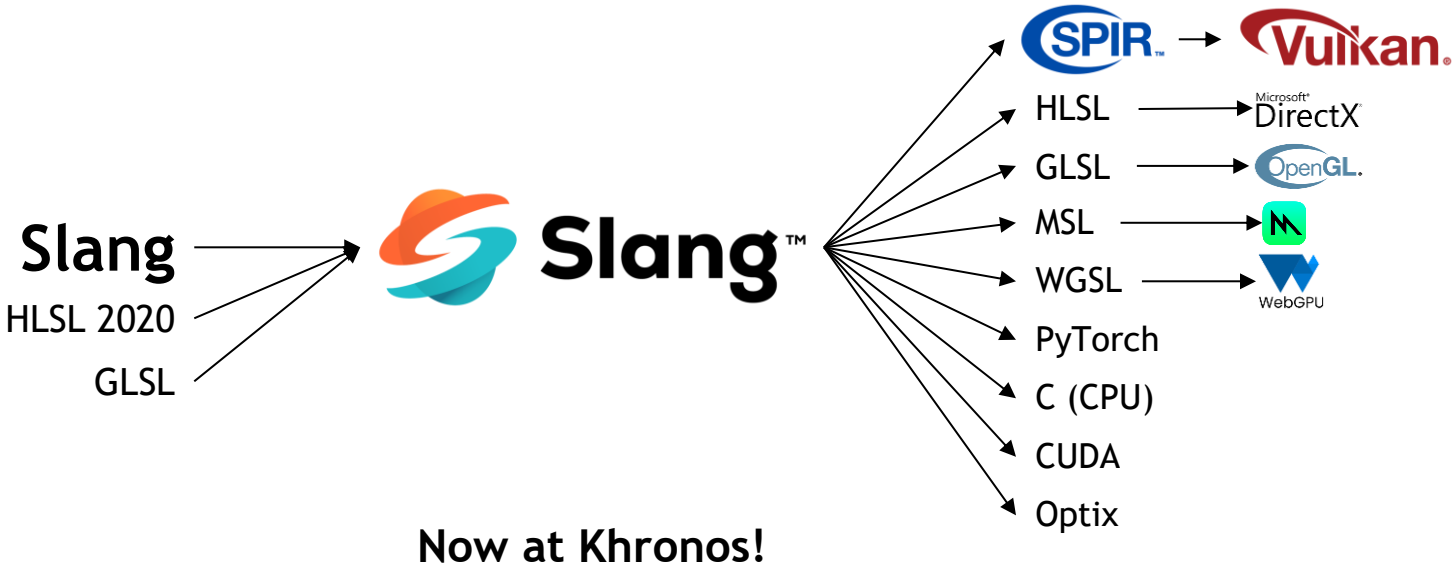
Slang and the 3D Shading Landscape

Shannon Woods, NVIDIA

The Shading Language Landscape Today

- Shader codebases have become incredibly large & complex
- Developers need to deploy to many platforms
- Shader combinatorial explosion
- New graphics techniques & neural graphics discontinuity
- GLSL no longer innovating new language features

Open-Source, Cross-Platform Compiler



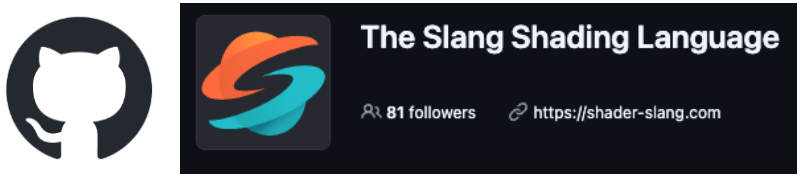
Now at Khronos!

Slang + Khronos = Developers Win

- Shading language diversity means more competition & innovation
- No single company controls the language, so it can evolve as developers need

For developers, by developers

- Community structure built from OSS best practices
- **Any company or individual** is welcome to become a contributor, not just Khronos members
- Decision-making and development in the open – you can join technical conversations today on Discord, or propose features directly to the repository.
- Slang developers make the decisions about what goes into the language, and you can become one



Why Another Shading Language?

