Vulkanised 2025

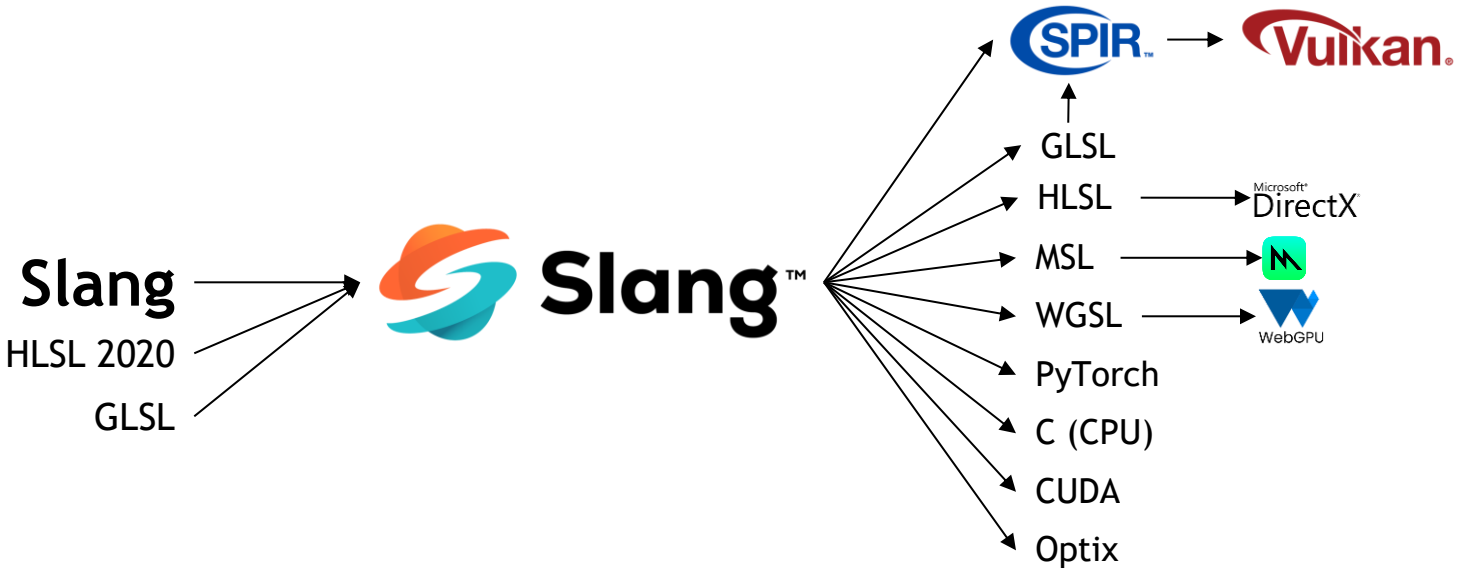
The 7th Vulkan Developer Conference
Cambridge, UK | February 11-13, 2025

Slang is for Neural Graphics

Shannon Woods, NVIDIA

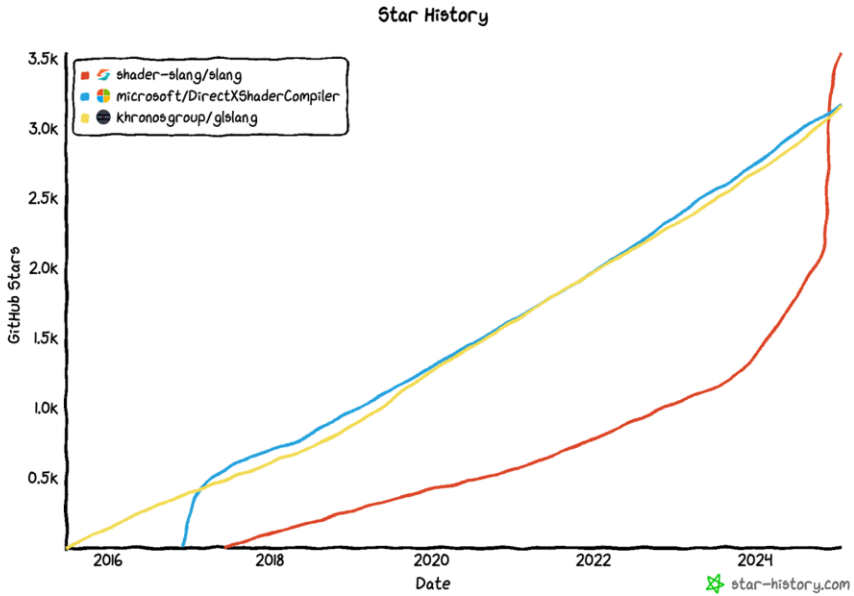
Quick Recap: What’s Slang?

