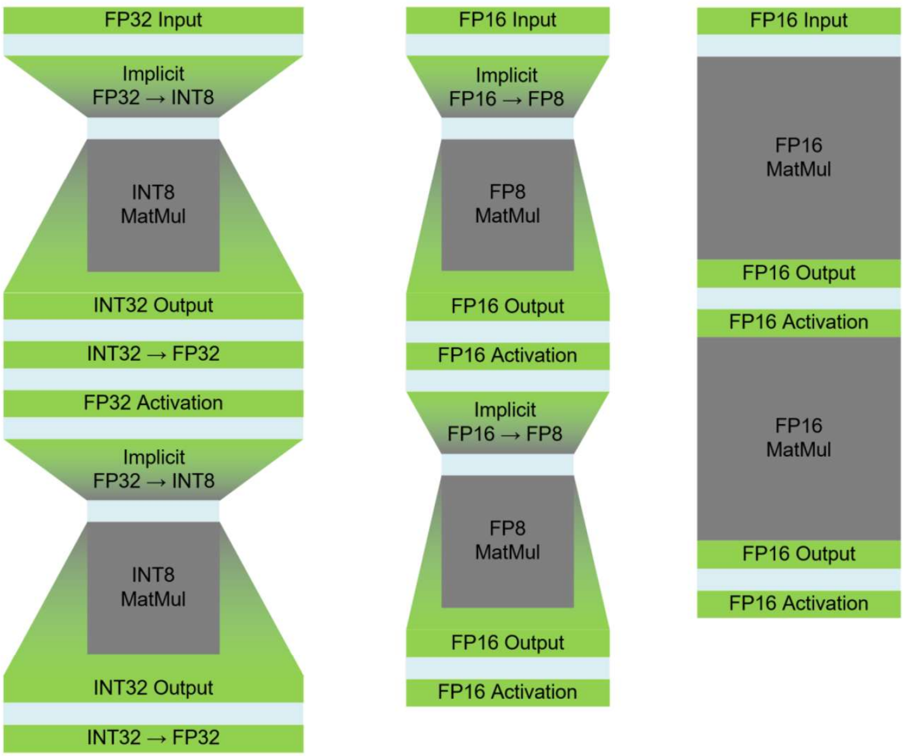
```
coopVecMatMul(layer0, input, weightBuffer, offset0);

let bias = CoopVec<half, 64>.load(biasBuffer, biasOffset0);
layer0 += bias;

layer0 = max(layer0, 0); // ReLU

coopVecMatMul(layer1, layer0, weightBuffer, offset1);
```

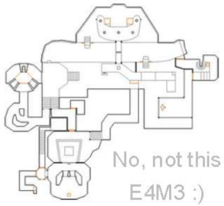
GOOD

```
coopVecMatMulAdd(layer0, input, weightBuffer, offset0,
    biasBuffer, biasOffset0);

layer0 = max(layer0, 0); // ReLU

coopVecMatMul(layer1, layer0, weightBuffer, offset1);
```

MATH PRECISION CONSIDERATIONS

SIGGRAPH 2025
Vancouver+ 10-14 August



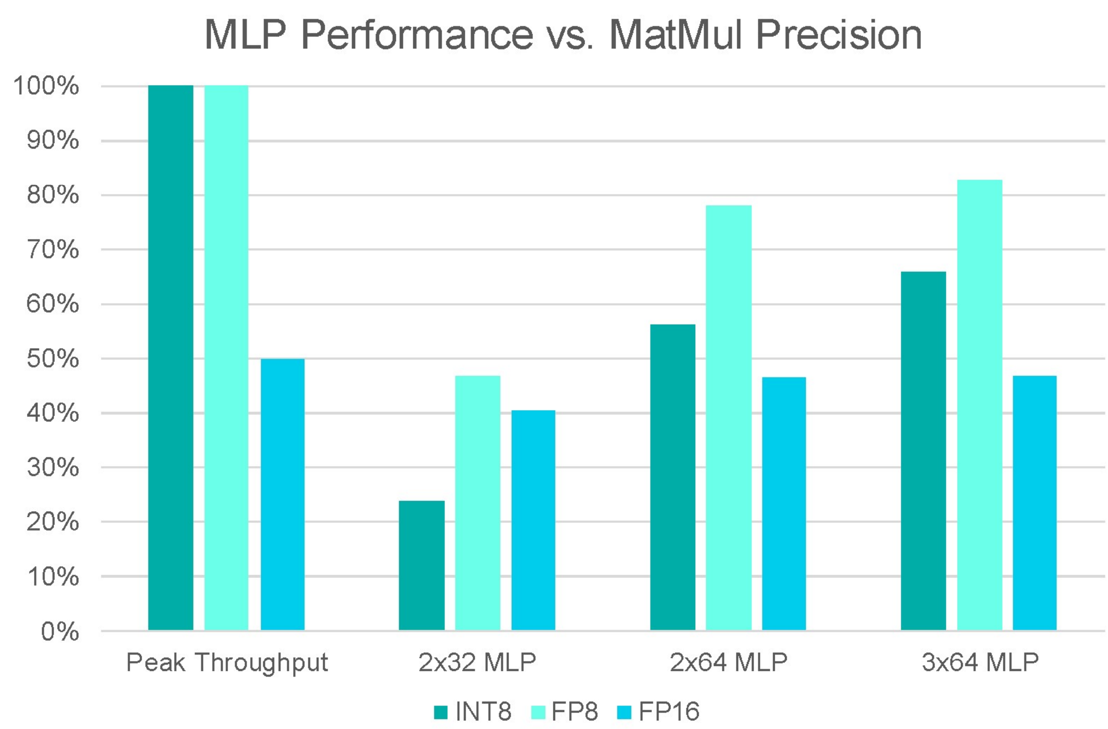


No, not this E4M3 :)

- INT8 – Most difficult to use
  - Extra scale operations needed for each layer
  - INT->FP conversions are not cheap
  - 32-bit per element vectors take many registers
- FP8 (E4M3 and E5M2)
  - No scaling or conversions needed, small register footprint
  - Very low precision, mostly suitable for hidden layers
- FP16 – Easiest to use
  - Lower peak throughput than FP8 or INT8
  - Enough precision for all practical inference needs

MATH PRECISION COMPARISON

SIGGRAPH 2025
Vancouver+ 10-14 August



PERFORMANCE TIPS - SUMMARY

- Convert matrices to optimal layouts offline
- Minimize matrix divergence
- Avoid elementwise access to vectors between network layers
- Watch out for layer fusion failures

Math Precision Tradeoffs

	INT8	FP8	FP16
Precision	✅	❌	✅✅
Throughput	✅	✅✅	❌
Register Pressure	❌	✅	✅
Ease of Use	❌	✅	✅✅
GPU Compatibility	✅✅	✅	✅✅

THE TRANSITION STORY

Training + Inference

\+ Slang™

SlangPy

Vulkan.
DirectX® 12

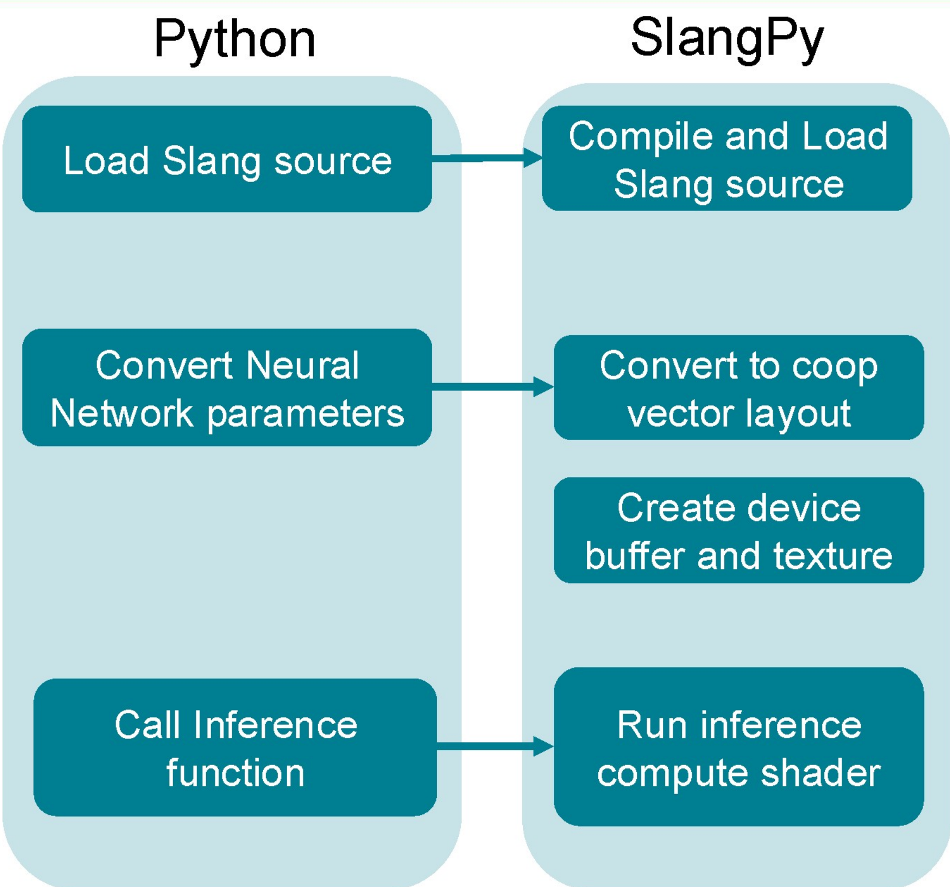
Inference

\+ Slang™

Vulkan.
DirectX® 12

PYTHON IMPLEMENTATION

SIGGRAPH 2025
Vancouver+ 10-14 August




```python
import slangpy as spy
module = spy.Module.load_from_file(app.device, "inference.slang")
app = App(width=512*3+10*2, height=512, title="Mipmap Example",
          device_type=spy.DeviceType.vulkan)
```

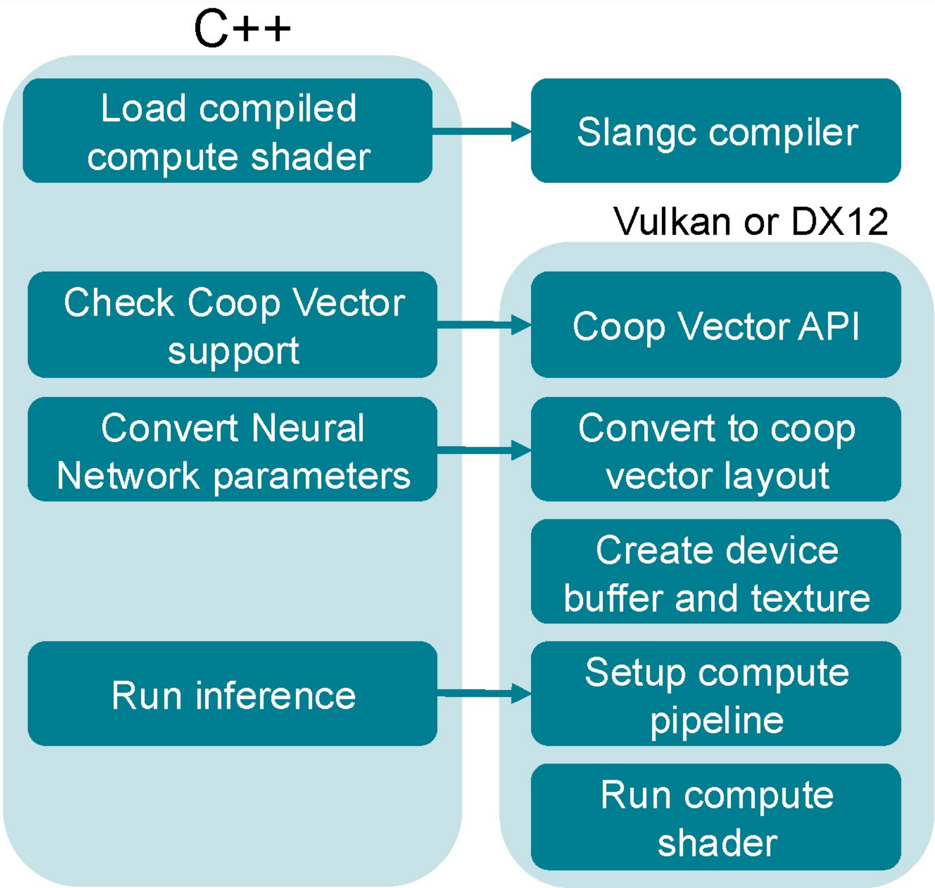
```python
# Load values of biases and weights
weights_np = np.array(data['weights'], dtype=np.float16).reshape((outputs, inputs))
biases_np = np.array(data['biases'], dtype=np.float16)

# Convert weights into coopvec layout
desc = app.device.coopvec_create_matrix_desc(self.outputs, self.inputs,
                                             self.layout,
                                             spy.DataType.float16, 0)
weight_count = desc.size // 2 # sizeof(half)
params_np = np.zeros((weight_count, ), dtype=np.float16)
app.device.coopvec_convert_matrix_host(weights_np, params_np,
                                       dst_layout=self.layout)
```

```python
self.biases = app.device.create_buffer(struct_size=2, element_count=self.outputs,
                                       data=biases_np)
self.weights = app.device.create_buffer(struct_size=2, element_count=weight_count,
                                        data=params_np)
```

```python
lr_output = spy.Tensor.empty_like(image)
module.inference(pixel = spy.call_id(),
                 resolution = res,
                 network = network,
                 _result = lr_output)
```