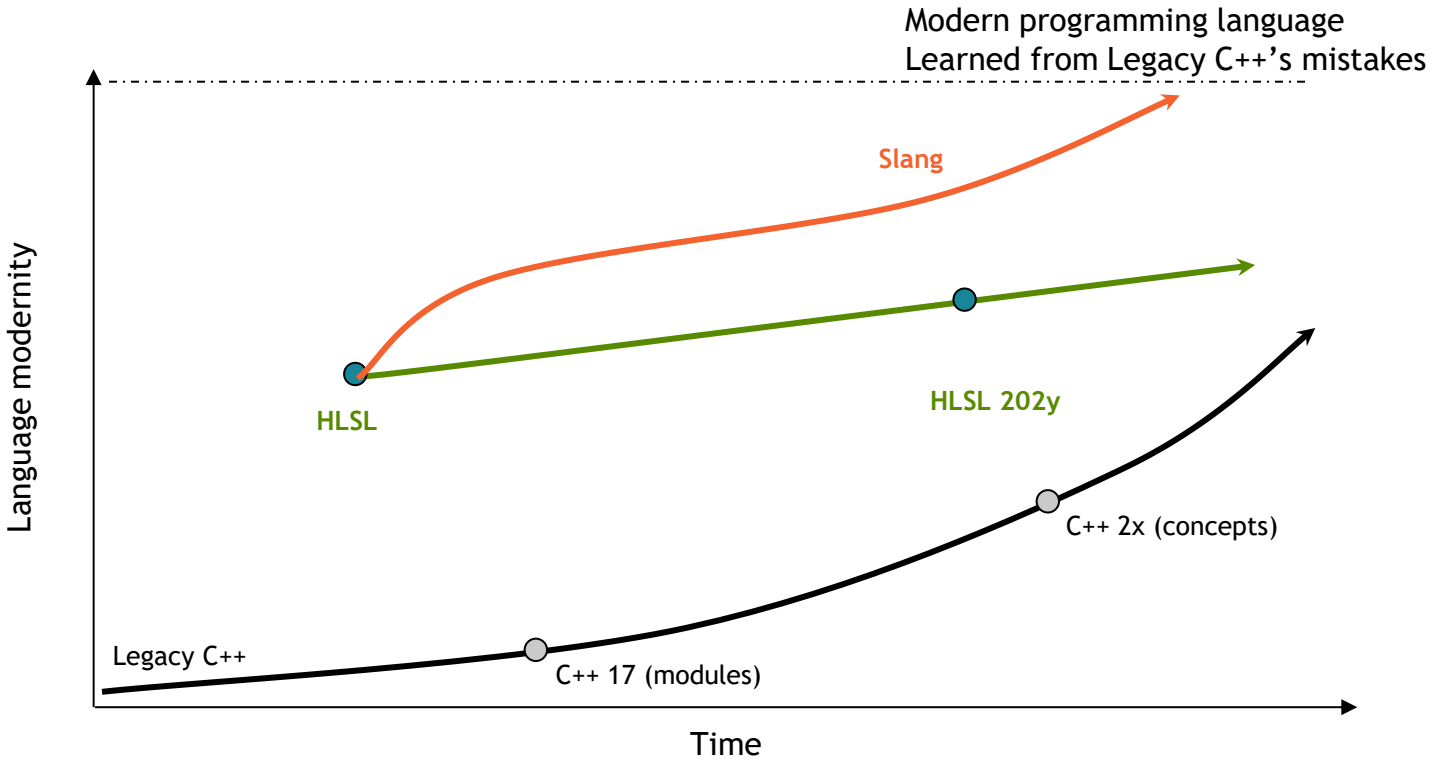
	GLSL	MSL	WGSL	HLSL	Slang
Actively Evolving	NO	YES	YES	YES	YES
Modular Code Management	NO	NO	NO	NO	YES
Converging with C++	NO	YES	NO	YES	NO*
Auto-diff / Neural Shading	NO	NO	NO	NO	YES
Diverse Backend Targets	NO	NO	NO	DXIL and SPIR-V	YES
Open-Source Compiler(s)	YES	NO	YES	YES	YES
Open Governance	YES	NO	YES	NO	YES

*** Slang and HLSL are taking complementary evolutionary paths**

HLSL will remain and evolve as a critically important shading language for many developers

Language diversity and choice is good for the graphics ecosystem!

Language Evolution



What makes Slang special?