Open-Source, Cross-Platform Compiler



Rapidly Growing Community

- **Join the Discord!**
  - 400 members and counting!



- **File issues, create PRs at** github.com/shader-slang

Community Feedback Fueling Development

- Specialization constant & push constant improvements & fixes
- SPIR-V pointer support improvements
- SPIR-V / GLSL improvements
- Reflection API & binding improvements
- Bindless support

The Good Stuff: Slang & Neural Graphics

Neural Graphics: The Next Frontier

Graphics is adopting more neural techniques to reach the next level of realism

Neural Materials

Neural Textures

DLSS

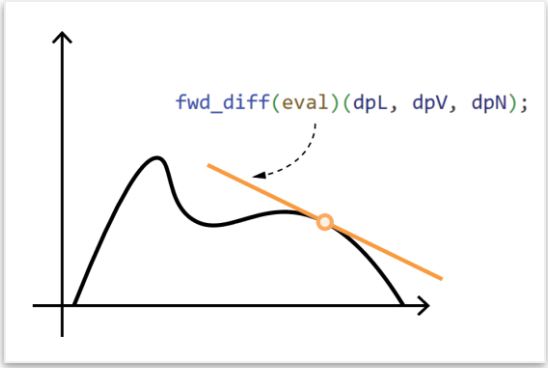
Easily maintainable differentiable code with Autodiff

- Differentiable functions power gradient descent solution approaches
  - Slang brings automatic differentiation to languages optimized for GPU usage
  - Developers can optionally provide custom derivatives for just the portions of a shader where it's necessary – flexibility & control
  - Autodiff support includes arbitrary control flow & dynamic dispatch

```
interface IBRDF : IDifferentiable
{
    [Differentiable] float3 eval(float3 L, float3 V, float3 N);
}

struct GGXBRDF : IBRDF
{
    float3 baseColor;
    float roughness;
    float metallic;
    float specular;

    [Differentiable] float3 eval(float3 L, float3 V, float3 N)
    {
        float NdotL = dot(N, L);
        float NdotV = dot(N, V);
        if (NdotL < 0 || NdotV < 0)
```