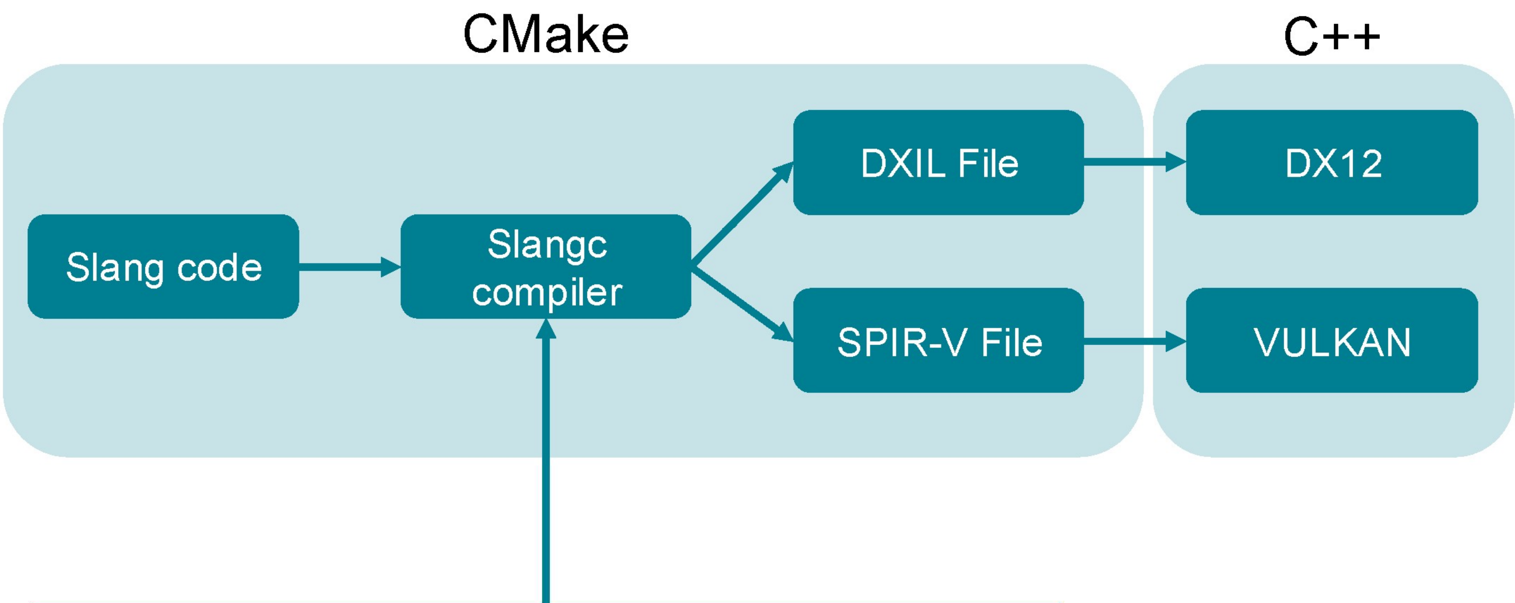
C++ IMPLEMENTATION

SIGGRAPH 2025
Vancouver+ 10-14 August

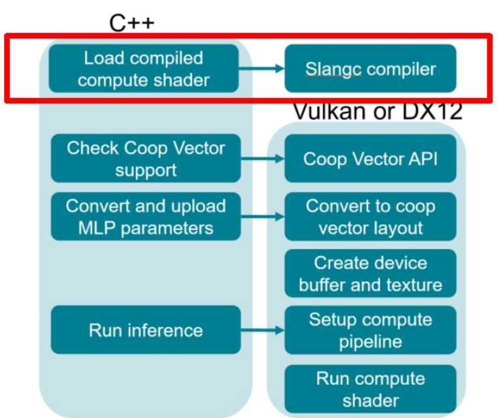


STEP 1 – SLANG AS SHADER MODULE

SIGGRAPH 2025
Vancouver+ 10-14 August

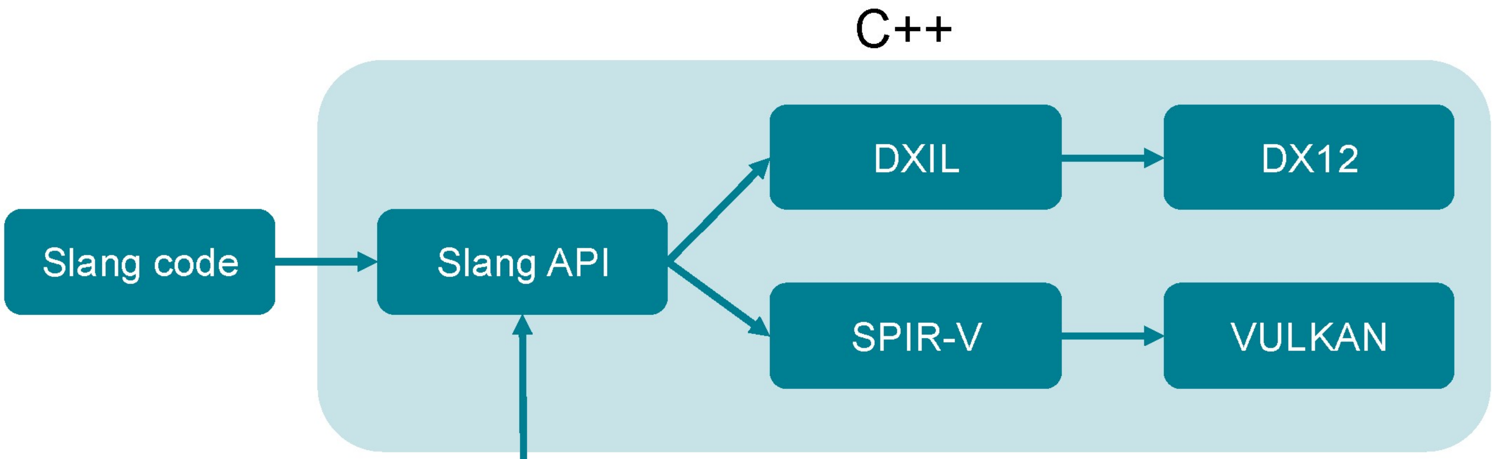



```
-capability spvCooperativeVectorNV
-capability pvCooperativeVectorTrainingNV
```

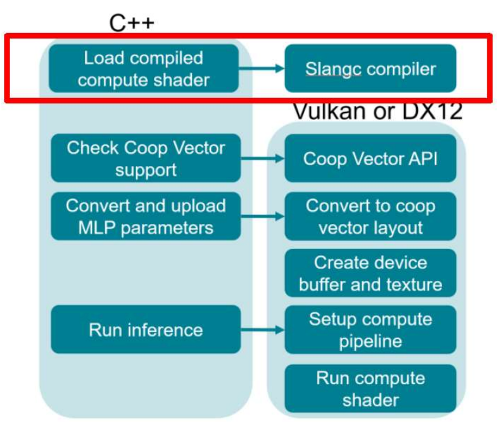


STEP 1 – SLANG AS SHADER MODULE

SIGGRAPH 2025
Vancouver+ 10-14 August



- Cooperative Vector parameters
  - -capability spvCooperativeVectorNV
  - -capability pvCooperativeVectorTrainingNV
- Compile time parameters
- Template parameters
- Reflection



STEP 2 - FEATURE DETECTION FOR COOPERATIVE VECTORS (VULKAN)

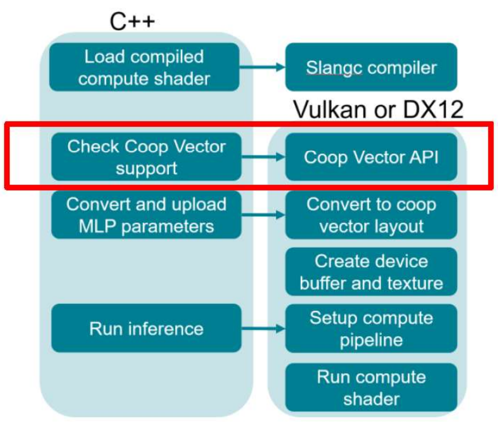
SIGGRAPH 2025
Vancouver+ 10-14 August

Required Vulkan device extensions:

`VK_NV_cooperative_vector`

Required Vulkan physical device features:

```
VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_COOPERATIVE_VECTOR_FEATURES_NV
cooperativeVector = true
cooperativeVectorTraining = true
```



STEP 2 - FEATURE DETECTION FOR COOPERATIVE VECTORS (DX12)

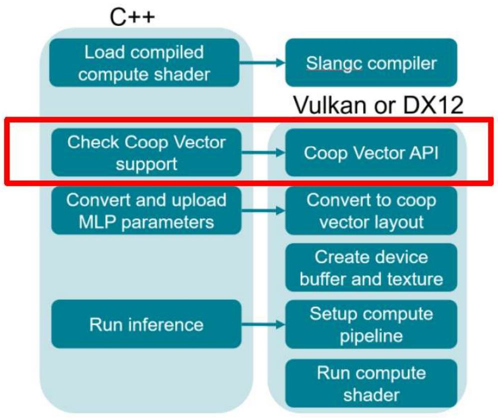
SIGGRAPH 2025
Vancouver+ 10-14 August

Call D3D12EnableExperimentalFeatures to enable following features:

- D3D12ExperimentalShaderModels
- D3D12CooperativeVectorExperiment

Call D3D12 device method CheckFeatureSupport for D3D12_FEATURE_D3D12_OPTIONS_EXPERIMENTAL and check:

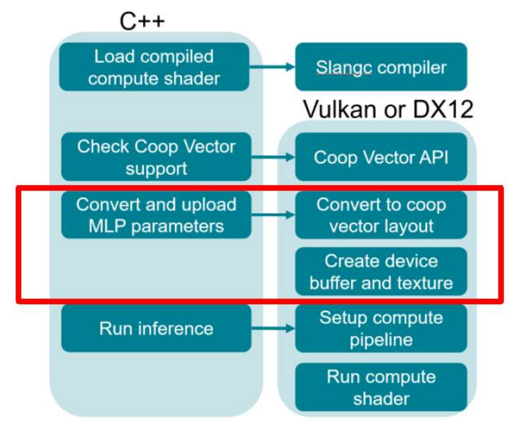
- D3D12_COOPERATIVE_VECTOR_TIER_1_0 for inference support
- D3D12_COOPERATIVE_VECTOR_TIER_1_1 for training support



STEP 3 – MATRIX OPTIMAL LAYOUTS

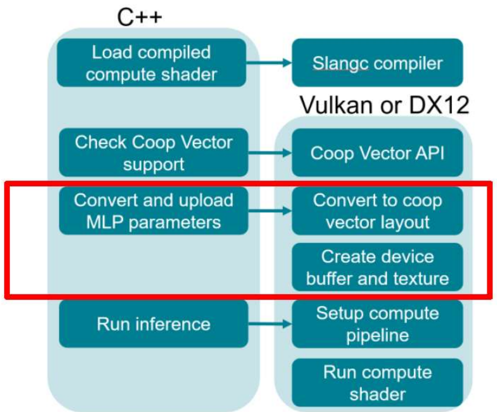
FEED-FORWARD LAYER USING COOPERATIVE VECTORS

```
struct NetworkParameters<int Inputs, int Outputs>
{
    ByteAddressBuffer weights, biases;

    CoopVec<half, Outputs> forward(CoopVec<half, Inputs> x)
    {
        return coopVecMatMulAdd<half, Outputs>(
            x, CoopVecComponentType.Float16,
            weights, 0, CoopVecComponentType.Float16,
            biases, 0, CoopVecComponentType.Float16,
            CoopVecMatrixLayout.InferencingOptimal,
            false,
            0
        );
    }
}
```



- Tensor cores use custom matrix layouts
  - Components of matrices are shuffled between threads
  - Specific layout depends on the GPU and data types

STEP 3 – MATRIX LAYOUT CONVERSION



1. Create parameters buffer in optimal layout
   - Query shuffled matrix size from VK/DX12
   - Upload row- or column-major matrix to GPU
   - Convert layout (perform the shuffle) on the GPU
     - Inference Optimal
     - Training Optimal

Vulkan

- `vkConvertCooperativeVectorMatrixNV`
  - CPU conversion / size query
- `vkCmdConvertCooperativeVectorMatrixNV`
  - GPU conversion

DX12

- `GetLinearAlgebraMatrixConversionDestinationInfo`
  - Size query
- `ConvertLinearAlgebraMatrix`
  - GPU conversion

2. Create constant buffer

3. Create output texture

STEP 4 – CONVERT INFERENCE FUNCTION TO SHADER

SLANG

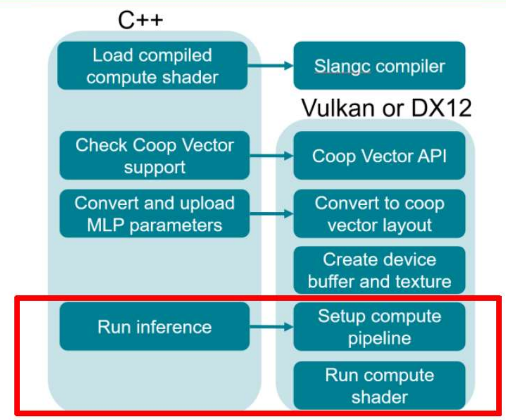
```
struct NetworkParameters<int Inputs, int Outputs> {
    ByteAddressBuffer<half> weights, biases;

    CoopVec<half, Outputs> forward(CoopVec<half, Inputs> x);
}
```

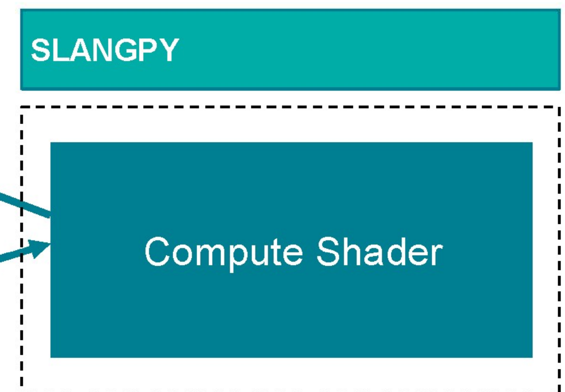
```
struct Network {
    NetworkParameters<16, 32> layer0, layer1, layer2;

    float3 eval(no_diff float2 uv);
}
```

```
float3 inference(int2 pixel, int2 resolution, Network network)
{
    float2 uv = (float2(pixel) + 0.5f) / float2(resolution);
    return network.eval(uv);
}
```

C++

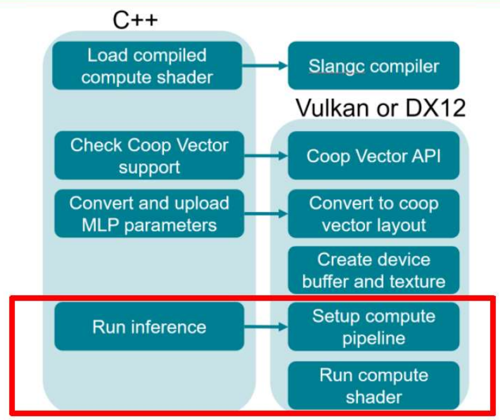
```
lr_output = spy.Tensor.empty_like(image)

module.inference(pixel = spy.call_id(), resolution = res, network = network, _result = lr_output)
```





