



Slang: Bridging Graphics & Machine Learning

Shannon Woods
Principal Technical Program Manager, NVIDIA
Slang Working Group Chair, Khronos
High Performance Graphics - June 23, 2025

Neural Graphics: The Next Frontier

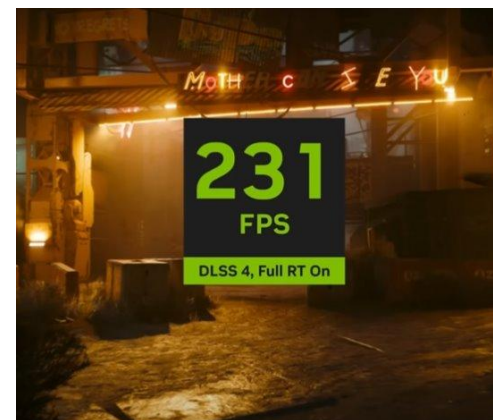
Graphics is adopting neural techniques to reach the next level of realism at real-time performance



Neural Materials



Neural Textures

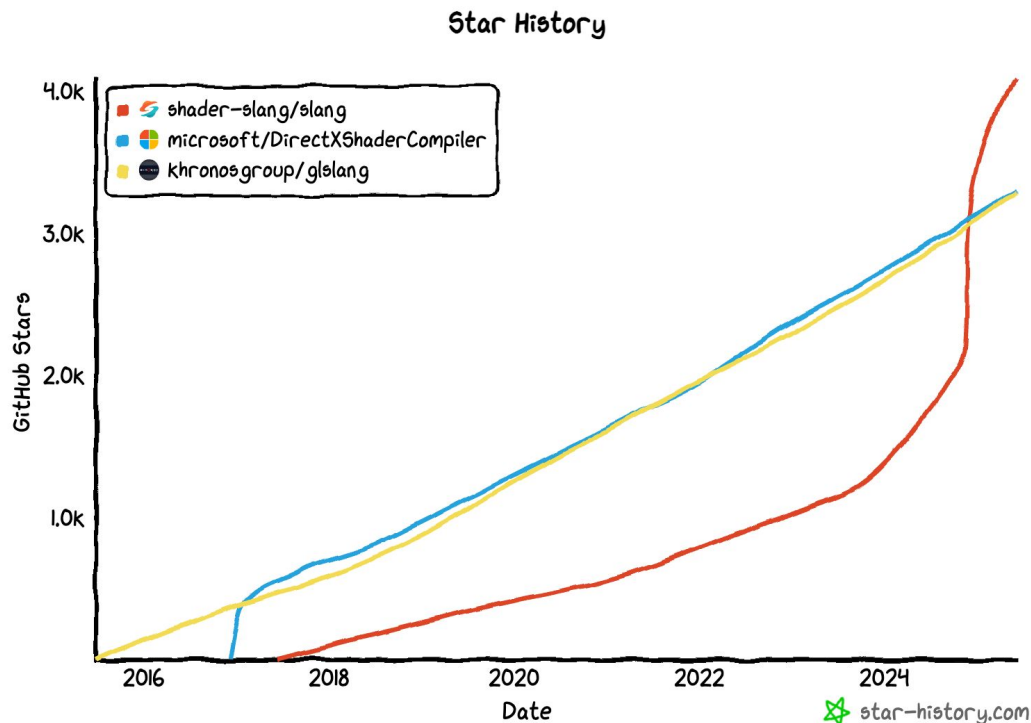


DLSS

Slang: The Language of Neural Graphics



Slang Community Strength



Join us on Discord!

50+ GitHub committers in 6mo
>700 Discord members

Active community!