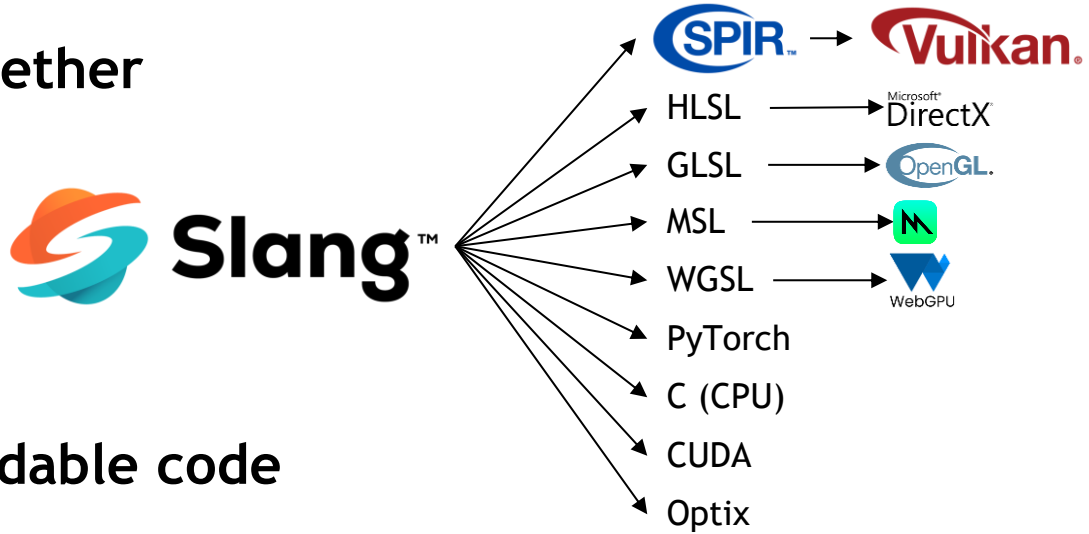
- **Cross-compilation in Slang is easy and ergonomic**
  - a seamless way – integrated in one place
  - tooling just works

- **Automatic differentiation**
  - unique among shading languages
  - starting to show up as a necessity
  - for most AI graphics work

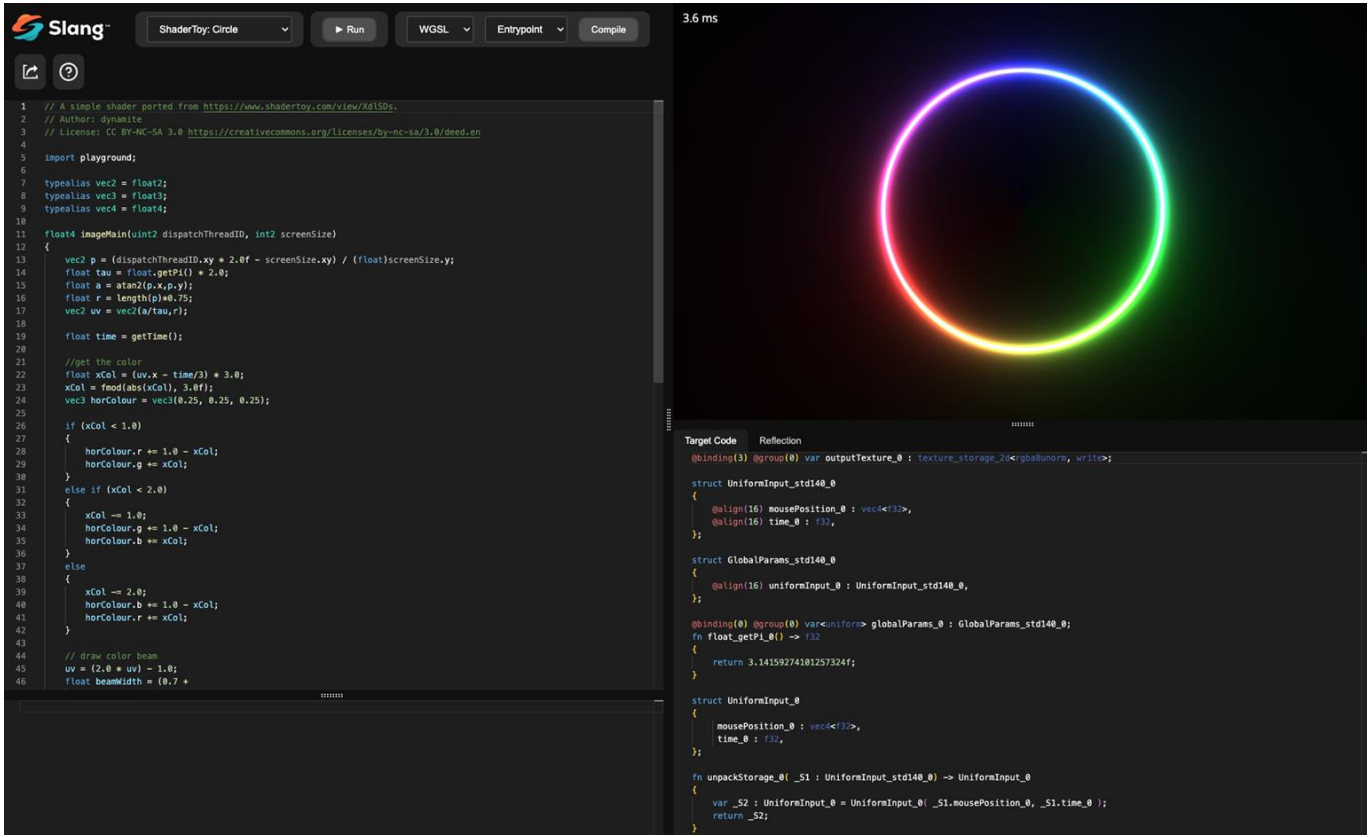
- **Proper solve for modularity, permutations, compile time explosion, and "string pasting"**
  - Drawing on advances from the broader language space, Slang addresses these issues with modules, generics, and interfaces