STEP 4 – CONVERT INFERENCE FUNCTION TO SHADER

SIGGRAPH 2025
Vancouver+ 10-14 August



SLANG

```
struct NetworkParameters<int Inputs, int Outputs> {
    ByteAddressBuffer <half> weights, biases;

    CoopVec<half, Outputs> forward(CoopVec<half, Inputs> x);
}

struct Network {
    NetworkParameters<16, 32> layer0;
    NetworkParameters<32, 32> layer1;
    NetworkParameters<32, 3>  layer2;

    float3 eval(no_diff float2 uv);
}

float3 inference(int2 pixel, int2 resolution, Network network)
{
    float2 uv = (float2(pixel) + 0.5f) / float2(resolution);
    return network.eval(uv);
}
```

```
Network network;

ConstantBuffer<NeuralConstants> gConst;
RWTexture2D<float4> outputTexture;

[shader("compute")]
[numthreads(8, 8, 1)]
void main_cs(uint3 pixel : SV_DispatchThreadID)
{
    outputTexture[pixel.xy] = inference(pixel.xy,
                                        gConst.resolution,
                                        network);
}
```

STEP 4 – CONVERT INFERENCE FUNCTION TO SHADER

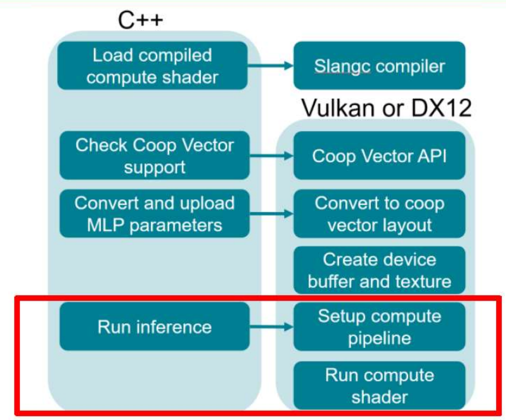
SIGGRAPH 2025
Vancouver+ 10-14 August

SLANG

```
struct NetworkParameters<int Inputs, int Outputs> {
    ByteAddressBuffer <half> weights, biases;
    CoopVec<half, Outputs> forward(CoopVec<half, Inputs> x);
}

struct Network {
    NetworkParameters<16, 32> layer0;
    NetworkParameters<32, 32> layer1;
    NetworkParameters<32, 3>  layer2;

    float3 eval(no_diff float2 uv);
}

Network network;

ConstantBuffer<NeuralConstants> gConst;
RWTexture2D<float4> outputTexture;

[shader("compute")]
[numthreads(8, 8, 1)]
void main_cs(uint3 pixel : SV_DispatchThreadID){
 // Compute shader
}
```

C++

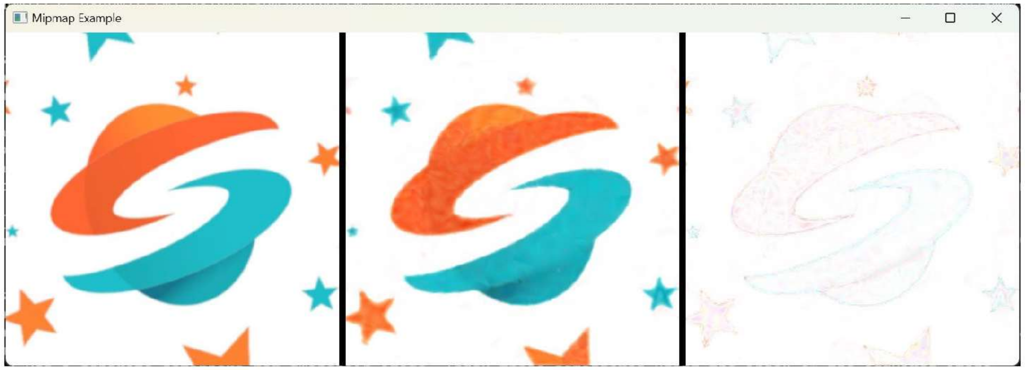
```
// Initialize parameters buffers

// Setup up constant buffer

// Bind output texture

// Dispatch compute shader
nvrhi::ComputeState state;
m_CommandList->setComputeState(state);
m_CommandList->dispatch(textureWidth, textureHeight, 1);
```



YOUR NEURAL SHADER IS NOW SHIPPED!

SIGGRAPH 2025
Vancouver+ 10-14 August

Python

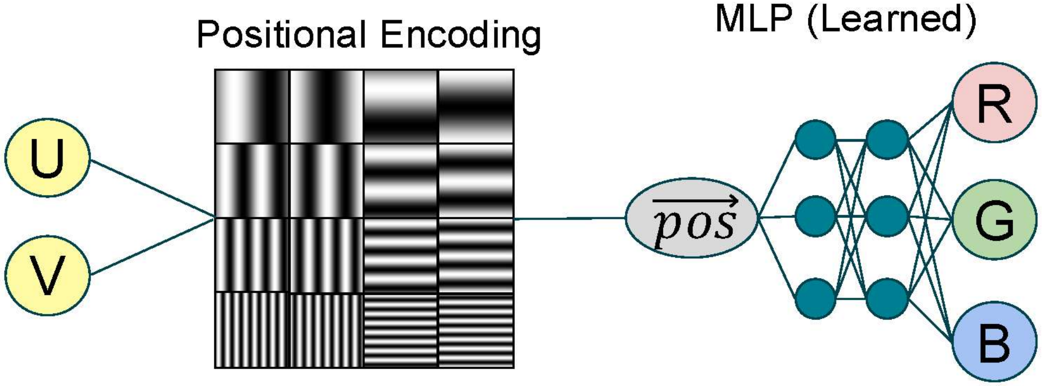


C++

SIGGRAPH 2025
Vancouver+ 10-14 August

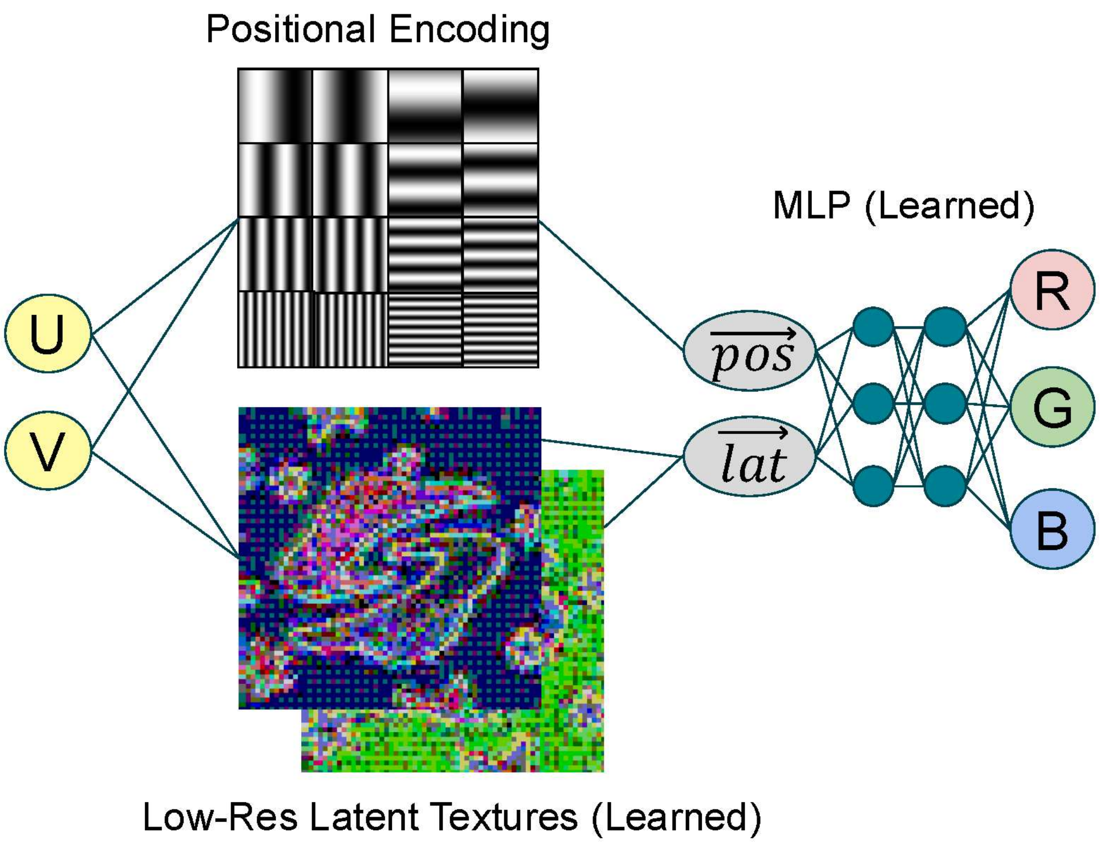
NEURAL TEXTURE COMPRESSION

FLASHBACK: PREVIOUS BEST RESULT



Output Image (meh)

ADDING LATENT TEXTURES

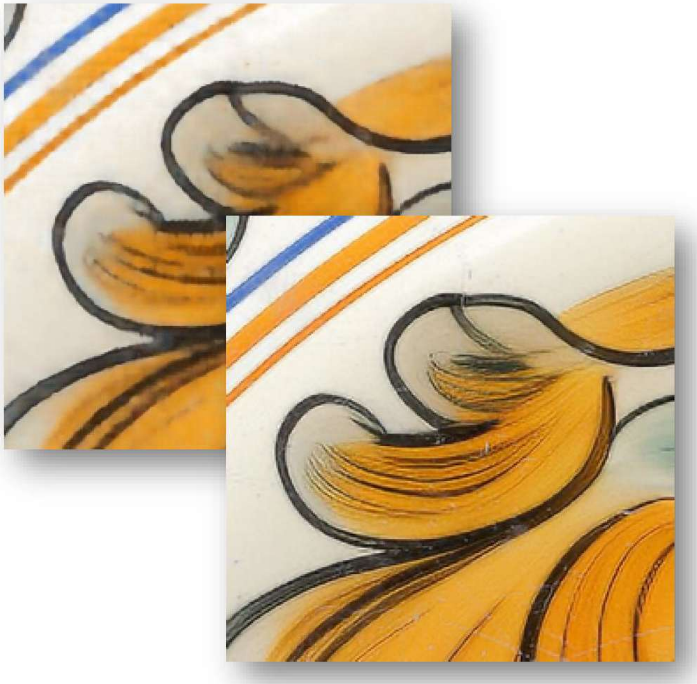


Output Image (pretty good)

KEY FACTS ABOUT NTC

SIGGRAPH 2025
Vancouver+ 10-14 August

- Wide range of latent texture parameters
  - Independent resolution, channel count, bits per feature controls
  - Overall practical range of 0.5 — 20 bits per pixel storing up to 16 channels
  - MLP weights are tiny compared to the latents (~12 KB vs. many MB)
- Each texel can be decoded independently
  - Suitable for use as a replacement for regular material textures
- Neural compression without hallucinations
  - Compression is training of the network and latent textures
  - Network is small and overfitted to reproduce only one material
- Original paper describing NTC and STF:
  - K. Vaidyanathan et al. “Random-Access Neural Compression of Material Textures”

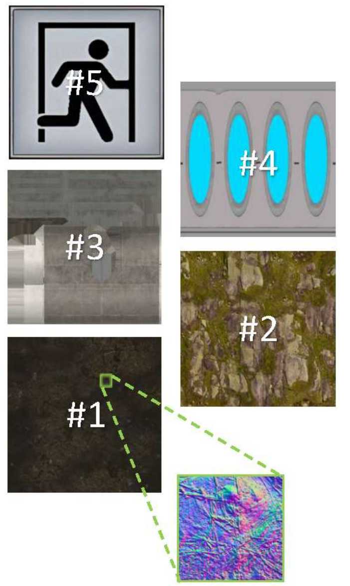


Crops from an NTC compressed texture at 0.5 and 20.0 bpp

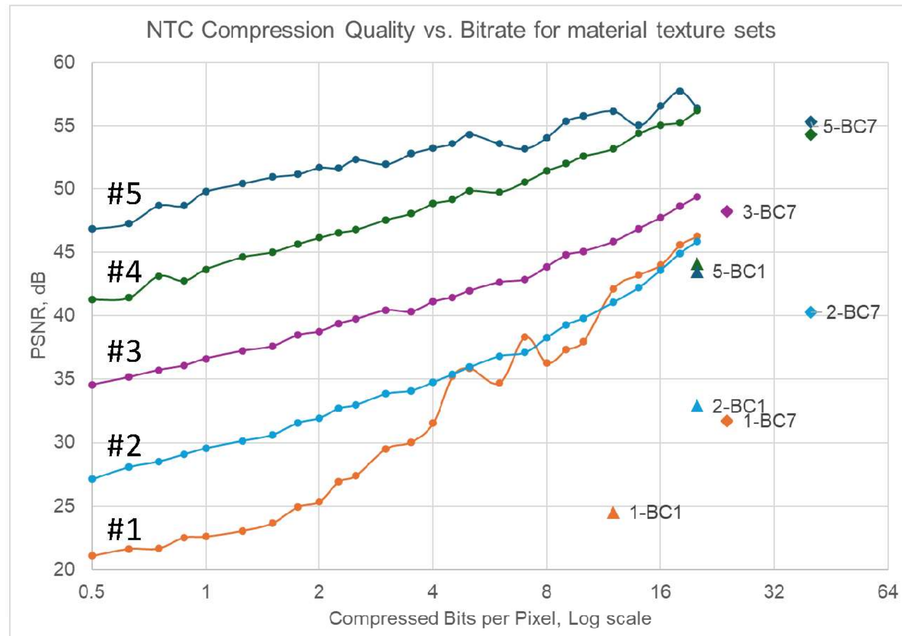
NEURAL TEXTURE COMPRESSION QUALITY

SIGGRAPH 2025
Vancouver+ 10-14 August

- Consistently better than BCn
  - More benefit for complex materials
  - Up to 8x better at similar visual quality
  - Introduces blur, not blockiness
- Mostly monotonic vs. BPP
  - More information means higher quality
  - Training finds effective ways to use the bits
- Benefits from correlated channels
  - All textures for a PBR material are better compressed together than separately