Writing gradient propagation code & keeping it in sync is hard.

```
void bwd_get_covariance_from_quat_scales(
  inout pair<float4, float4> dpq,
  inout pair<float4, float3> dps,
  float3x3 dOut)
{
  float tmp6 = dpq.primal[2];
  float tmp7 = tmp6 * tmp6;
  float tmp8 = dpq.primal[3] * dpq.primal[3];
  float tmp9 = dpq.primal[1] * dpq.primal[2];
  float tmp10 = dpq.primal[0] * dpq.primal[3];
  float tmp11 = dpq.primal[1] * dpq.primal[3];
  float tmp12 = dpq.primal[0] * dpq.primal[2];
  float tmp13 = dpq.primal[1] * dpq.primal[1];
  float tmp14 = dpq.primal[2] * dpq.primal[3];
  float tmp15 = dpq.primal[0] * dpq.primal[1];
  float3x3 rotation_matrix = float3x3(
     1.0 - 2.0 * (tmp7 + tmp8),
     2.0 * (tmp9 - tmp10),
     2.0 * (tmp11 + tmp12),
     2.0 * (tmp9 + tmp10),
     1.0 - 2.0 * (tmp13 + tmp8),
     2.0 * (tmp14 - tmp15),
     2.0 * (tmp11 - tmp12),
     2.0 * (tmp14 + tmp15),
     1.0 - 2.0 * (tmp13 + tmp7));
  float3x3 scales_matrix = float3x3(dps.primal[0], 0.0, 0.0,
                                                                    0.0, dps.primal[1],
0.0,
                                                                    0.0, 0.0,
dps.primal[2]);
  float3x3 tmp16 = s_primal_ctx_mul(rotation_matrix, scales_matrix);
  float3x3 tmp17 = transpose(tmp16);
  float3x3 tmp18 = float3x3(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
  ForwardContext tmp19;
  tmp19.primal = tmp16;
  tmp19.differential = tmp18;
  ForwardContext tmp20;
  tmp20.primal = tmp17;
  tmp20.differential = tmp18;
  s_bwd_prop_mul(tmp19, tmp20, dOut);
  float3x3 tmp21 = tmp19.differential + transpose(tmp20.differential);
  ForwardContext tmp22;
  tmp22.primal = rotation_matrix;
  tmp22.differential = tmp18;
  ForwardContext tmp23;
  tmp23.primal = scales_matrix;
  tmp23.differential = tmp18;
  s_bwd_prop_mul(tmp22, tmp23, tmp21);
  float tmp24 = 2.0 * - tmp22.differential[2][2];
  float tmp25 = 2.0 * tmp22.differential[2][1];
  float tmp26 = 2.0 * tmp22.differential[2][0];
  float tmp27 = 2.0 * tmp22.differential[1][2];
  float tmp28 = tmp25 + - tmp27;
  float tmp29 = tmp25 + tmp27;
  float tmp30 = 2.0 * - tmp22.differential[1][1];
  float tmp31 = dpq.primal[1] * (tmp24 + tmp30);
  float tmp32 = 2.0 * tmp22.differential[1][0];
  float tmp33 = 2.0 * tmp22.differential[0][2];
  float tmp34 = - tmp26 + tmp33;
  float tmp35 = tmp26 + tmp33;
  float tmp36 = 2.0 * tmp22.differential[0][1];
  float tmp37 = tmp32 + - tmp36;
  float tmp38 = tmp32 + tmp36;
  float tmp39 = 2.0 * - tmp22.differential[0][0];
  float tmp40 = dpq.primal[3] * (tmp30 + tmp39);
  float tmp41 = dpq.primal[2] * (tmp24 + tmp39);
  float tmp42 = dpq.primal[2] * tmp29 + dpq.primal[1] * tmp35
              + dpq.primal[0] * tmp37 + tmp40 + tmp40;
  float tmp43 = dpq.primal[3] * tmp29 + dpq.primal[0] * tmp34
              + dpq.primal[1] * tmp38 + tmp41 + tmp41;
  float tmp44 = dpq.primal[0] * tmp28 + tmp31 + tmp31 + dpq.primal[3] * tmp35
              + dpq.primal[2] * tmp38;
  float tmp45 = dpq.primal[1] * tmp28 + dpq.primal[2] * tmp34 + dpq.primal[3] * tmp37;
  dps.primal = dps.primal;
  dps.differential = float3(
     tmp23.differential[2][2];
     tmp23.differential[1][1];
     tmp23.differential[0][0]);
  dpq.primal = dpq.primal;
  dpq.differential = float4(tmp42, tmp43, tmp44, tmp45);
}
```

Slang makes it easy

```
// Excerpted from https://github.com/google/slang-gaussian-rasterization/.../slang/utils.slang
[Differentiable]
float3x3 get_covariance_from_quat_scales(float4 q, float3 s) {
   float r = q[0], x = q[1], y = q[2], z = q[3];

   float3x3 rotation_matrix = float3x3(
       1 - 2 * (y * y + z * z), 2 * (x * y - r * z), 2 * (x * z + r * y),
       2 * (x * y + r * z), 1 - 2 * (x * x + z * z), 2 * (y * z - r * x),
       2 * (x * z - r * y), 2 * (y * z + r * x), 1 - 2 * (x * x + y * y));

   float3x3 scales_matrix = float3x3(s[0], 0, 0,
                                     0, s[1], 0,
                                     0, 0, s[2]);

   float3x3 L = mul(rotation_matrix, scales_matrix);

   return mul(L, transpose(L));
}
```

3D Gaussian Splatting in Slang!