Try it out on the web: <https://try.shader-slang.org>

See the documentation: <https://docs.shader-slang.org>

Graphics & ML Ecosystems Are Disjoint

But Slang and SlangPy are a bridge

Machine Learning

PyTorch

NVIDIA
CUDA

python™

SlangPy



Slang™
with Automatic Differentiation

Graphics

Microsoft®
DirectX®

Vulkan®

WebGPU

OpenGL®

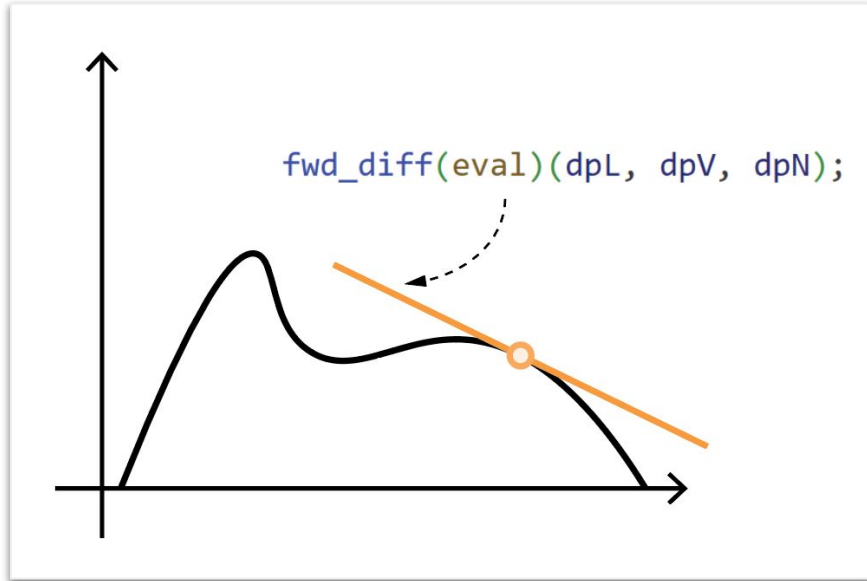


Shading Languages

Using work done in
the other ecosystem
is extremely difficult

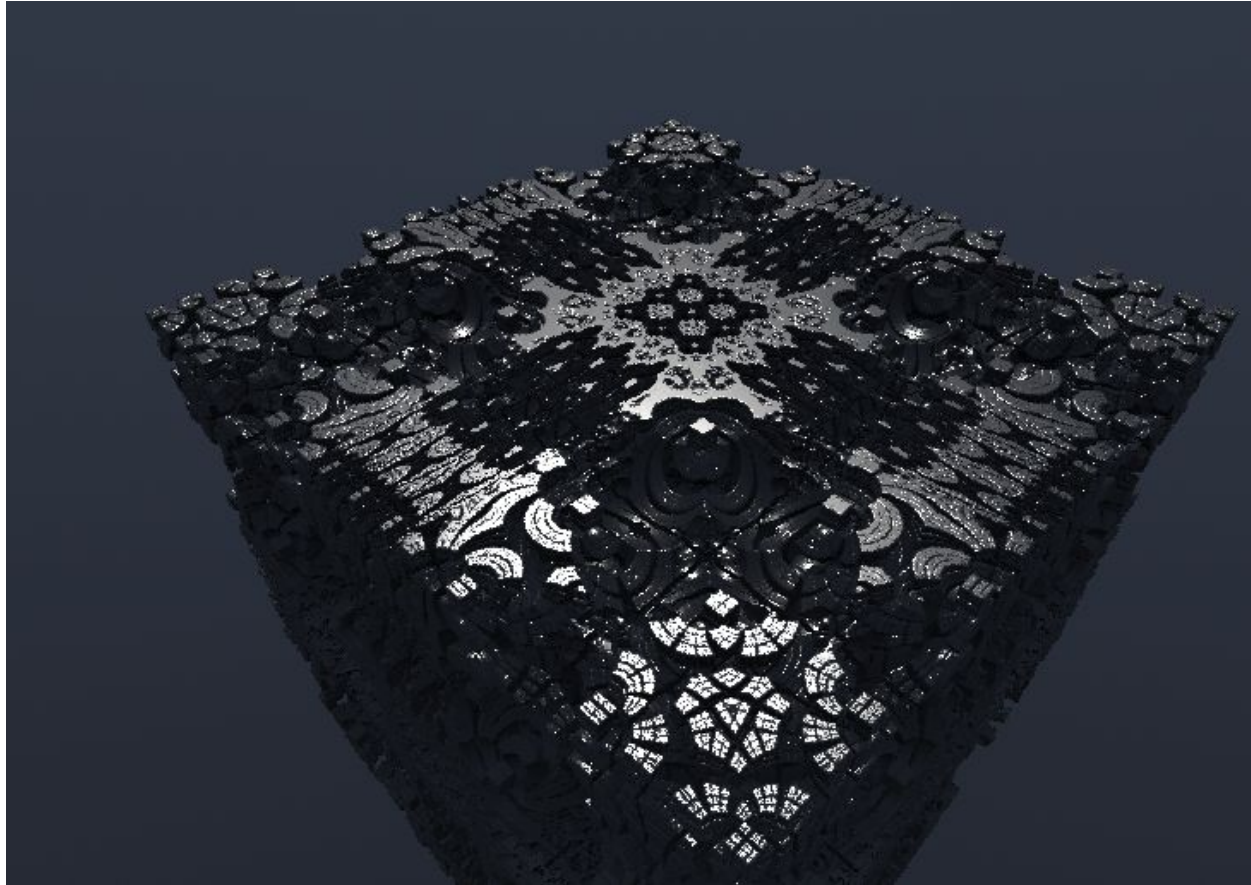
Automatic Differentiation

Your secret weapon for developing neural rendering applications

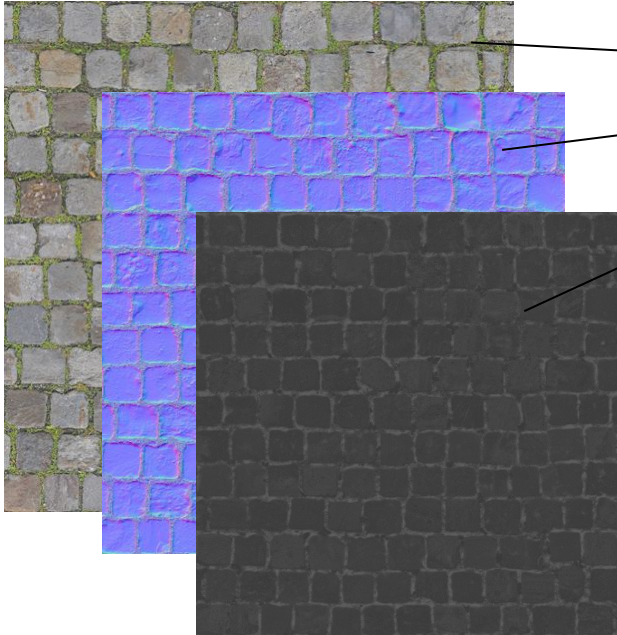


- Automatically computes exact derivatives of any function
- No manual gradient derivation required
- Supports arbitrary control flow & dynamic dispatch
- Enables any graphics function to become trainable