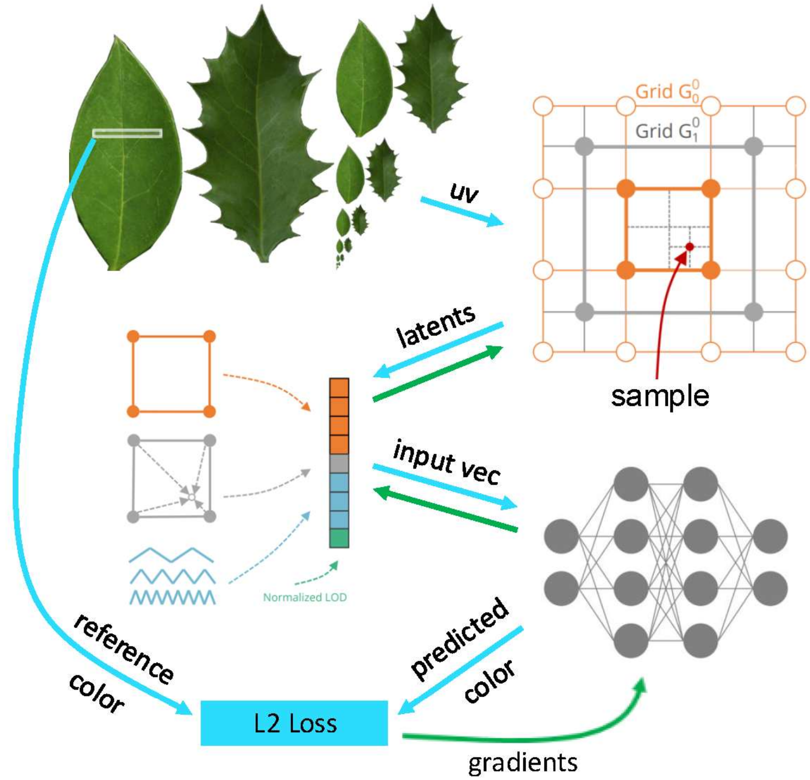
High detail normal map is hard to compress



NTC TRAINING PROCESS

SIGGRAPH 2025
Vancouver+ 10-14 August

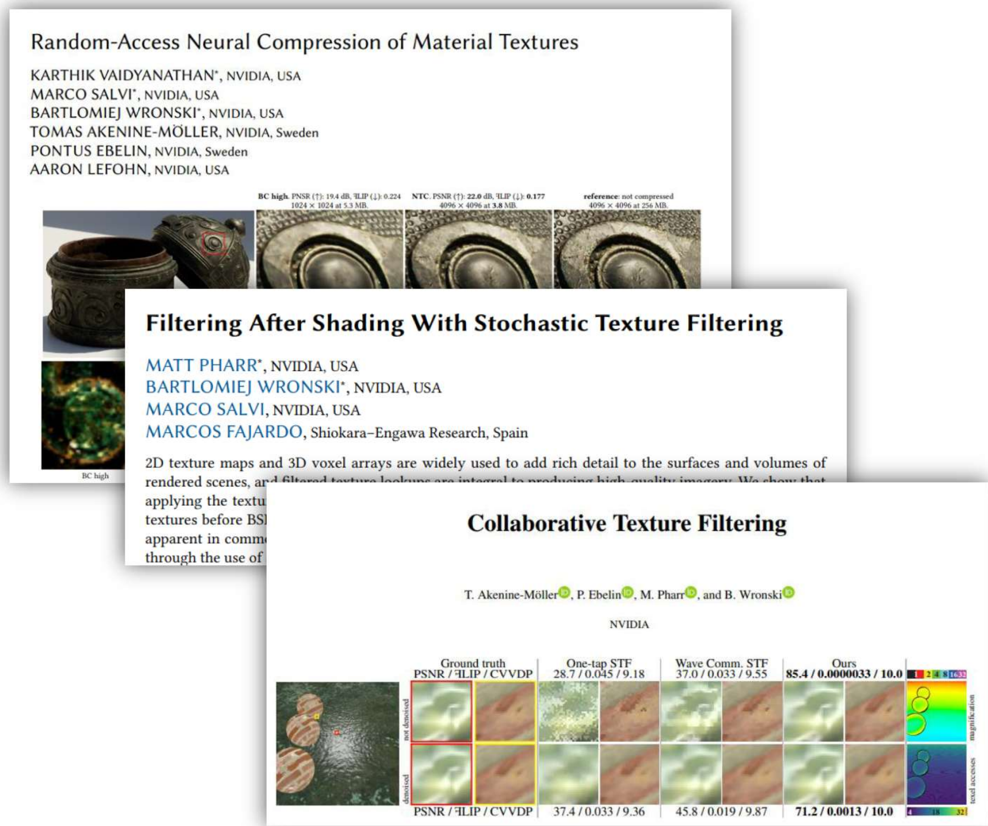
- Pick a set of pixels to process on this step
  - 64k pixels per step is usually good
- For each pixel in parallel:
  - Calculate positional encoding based on UV
  - Sample the latent textures at UV
  - Prepare MLP input vector
  - Evaluate MLP
  - Load reference texture channels
  - Compute gradient ~= (predicted – reference)
  - Backpropagate gradients
  - Accumulate gradients for MLP and features
- Run an optimizer step
  - Add the quantization noise
- Repeat 100-200k times



NTC MODES OF OPERATION 1/3

INFERENCE ON SAMPLE

- Replace material texture sampling with NTC decode
- Apply Stochastic Texture Filtering (STF) instead of hardware filtering
  - NTC gives one pixel per decode, direct trilinear or aniso filtering would be too expensive
- Benefits:
  - Minimal VRAM footprint
- Drawbacks:
  - Makes shaders larger and slower (depends…)
  - STF is a requirement

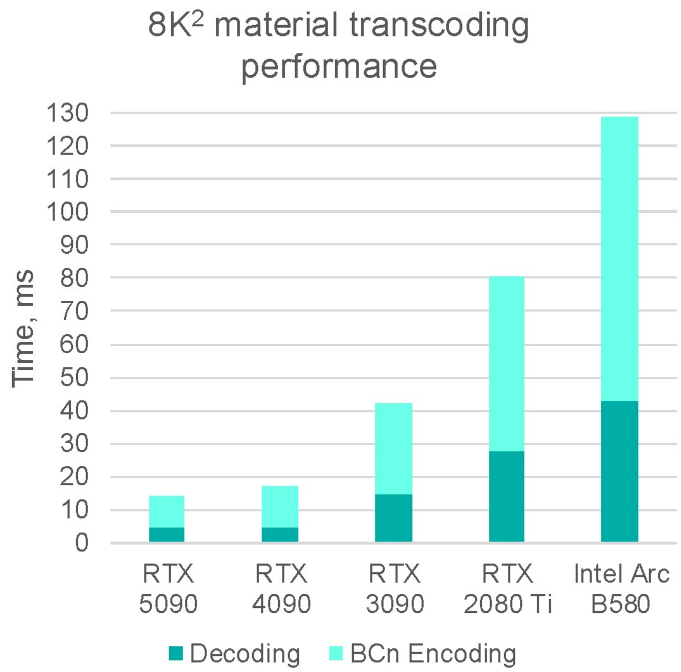


NTC MODES OF OPERATION 2/3

SIGGRAPH 2025
Vancouver+ 10-14 August

INFERENCE ON LOAD

- Transcode NTC materials into BCn at load time
- Sample BCn textures as usual
- Benefits:
  - Easy integration
  - Rendering performance is unaffected
- Drawbacks:
  - No VRAM savings, only disk space / network traffic



Using FP8 inference with CoopVec for NTC

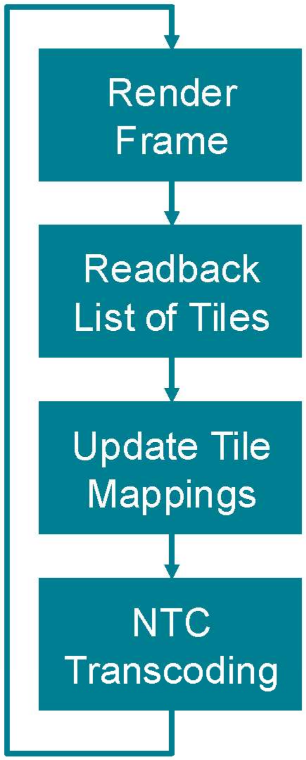
Encoding 2x BC7 and 1x BC5

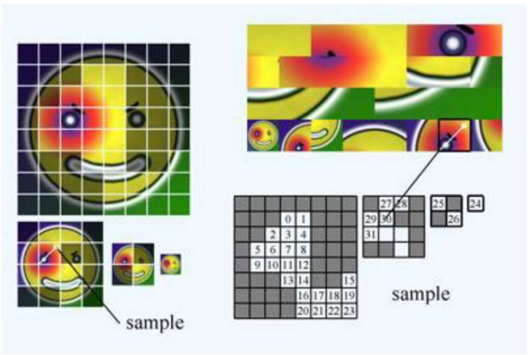
NTC MODES OF OPERATION 3/3

SIGGRAPH 2025
Vancouver+ 10-14 August

INFERENCE ON FEEDBACK

- Extension for a virtual texturing system
- Track which texture MIPs and tiles are needed
- Transcode the necessary tiles to BCn at runtime
- Benefits:
  - Low performance impact on render passes
- Drawbacks:
  - VRAM usage includes both NTC and partial BCn data
  - Potentially uneven frame times





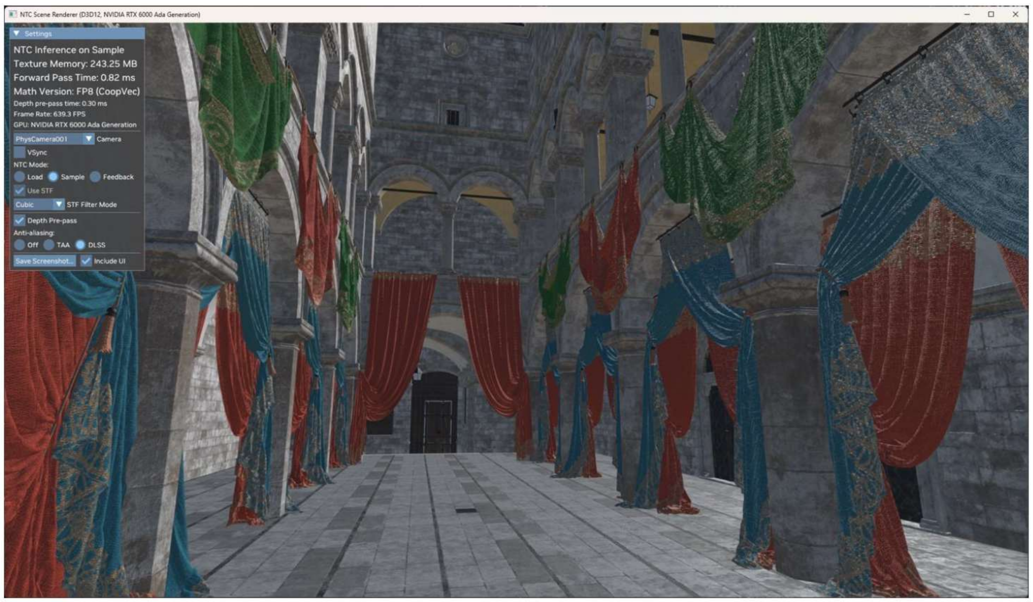
Virtual texture

Image: Sean Barrett, GDC 2008

https://silverspaceship.com/src/svt/

NTC PERFORMANCE COMPARISON

SIGGRAPH 2025
Vancouver+ 10-14 August



Intel Sponza scene in the NTC SDK Renderer

Mode	Render Time	Texture Memory
On Load	0.39 ms	2041 MB
On Sample	0.82 ms	243 MB
On Feedback	0.65 ms	555 MB

Measured on an RTX PRO 6000 Blackwell WE

BENEFITS OF NEURAL TEXTURE COMPRESSION

PRACTICAL

- Saves disk space
- Saves download traffic
- Saves VRAM (maybe)
- Can be used on any platform
  - High-end PC – on-sample
  - Low-end and consoles – on-load or on-feedback
- Can be used now
  - SDK available: github.com / NVIDIA-RTX / RTXNTC

CONCEPTUAL

- Allows using higher detail materials
  - Fewer bits per pixel => more pixels with the same storage
- Can be extended with perceptual loss functions
  - Higher compression ratios with better visual detail
  - Ongoing research direction

MORE NEURAL COMPRESSION TECHNIQUES

SIGGRAPH 2025
Vancouver+ 10-14 August

INSPIRATION: REAL MATERIALS ARE COMPLEX

SIGGRAPH 2025
Vancouver+ 10-14 August

Real-Time Neural Appearance Models

Tizian Zeltner† Fabrice Rousselle† Andrea Weidlich† Petrik Clarberg† Jan Novák†
Benedikt Bitterli† Alex Evans Tomáš Davidovič Simon Kallweit Aaron Lefohn

NVIDIA

†equal contribution, order determined by a rock-paper-scissors tournament.