Seamless Cross-Compilation in Slang

- No need to chain together multiple libraries
- Tooling just works
- Produces human-readable code



New in Slang: WGSL Support for WebGPU



New in Slang: Metal Support

- **Vertex, fragment, compute, mesh, and amplification shaders supported**
  - No ray tracing yet
- **Automatic transformations performed for Metal legalization documented here**
  - Enables Slang to support some functionality even though a Metal equivalent doesn't exist. E.g. combined samplers, pointer to vector element
- **Debug & tooling compatibility w/ #line directives**

```
#line 11 "/user.slang"
float4 imageMain_0(uint2 dispatchThreadID_0, int2 screenSize_0, KernelContext_0 thread* kernelContext_1)
{
    float2 p_0 = (float2(dispatchThreadID_0.xy) * 2.0 - float2(screenSize_0.xy)) / float(screenSize_0.y);
    float tau_0 = float_getPi_0() * 2.0;


    float _S1 = atan2(p_0.x, p_0.y) / tau_0;

#line 17
    float2 uv_0 = float2(_S1, length(p_0) * 0.75);

#line 17
    float _S2 = getTime_0(kernelContext_1);

#line 23
    float _S3 = abs((_S1 - _S2 / 3.0) * 3.0);

#line 23
    float xCol_0 = ((((_S3) < 0.0) ? -fmod(-(_S3),abs((3.0))) : fmod((_S3),abs((3.0)))));
    thread float3 horColour_0 = float3(0.25, 0.25, 0.25);

    if(xCol_0 < 1.0)
    {
        horColour_0.x = horColour_0.x + (1.0 - xCol_0);
        horColour_0.y = horColour_0.y + xCol_0;

#line 26
    }
    else
    {


        if(xCol_0 < 2.0)
        {
            float xCol_1 = xCol_0 - 1.0;
            horColour_0.y = horColour_0.y + (1.0 - xCol_1);
            horColour_0.z = horColour_0.z + xCol_1;
```

Slang Tooling

You can already use Slang with existing toolchains

- Step-through debugging in Renderdoc
- Shader inspection in Nsight
- Slang in Vulkan SDK 1.3.296.0 and above

Step-through debugging in RenderDoc

Amazing autocomplete support

- Extensions for Visual Studio & VSCode provide IntelliSense support
- Language server module available for integration into other IDEs
- No other shading language offers something this cool