google/slang-gaussian-rasterization

- Ported from CUDA to Slang by George Kopanas, the original author

40%

Reduction in lines of code

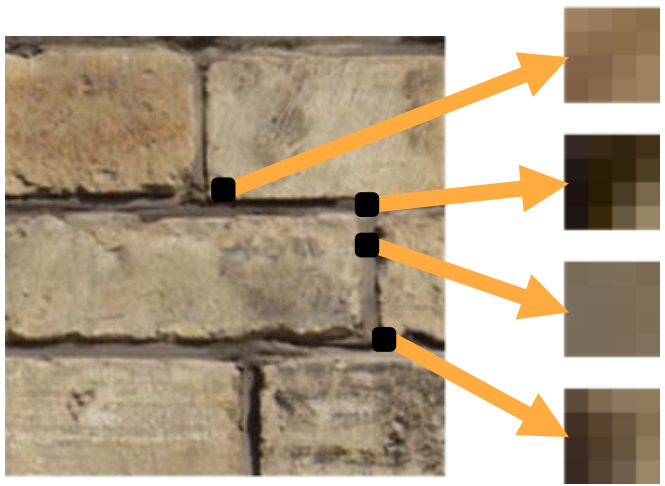
Equal

Runtime performance

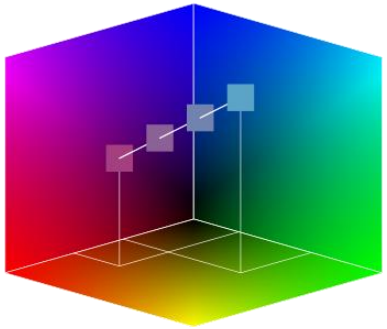
- The authors are committed to use Slang for future research projects.

Auto-diff enables creative solutions to traditional problems

Block Compression (BC7-mode6)



For every 4x4 block



1. Find 1 pair of end-points in color-space
2. Linear interpolation coefficient for each texel

Can be framed as an optimization problem

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

Writing a texture compressor by writing a decompressor

1. Implement a de-compressor

```
TextureBlock decompress(CompressedTextureBlock blockCoefficients)
{
      // Implement BC7 decompression here. Super easy!
}
```

2. Implement a loss function

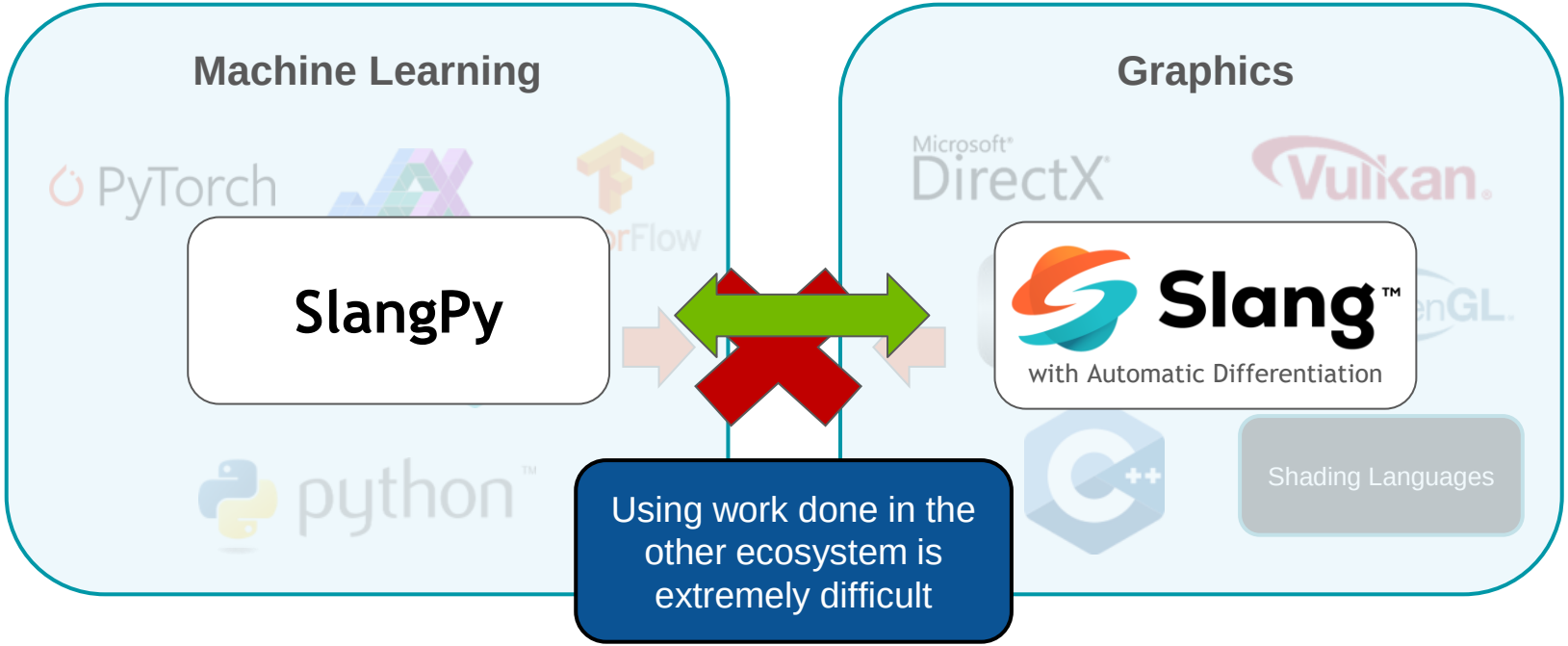
```
float loss(TextureBlock groundtruth, CompressedTextureBlock compressed)
{
    return distance(decompress(compressed), groundtruth);
}
```