INSPIRATION: REAL MATERIALS ARE COMPLEX

SIGGRAPH 2025
Vancouver+ 10-14 August

WE CAN RENDER THESE, BUT NOT IN REAL-TIME


Metal teapot handle

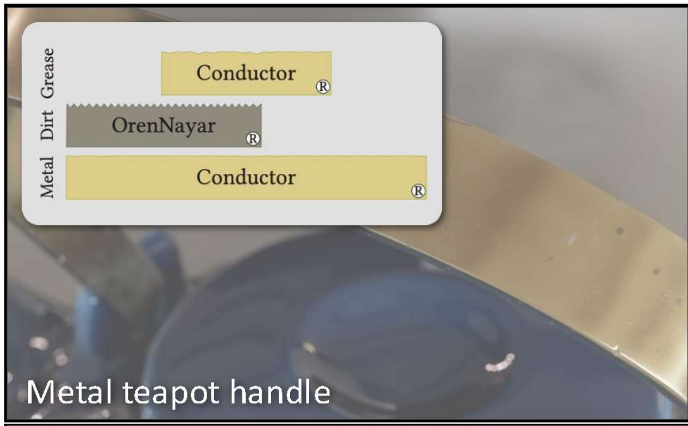

Plastic slicer handle


Blue teapot ceramic

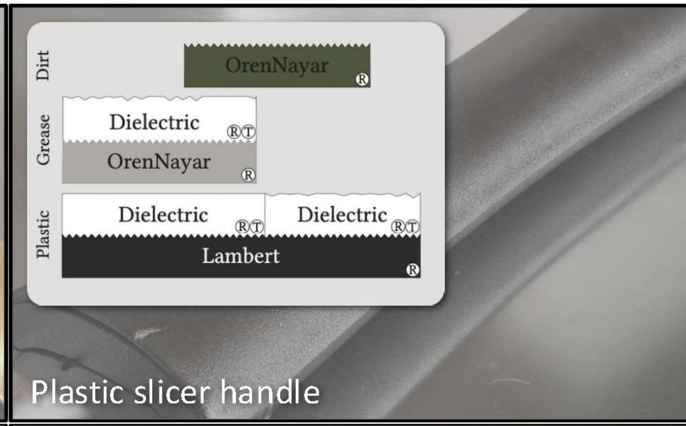

Metal slicer blade


Aged metal inkwell

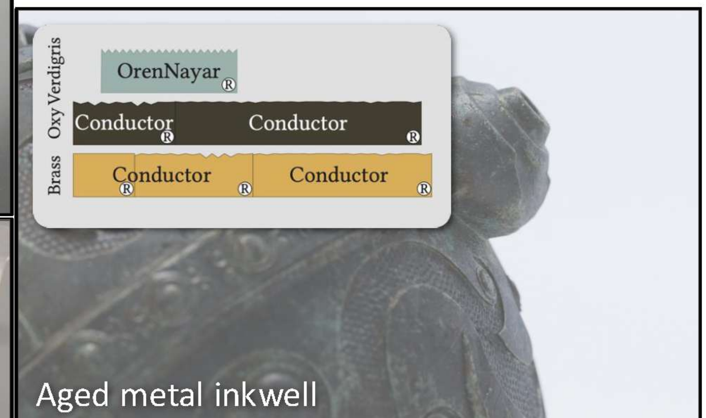
THESE ARE COMPLEX MATERIAL GRAPHS…



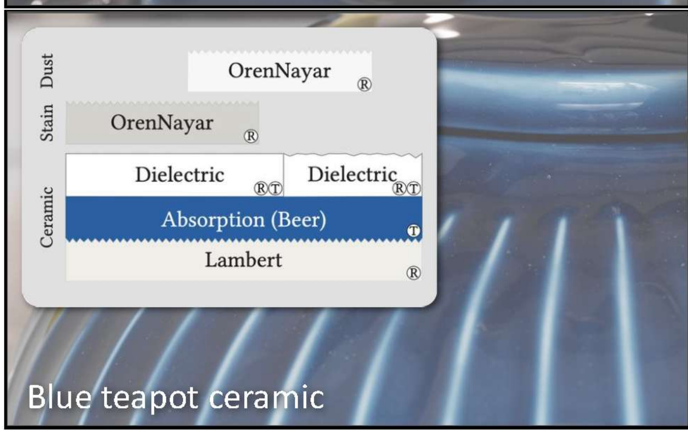
Metal teapot handle



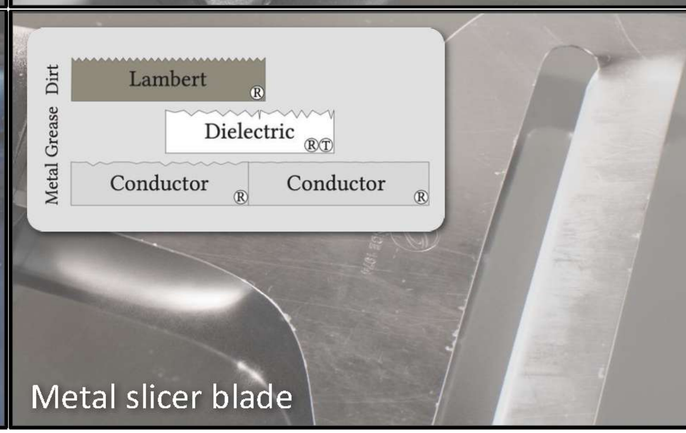
Plastic slicer handle



Aged metal inkwell



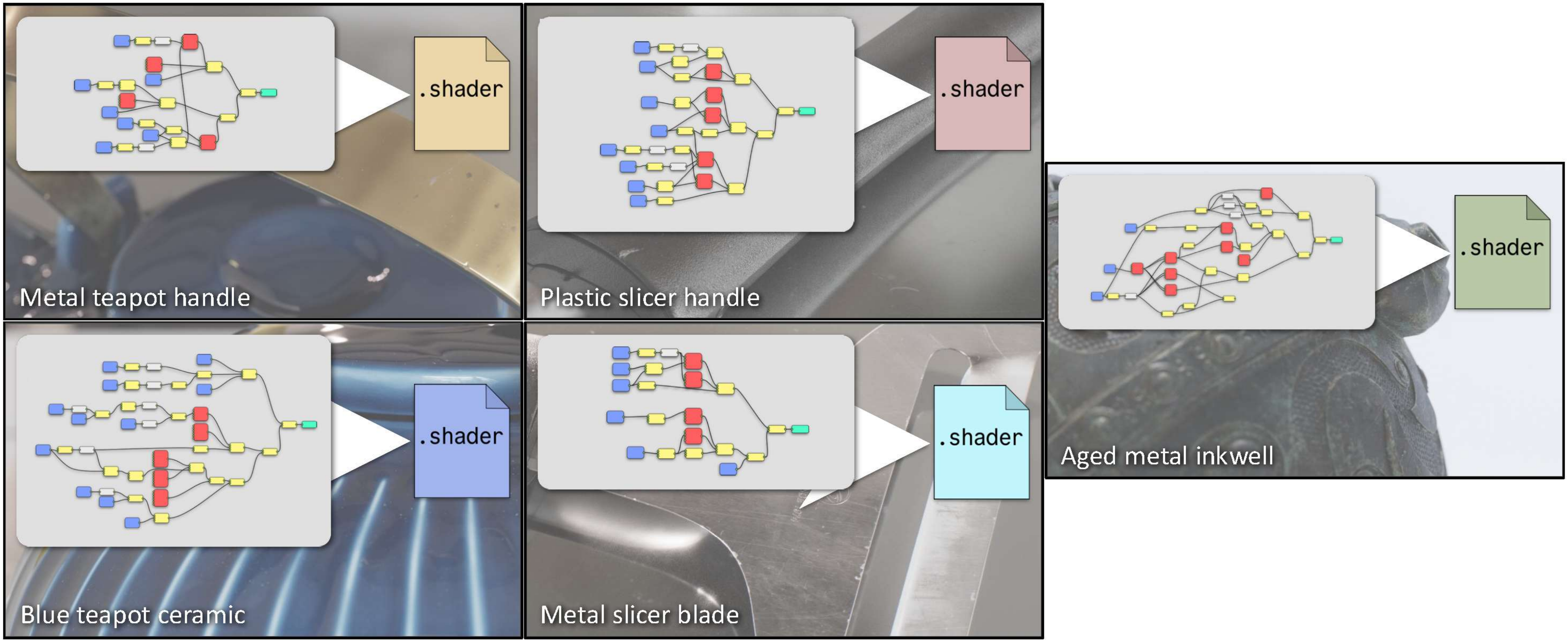
Blue teapot ceramic



Metal slicer blade

…AND WE DON'T KNOW HOW TO SIMPLIFY THEM WELL

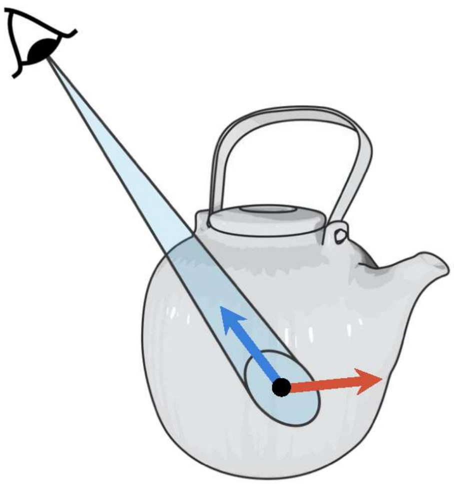
SIGGRAPH 2025
Vancouver+ 10-14 August

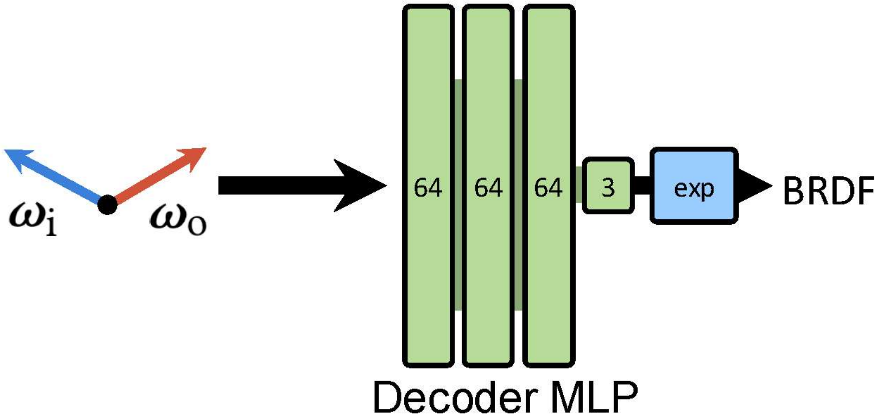


WHAT IF WE USED A NEURAL NETWORK FOR THE TASK?

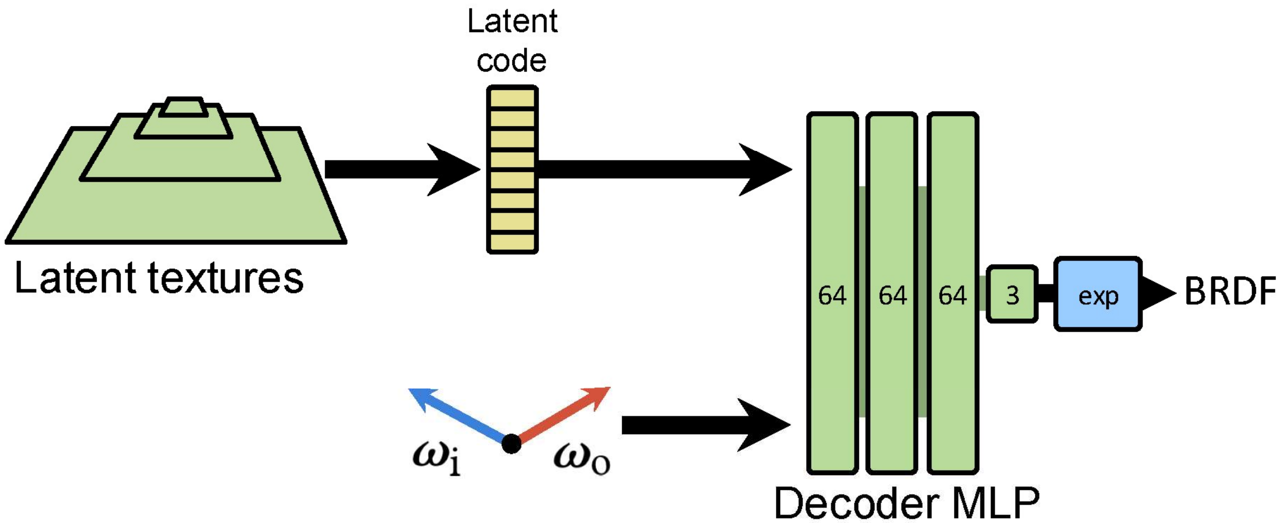
SIMPLEST FIRST: PASS DIRECTIONS TO A NETWORK

SIGGRAPH 2025
Vancouver+ 10-14 August





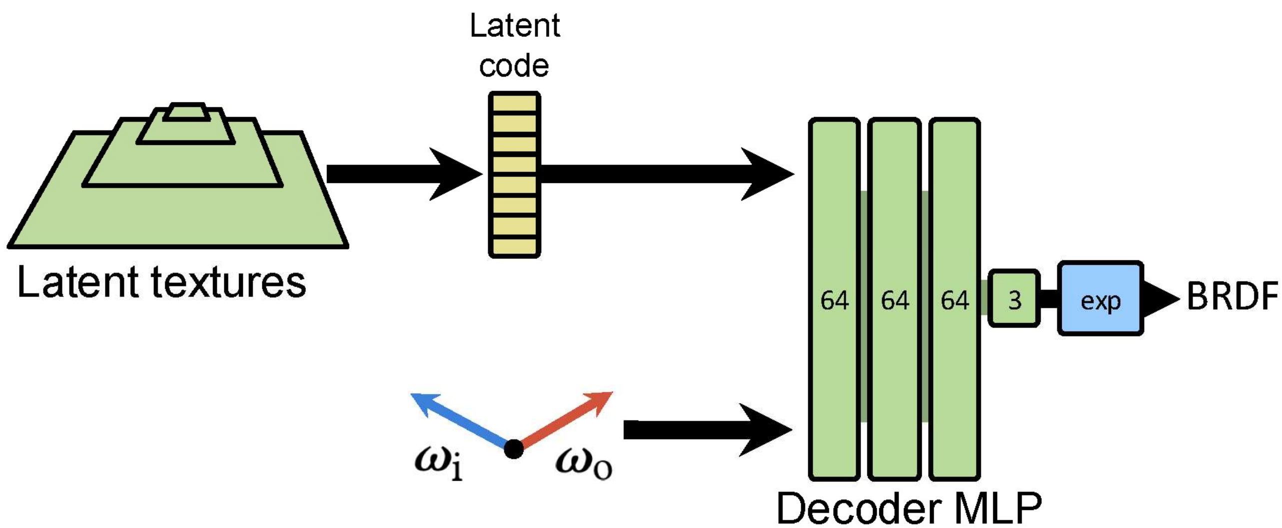
SIGGRAPH 2025
Vancouver+ 10-14 August



THIS WORKS!