IntelliSense / Language Server support in action

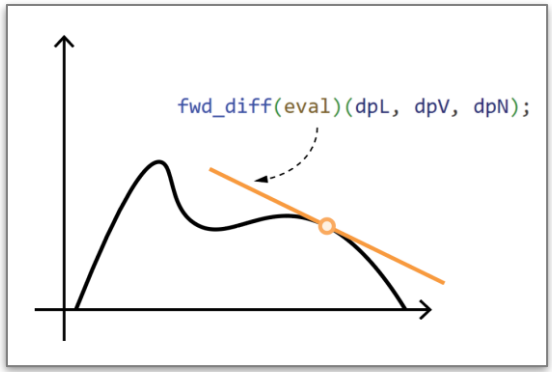
Easily write & maintain differentiable code

- **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)
```



Use Case: Gaussian Splatting

- **Slang implementation of the rasterizer from [3D Gaussian Splatting for Real-Time Rendering of Radiance Fields]**
  - Kernel performance equivalent to hand-written CUDA

Slang.D Gaussian Splatting Rasterizer

This repository contains a Slang.D implementation of the CUDA accelerated rasterizer that is described in the 3D Gaussian Splatting for Real-Time Rendering of Radiance Fields. We provide only the rasterizer and API bindings to the most popular implementations of the complete 3D Gaussian Splatting implementations:

1. The original inria code-base and
2. gsplat from nerf-studio.

But why Slang.D?

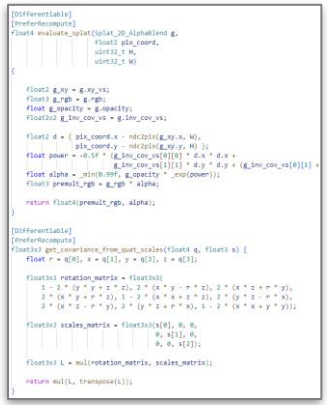
Slang.D is a *unified platform for real-time, inverse and differentiable rendering*. Slang serves as an open-source language that allows developers to maintain a single code-base for differentiable high-performance rendering code-bases which can compile down to rendering code for different platforms like D3D, Vulkan, OpenGL, OptiX, CUDA etc.

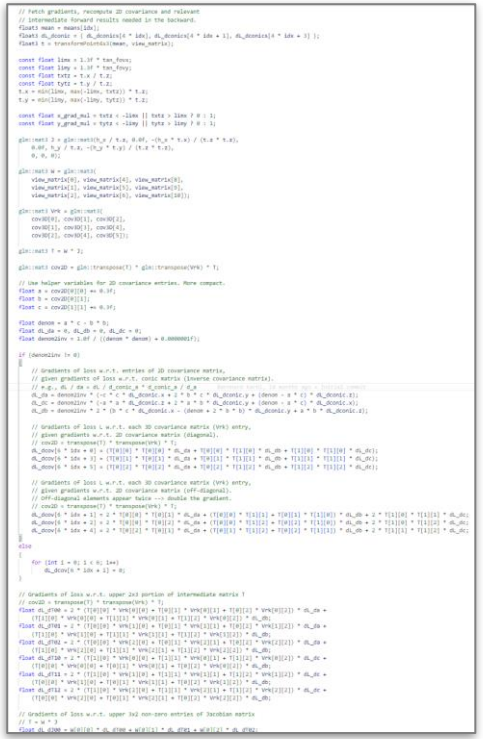
This allows for a **single code-base** of the rendering code which has the capability to run at the same time under the differentiable framework of your choice (i.e Pytorch) and under the actual renderer that can be released for production (i.e Vulkan). This makes significant strides toward maintainability and reduces the likelihood of errors.

On top of that and potentially the most important for *research* is the ability of Slang.D to **differentiate automatically** through complicated rendering kernels that include *arbitrary control flow, user-defined types, dynamic dispatch, generics, and global memory accesses.*

This code is authored by: George Kopanas

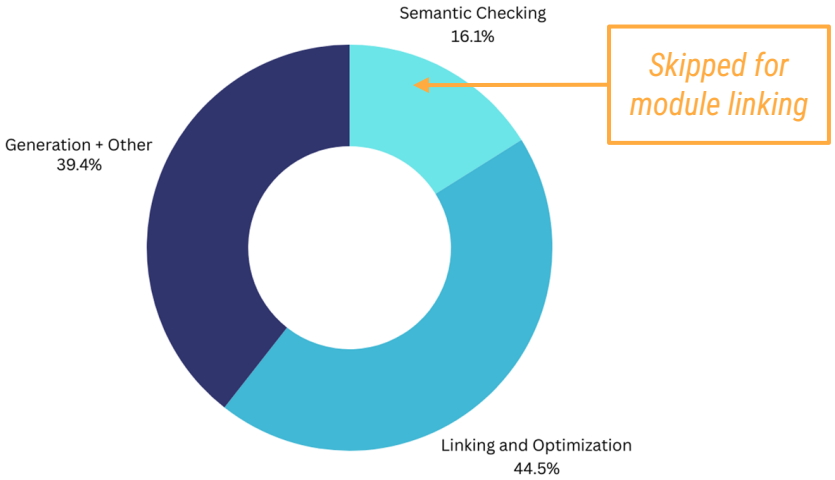
github.com/google/slang-gaussian-rasterization





1495 lines (Slang + Slang-torch version) vs.
2414 lines (CUDA version)

Modules + Interfaces + Generics = Faster Compiles



- **Compilation time**
  - Within 10% of glslc/dxc compilation times with monolithic code
  - Modularized code can reduce time spent in front-end compilation

Modules

Provide separation of compilation and control over visibility

material.slang

material.slang > Material > somePrivateMethod