A Practical Application of Neural Shading



Practical Applications for Neural Shading



```
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);
    }
};
```

Practical Applications for Neural Shading

Render a material... or learn one

```
[Differentiable]
float3 render(no_diff float3 albedo, float3 normal, float roughness, no_diff float3 lightDir, no_diff float3 viewDir)
{
    float3 N = normalize(normal);
    float3 L = normalize(lightDir);           // Light direction.
    float3 V = normalize(viewDir);          // View direction.

    float metallic = 0.0;
    float specular = 1.0;

    float lightIntensity = 3.0;

    float3 brdf = DisneyBRDF(albedo, L, V, N, roughness, metallic, specular);
    return brdf * lightIntensity * max(0, dot(N, L));
}
```

Practical Applications for Neural Shading

Rendering and Downsampling

```
# 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(
        downsample(albedo_map, 2), downsample(normal_map, 2), 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 = output)

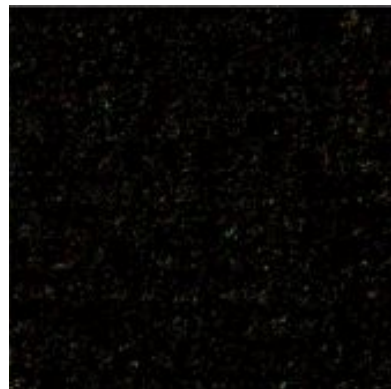
# Blit tensor to screen.
app.blit(output)
app.present()
```

Practical Applications for Neural Shading

Loss calculation

```
// Calculate the loss between a target color and calculated BRDF from the given
// inputs. This is used for training the normal and roughness values so these
// need to be differentiable. Other inputs are not trained.
[Differentiable]
float calculateLoss(no_diff float3 targetColor, no_diff float3 albedo, float3 normal, float roughness, no_diff
float3 lightDir, no_diff float3 viewDir)
{
    float3 inputColor = render(albedo, normal, roughness, lightDir, viewDir);
    float3 diff = targetColor - inputColor;

    float loss = (diff.x * diff.x + diff.y * diff.y + diff.z * diff.z) / 3.0;
    return loss;
}
```



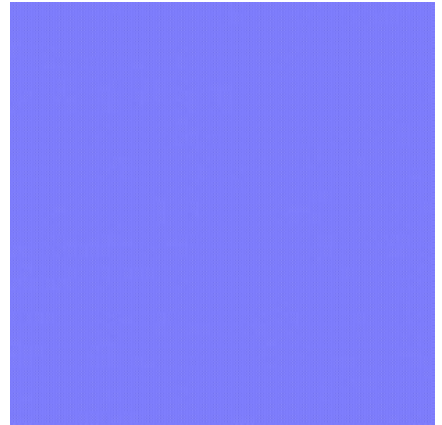
Practical Applications for Neural Shading

Automatic differentiation

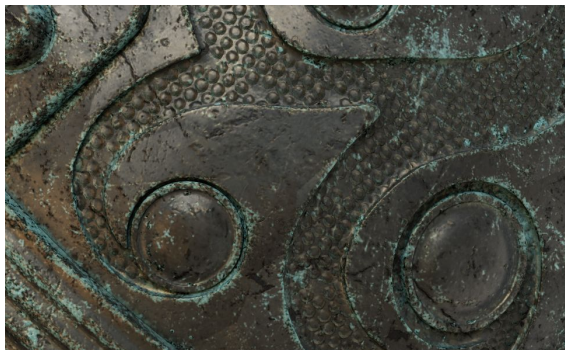
```
iterations = 10000
for iter in range(iterations):

    # Take the function that calculates the loss, i.e. the difference between the downsampled output
    # and the output calculated with downsampled albedo/normals, and run it 'backwards'
    # This propagates the gradient of training_loss back to the gradients of trained_normals.
    module.calculateLoss.bwds(downsampled_ideal_result, downsampled_albedo_map,
                              trained_normals, trained_roughness, light_dir, view_dir, _result=training_loss)
    # trained_normals.grad_out now tells us how trained_normals needs to change
    # so that training_loss changes by training_loss.grad_in

    module.updateValues(trained_normals, trained_normals.grad, learning_rate)
    module.updateValues(trained_roughness, trained_roughness.grad, learning_rate)
```



Solving Really Big Problems



Neural Material Examples

Summary

- Join us on Discord



- Get started with our web playground!