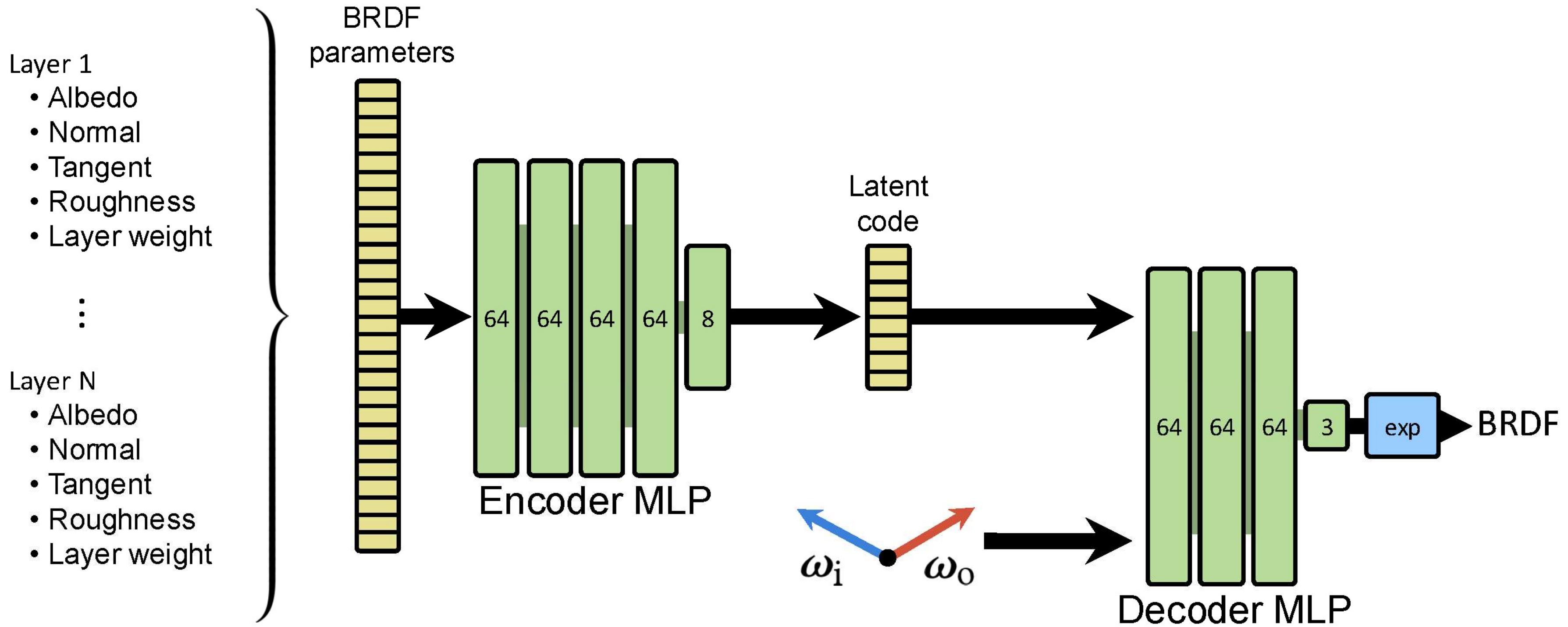
...UP TO A POINT

WE'RE LACKING A "GLOBAL" VIEW



IDEA: LEARN TO TRANSLATE THE ORIGINAL TEXTURES

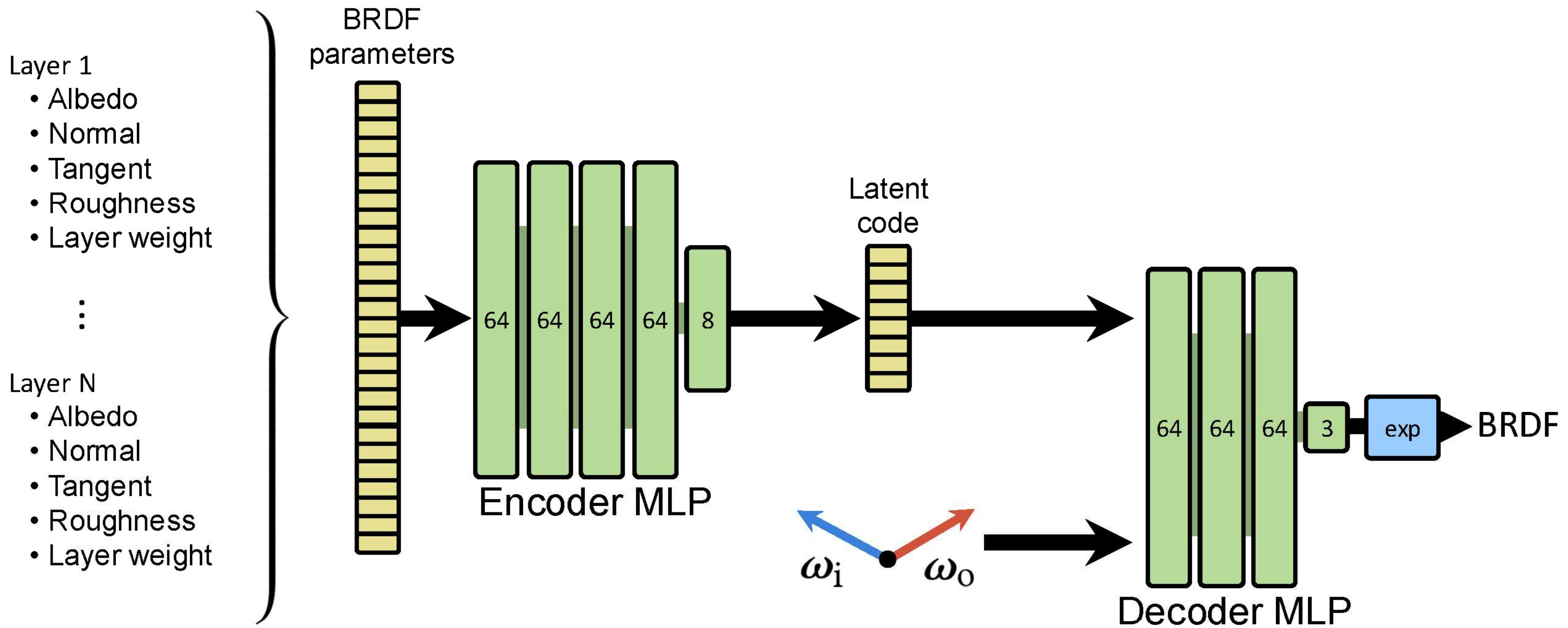
SIGGRAPH 2025
Vancouver+ 10-14 August



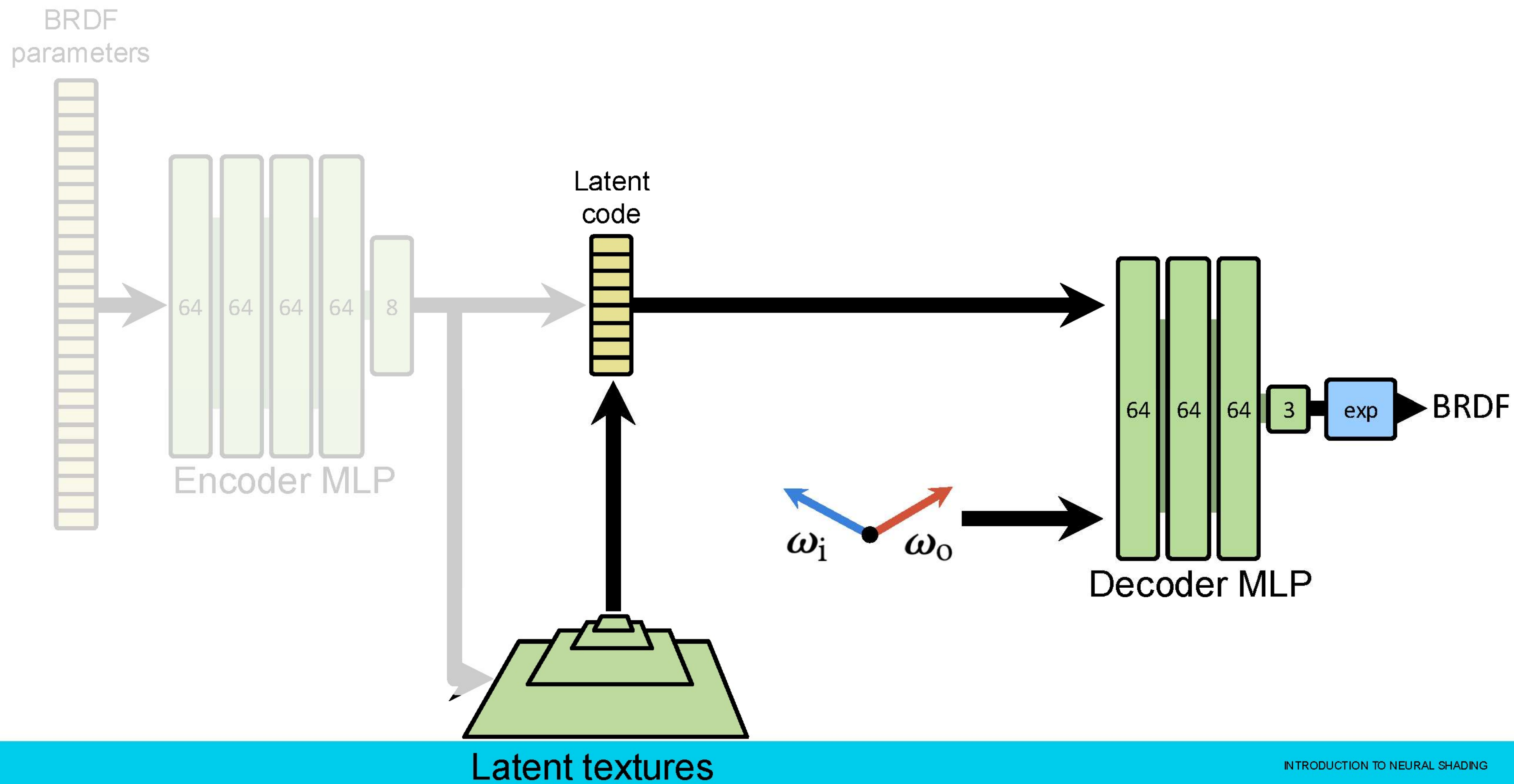