3. Use gradient descent to find the compression coefficients

```
CompressedTextureBlock compress(TextureBlock data)
{
      CompressedTextureBlock result = random_init();
      for (int i = 0; i < N_STEPS; i++) {
          derivative = computeDerivativeOfLoss(result, data);
          result += derivative * learning_rate;
      }
      return result;
}
```

Graphics & ML Ecosystems Are Disjoint

Despite both workloads running on the GPU



SlangPy: The bridge between Python & Graphics ecosystems

Slang makes writing kernel code easy

- Rich set of built-in functions for kernel programming
- Modules, generics, and modern language constructs
- Automatic differentiation
- IntelliSense

But running shader code is traditionally laborious

C++ setup & dispatch w/ help from a rich library

```
// C++ psedo code

// Create compute shader and get vars/state
mpProgram = Program::createCompute(mpDevice, "tonemapper.slang", "main",
                                   DefineList(), SlangCompilerFlags::TreatWarningsAsErrors);
mpVars = ProgramVars::create(mpDevice, mpProgram->getReflector());
mpState = ComputeState::create(mpDevice);

// Setup variables
auto var = mpVars->getRootVar();
var["g_params"]["filmic"] = 0.5;
var["g_params"]["input"] = inputTexture;
var["g_params"]["output"] = outputTexture;

// Dispatch shader
auto pProgram = mpProgram;
uint3 numGroups = div_round_up(uint3(outputTexture.width, outputTexture.height, 1u),
                               pProgram->getReflector()->getThreadGroupSize());
mpState->setProgram(pProgram);
pRenderContext->dispatch(mpState.get(), mpVars.get(), numGroups);
```

Slang kernel

```
[[shader("compute")]]
[[numthreads(8, 8, 1)]]
void main(uint3 tid: SV_DispatchThreadID)
{
    uint width,height;
    g_params.output.GetDimensions(width,height);
    if(tid.x > width || tid.y > height)
        return;

    float4 color = g_params.input.Load(int3(tid.xy,0));
    color.rgb = lerp(color.rgb, aces_film(color.rgb), g_params.filmic);
    g_params.output[tid.xy] = color;
}
```

Let’s make it simple.

Python setup & dispatch

```python
module = slangpy.Module.load_from_file(device, "test_modules.slang")

module.tonemap(0.5, inputTexture, _result=outputTexture)
```

Slang kernel

```
float3 aces_film(float3 x)
{
   float a = 2.51;
   float b = 0.03;
   float c = 2.43;
   float d = 0.59;
   float e = 0.14;
   return saturate((x * (a * x + b)) /
                   (x * (c * x + d) + e));
}

float4 tonemap(float filmic, float4 input)
{
   return lerp(input.rgb,
               aces_film(input.rgb),
               filmic);
}
```

Just:

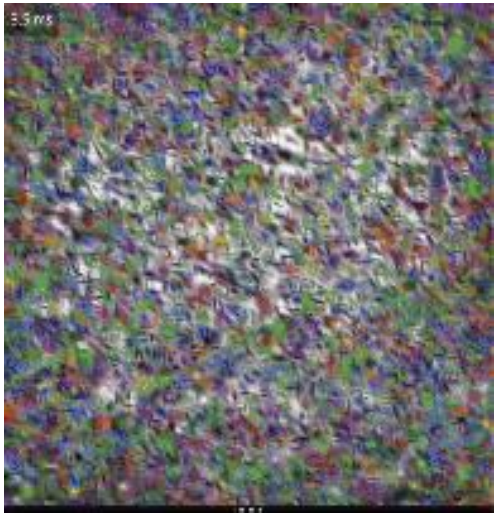
- Load a Slang module
- Call a Slang function directly with a PyTorch tensor, numpy array, or a python array/dictionary

What kind of problems can you solve?

```
struct SymmetricGaussian2D
{
    float2 center;
    float sigma;
    float3 color;
}
```