IDEA: LEARN TO TRANSLATE THE ORIGINAL TEXTURES

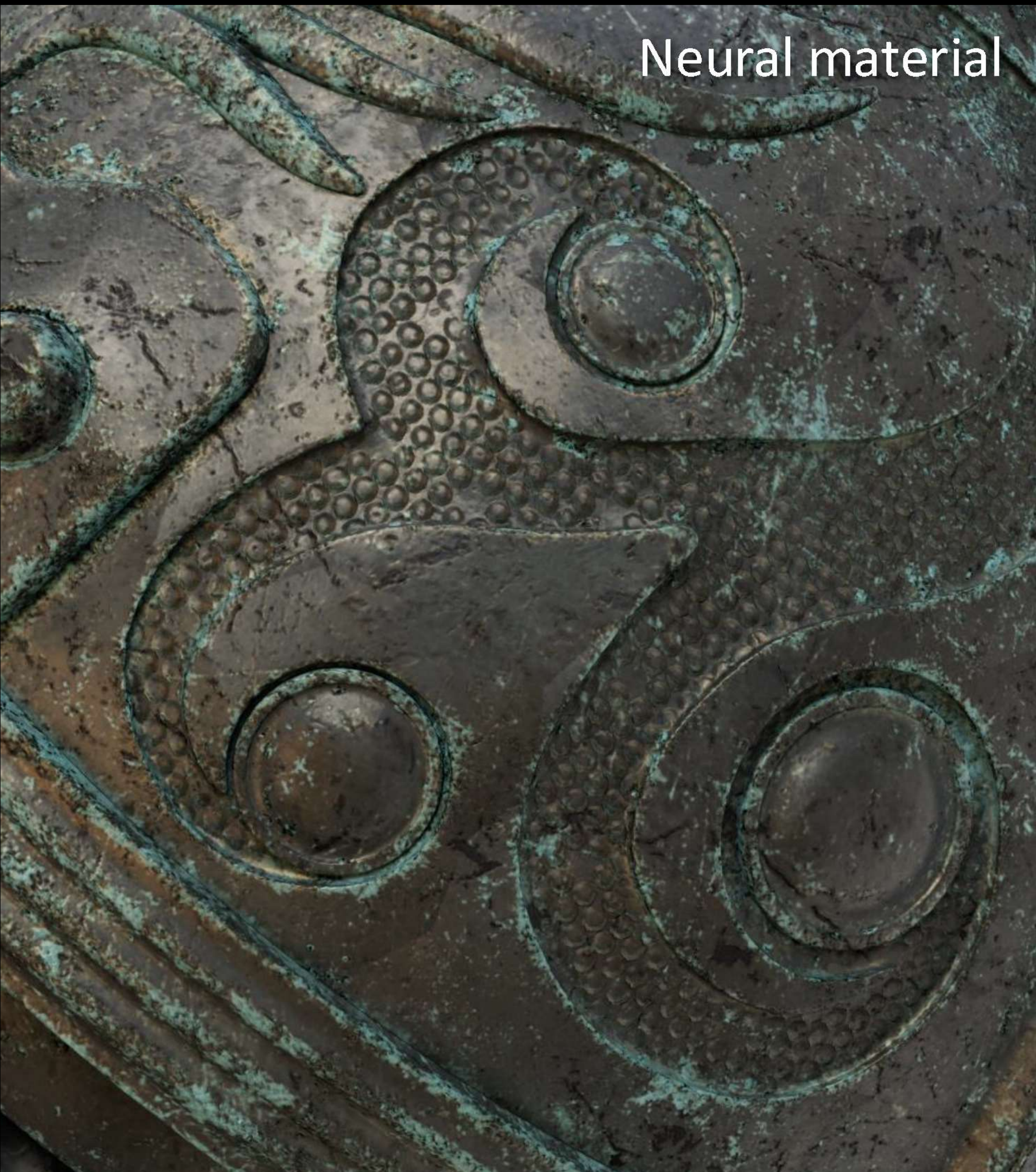
SIGGRAPH 2025
Vancouver+ 10-14 August

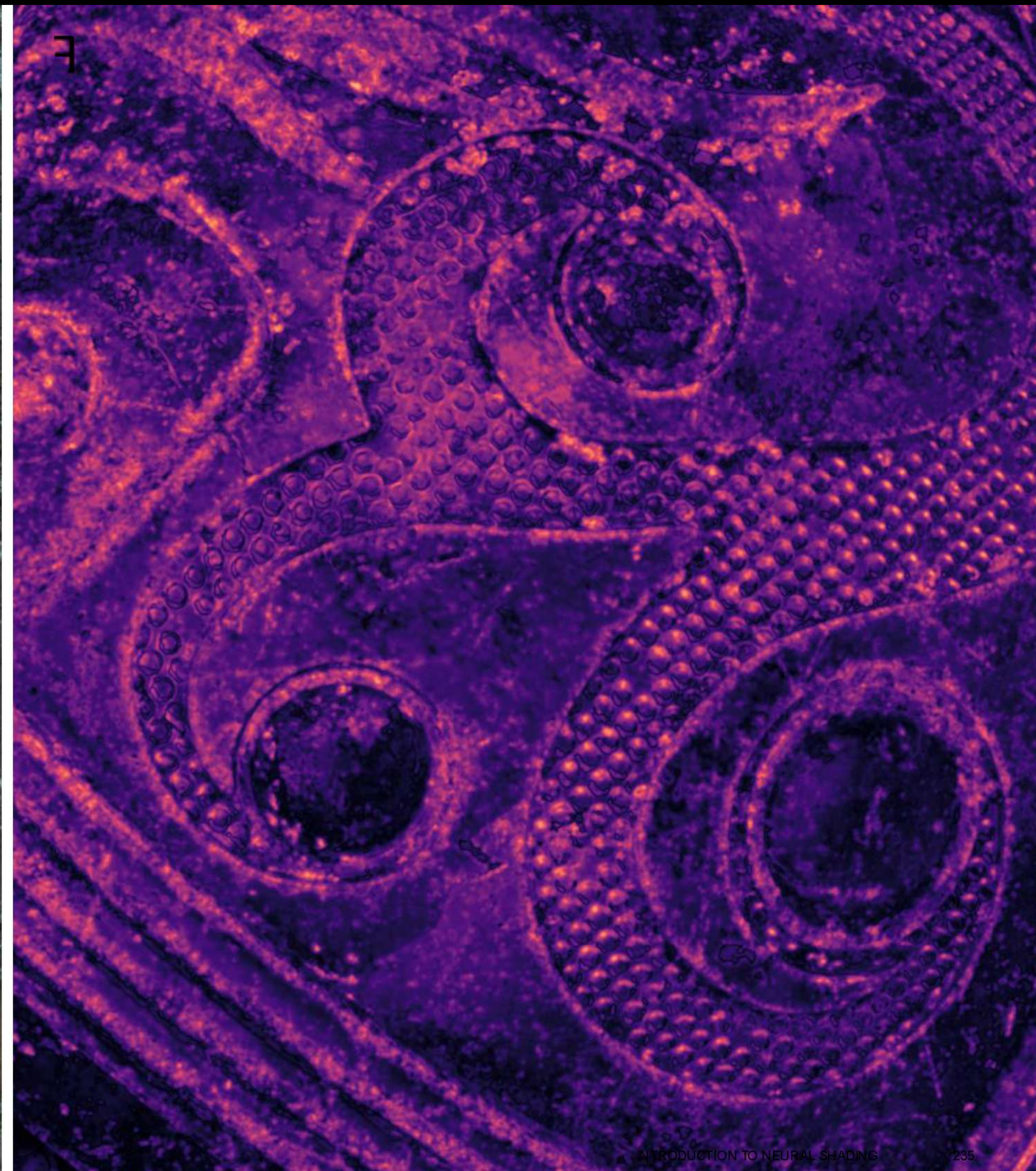
YOU ONLY NEED THIS TRICK DURING TRAINING!



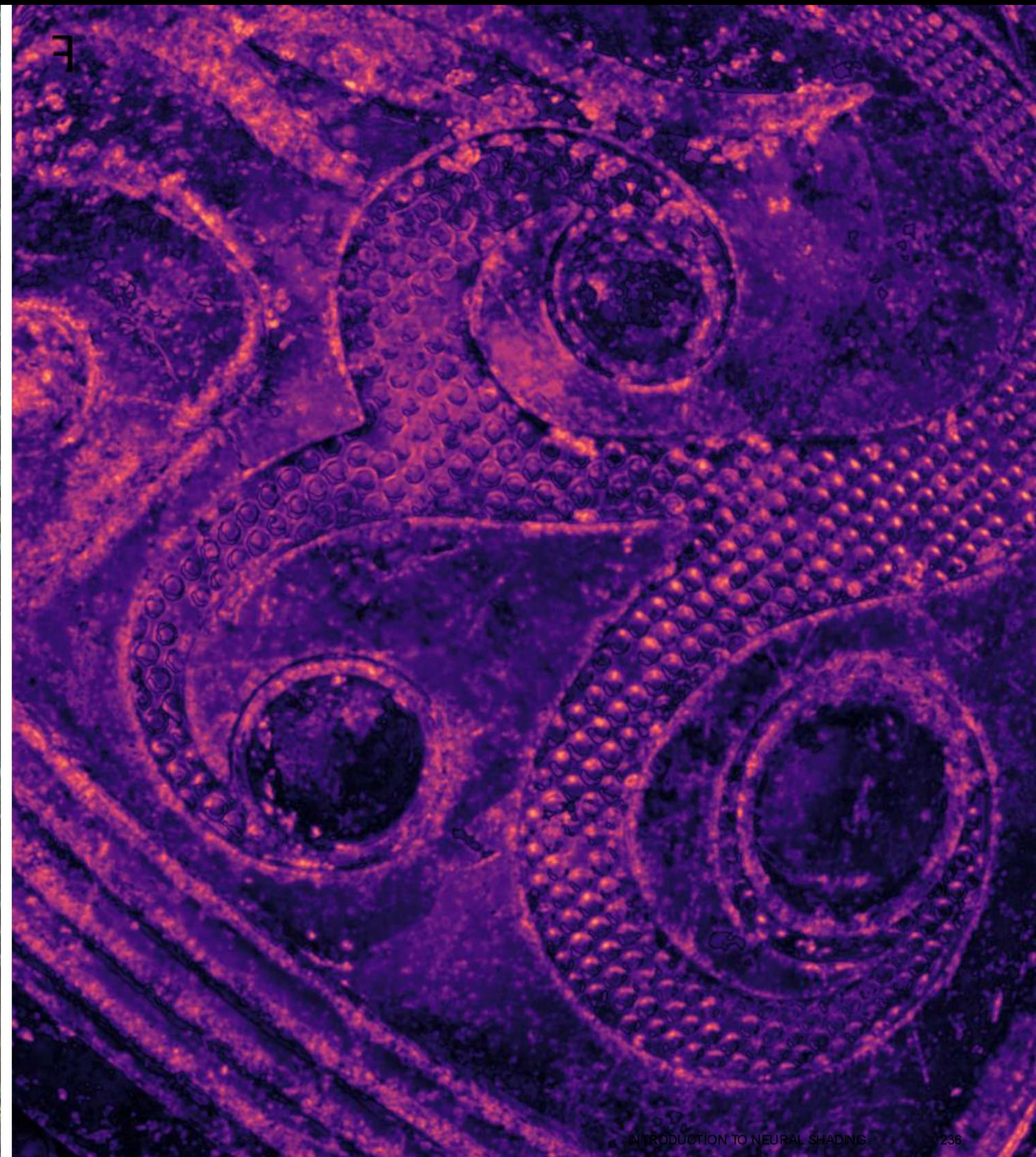
YOU ONLY NEED THIS TRICK DURING TRAINING!




Neural material

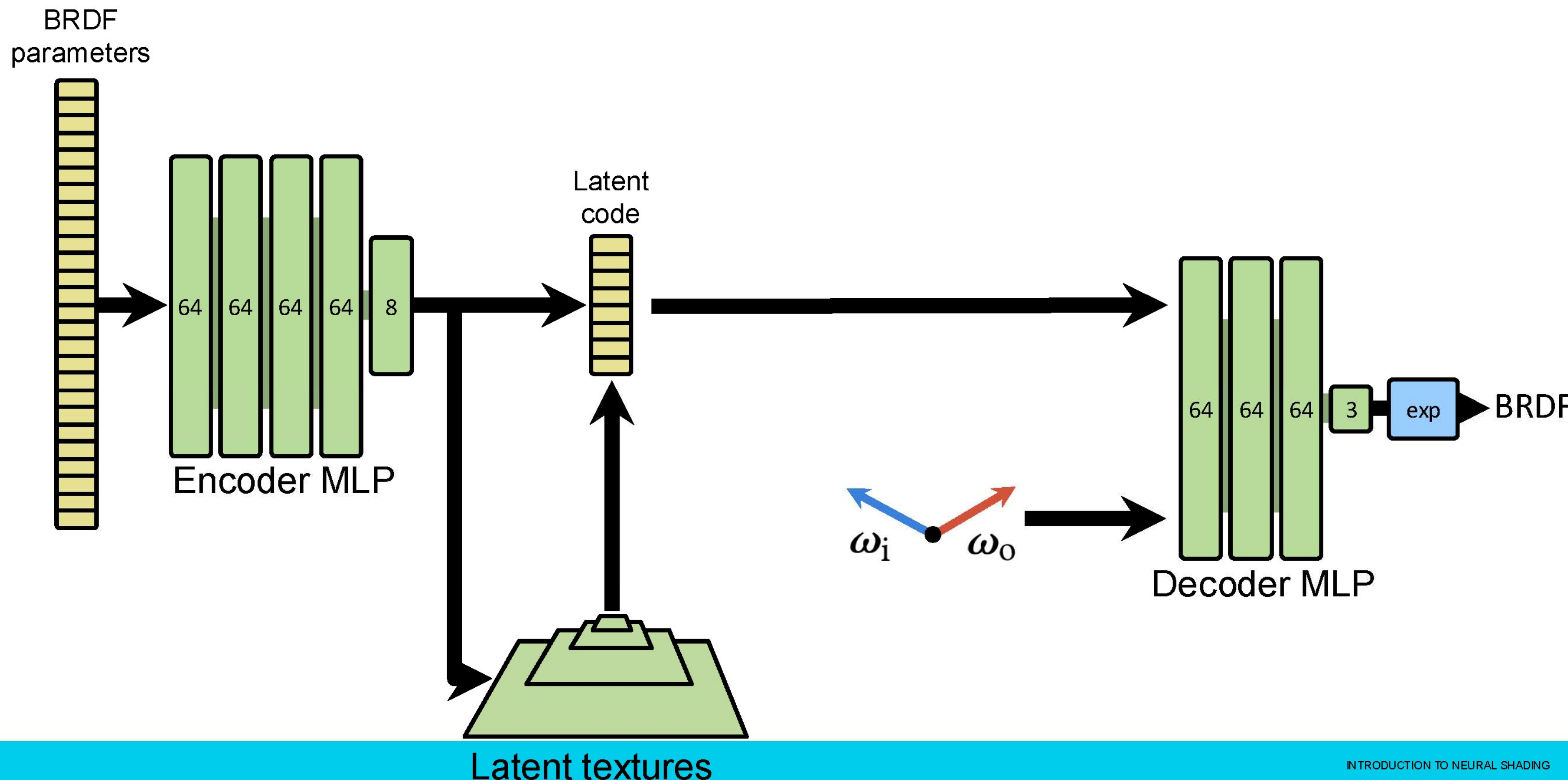



Reference

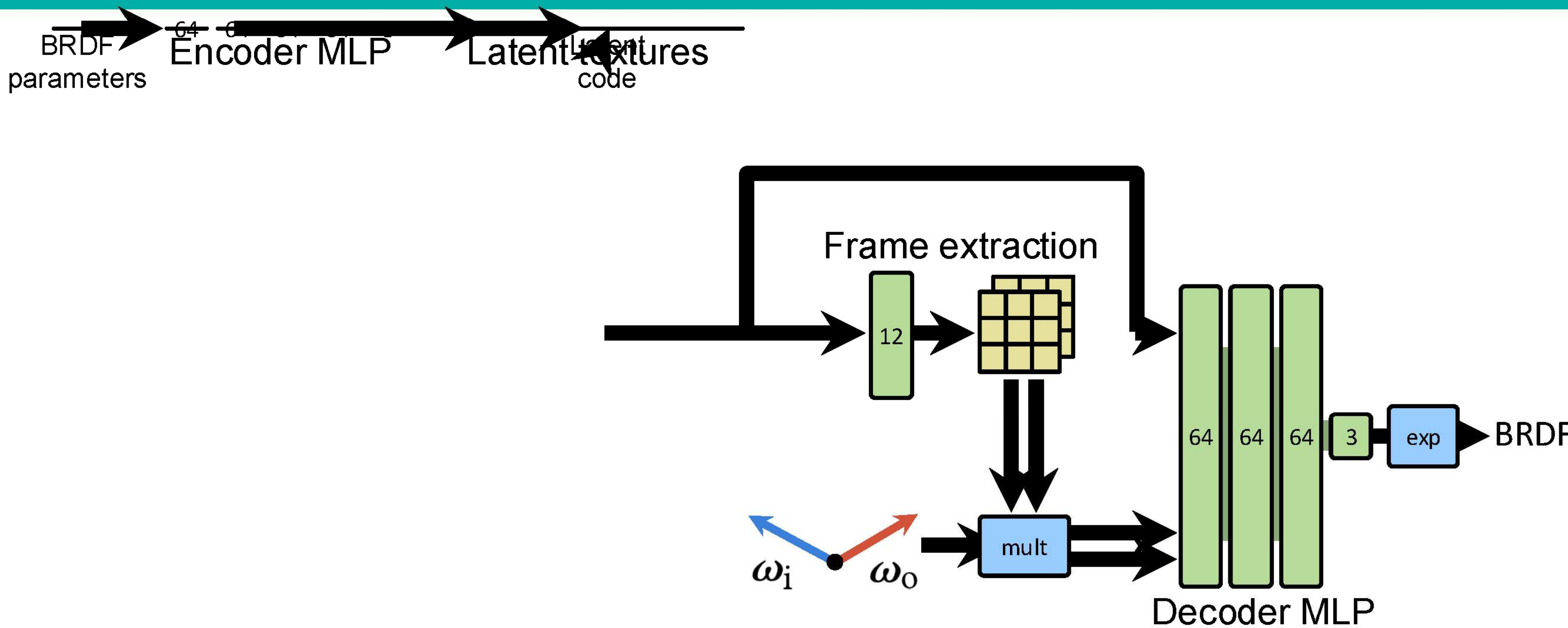


LET’S ADD A FIXED-FUNCTION FRAME TRANSFORM

SIGGRAPH 2025
Vancouver+ 10-14 August

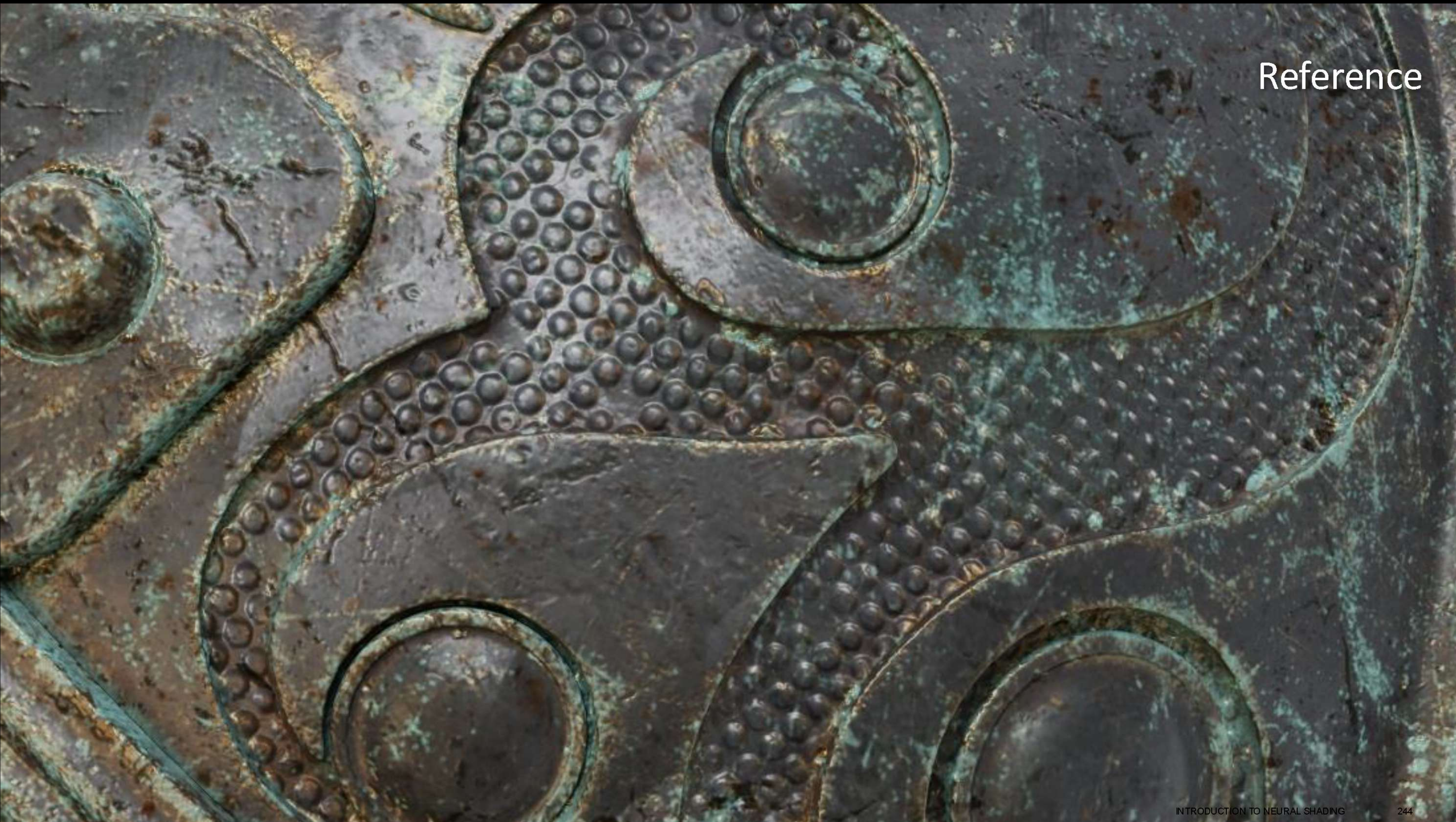


LET'S ADD A FIXED-FUNCTION FRAME TRANSFORM




Neural without rotations

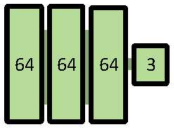
Neural with rotations


Reference

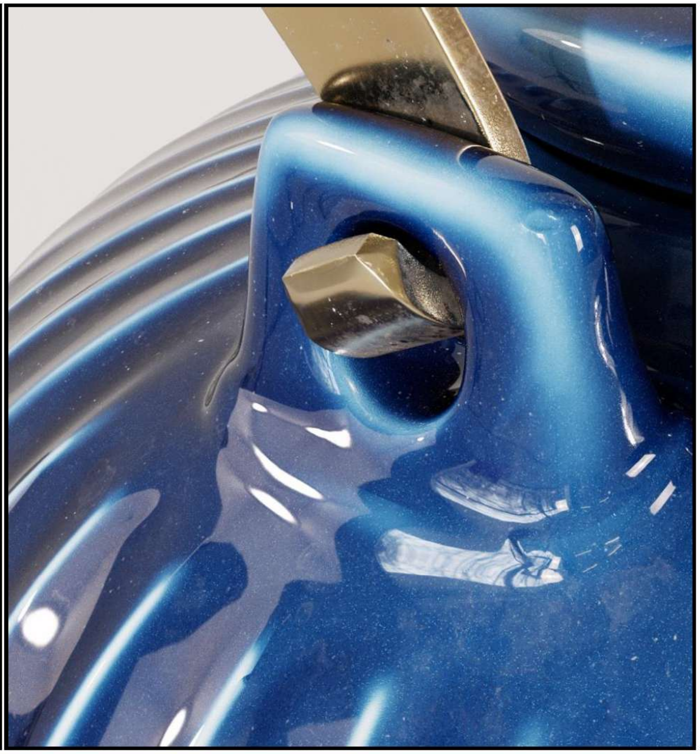
END RESULT

SIGGRAPH 2025
Vancouver+ 10-14 August

Neural
3x64 wide layers



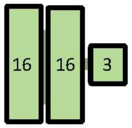
Reference

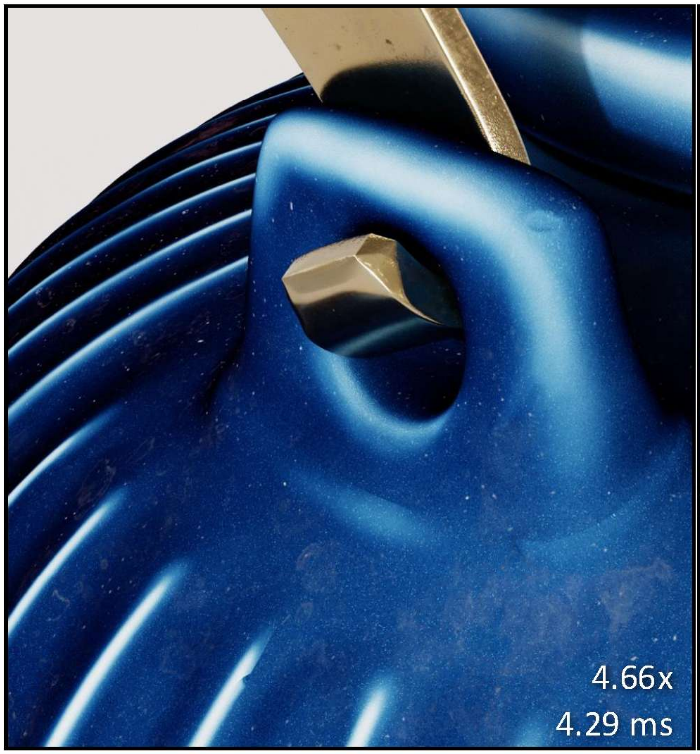


NEURAL SHADING GIVES YOU A QUALITY/COST DIAL

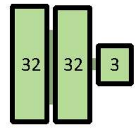
SIGGRAPH 2025
Vancouver+ 10-14 August

Neural
2x16 wide layers

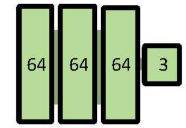




Neural
2x32 wide layers

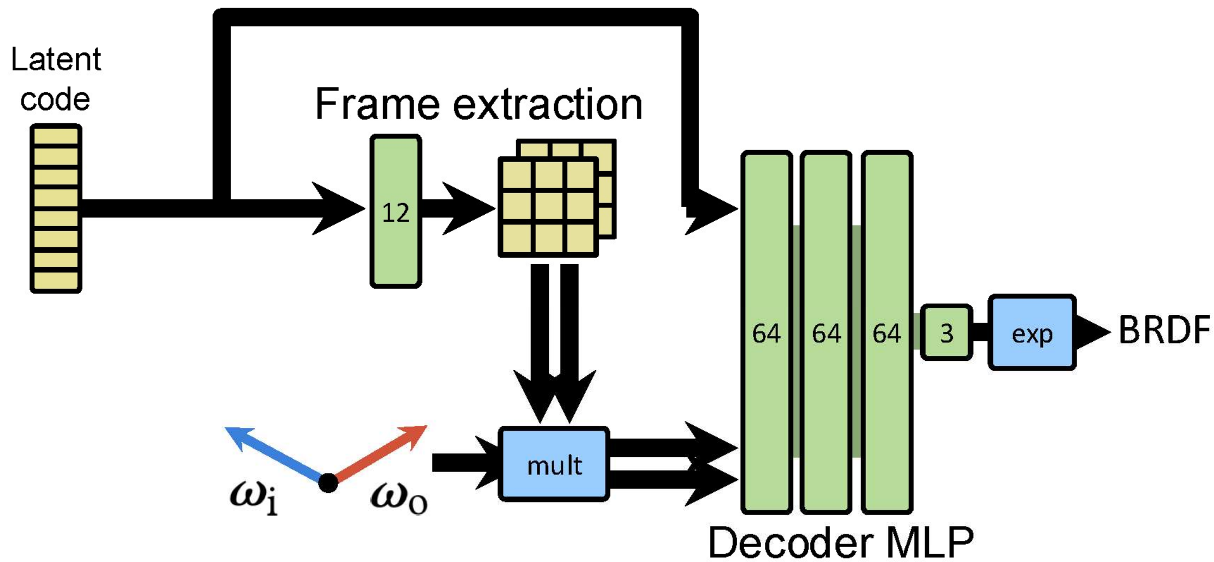


Neural
3x64 wide layers



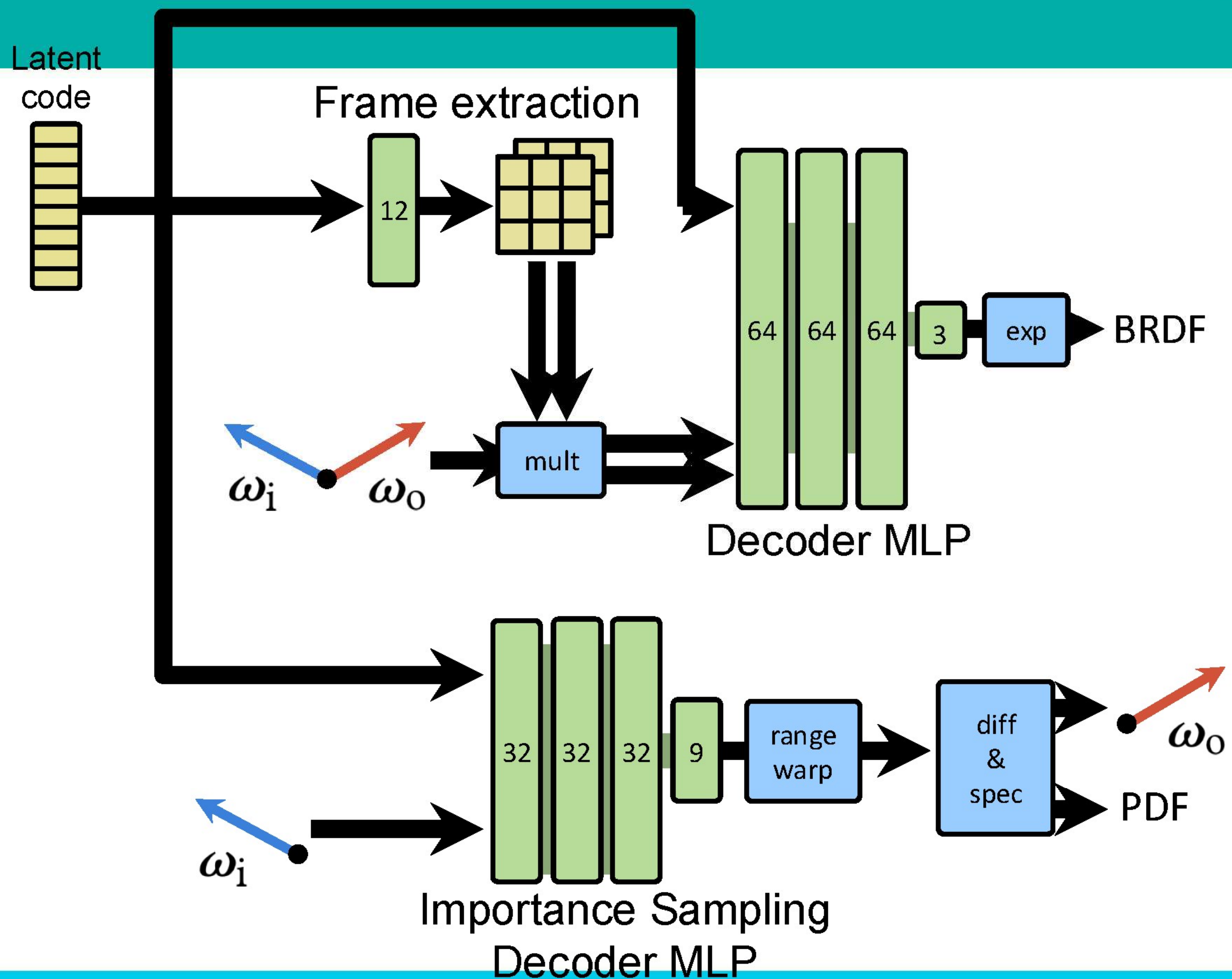
ADDING ADDITIONAL CAPABILITIES

SIGGRAPH 2025
Vancouver+ 10-14 August



ADDING ADDITIONAL CAPABILITIES

SIGGRAPH 2025
Vancouver+ 10-14 August



Thank you.