Starting with a very simple means of representing an image…

And randomly generated data…



“Learn” the parameters that for the data to arrive at a target image

Computing an image from Gaussians

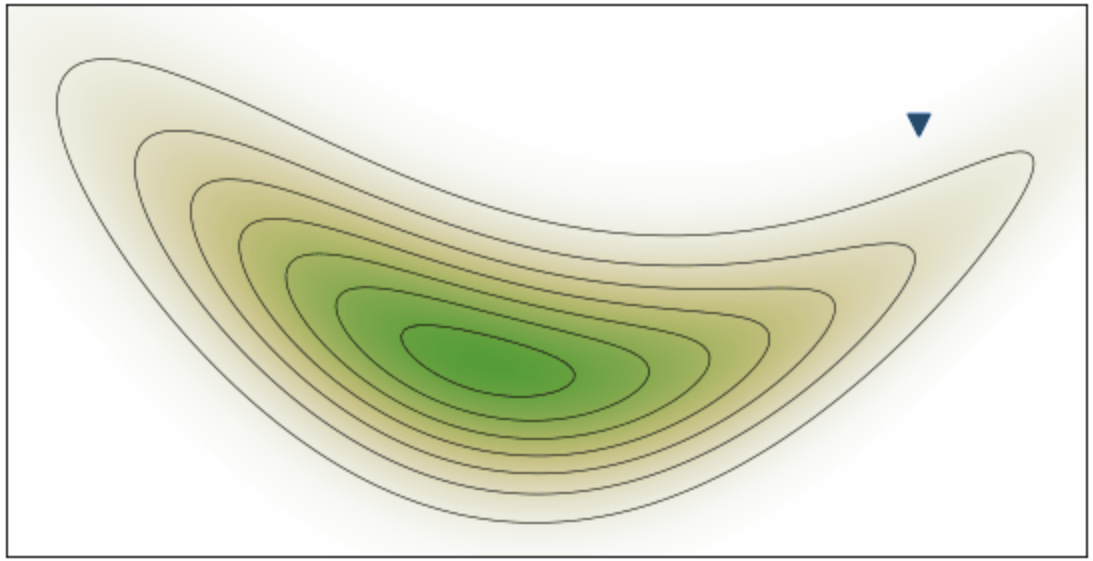
```
float4 computeImageFromGaussian(GradInOutTensor<float, 1> gaussiansBuffer, float2 uv)
{
    foreach (G in gaussiansBuffer)
    {
        result += G.getColor(uv);
    }
    return result;
}
```

But how do you determine all the parameters?

First define a loss function:the difference between what you computed and the target image

```
void perPixelLoss(
   GradInOutTensor<float4, 2> output,
   uint2 uv,
   GradInOutTensor<float, 1> gaussiansBuffer,
   Texture2D<float4> targetTexture)
{
   float4 inferred = computeImageFromGaussians(gaussiansBuffer, uv);
   float4 reference = targetTexture[uv];
   output[uv] = dot(reference - inferred, reference - inferred); // L2 distance
}
```

Gradient descent



Justinkunimune, CC0, via Wikimedia Commons

Gradient descent

Training: compute the gradient of the loss WRT all of the parameters

```
# load the Slang module we wrote
module = slangpy.Module.load_from_file(device, "diffsplatting2d.slang")

for iter in range(iterations):
   # Back-propagate the unit per-pixel loss with auto-diff.
   module.perPixelLoss.bwds(per_pixel_loss, uv, blobs, input_image)

   # Update the parameters using the gradients - move in the direction of lower loss
   module.adamUpdate(blobs, blobs.grad_out, adam_first_moment, adam_second_moment)
```

SlangPy Example: 2D Gaussian Splatting Training

Reconstructing 2D Image with Gaussians

[Link to code]

90 lines of Python + **800** lines of Slang

Same Slang code also runs in the browser through WebGPU:
https://try.shader-slang.org?demo=gsplat2d-diff

Summary

- Slang opens the door for neural graphics techniques
- Slang and SlangPy work together as a bridge, making it easy to bring neural rendering into machine learning contexts, and to bring machine learning into the rendering world
- Neural techniques benefit not just things like scene reconstruction, but also bring new ways to address traditional problems like texture compression or fluid simulation

Get started with SlangPy – A collection of great samples in our repository will get you started!

https://github.com/shader-slang/slangpy/tree/main/examples