```
1   module material;
2
3   public struct Material
4   {
5       public float4 evalBRDF(float3 wi, float3 wo)
6       {
7           // ...
8       }
9       internal float4 somePrivateMethod()
10      {
11          // ...
12      }
13  }
14
```

scene.slang 1

scene.slang > Scene > compute

```
1   module scene;
2
3   import material;
4
5   struct Scene
6   {
7       StructuredBuffer<Material> materials;
8
9       void compute(float3 wi, float3 wo)
10      {
11          float4 result = materials[0].evalBRDF(wi, wo);
12          materials[0].somePrivateMethod();
13      }
14  }
15
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

scene.slang 1

'somePrivateMethod' is not accessible from the current context. (30600) [Ln 12, Col 22]

Generics & Interfaces

Generics improve code maintainability

Allows Intellisense to provide accurate assistance

Faster front-end compilation time from reusing type checking results

Interfaces make requirements explicit

Similar to Rust traits, Swift protocols, Haskell typeclasses …

```
1  interface IMaterial
2  {
3      associatedtype BRDF : IBRDF;
4      BRDF sampleAt(SurfacePoint p);
5  }
6
7  interface IBRDF
8  {
9      float4 evaluate(float3 lightDir, float3 eyeDir);
10 }
11
12 interface IGeometry { /*...*/ }
13 interface ILighting { /*...*/ }
```

Switching to Slang isn't Hard!

VALVE source 2

- Valve migrated entire Source 2 HLSL codebase
- Slang in use in production
- Minimal changes (~10 lines) needed to compile existing shaders with Slang

AUTODESK

- Slang is used by Aurora path tracing renderer, enables single-source ray tracing codebase
- Ray tracing support just worked!
- Slang shaders are open source & available to check out

- **Binary Size**: 8mb uncompressed
  - Wasm implementation compressed to 5MB
  - No LLVM - we generate C++
  - Includes all backends!

- **Runtime performance**
  - Meet or beat handwritten code
  - Even when using advanced features such as generics

Templates port to generics

```
template<typename T> T selectValue(float inVal, T v0, T v1)
{
    if (inVal <= 1.0)
        return v0;
    else
        return v1;
}
```

```
__generic<typename T> T selectValue(float inVal, T v0, T v1)
{
    if (inVal <= 1.0)
        return v0;
    else
        return v1;
}
```

Some template code can be ported trivially to generics by replacing “template” with “__generic”

Generics are templates, but better…

```
template<typename T>
int compute(T v)
{
    return v.eval();
}
```

This is valid C++ but invalid Slang code.

The compiler cannot prove that `T` has the `eval` method.

```
interface IEvaluable
{
    int eval();
}
__generic<typename T>
int compute(T v) where T:IEvaluable
{
    return v.eval();
}
```

The solve is telling Slang your type constraints. This easily ifdefs:

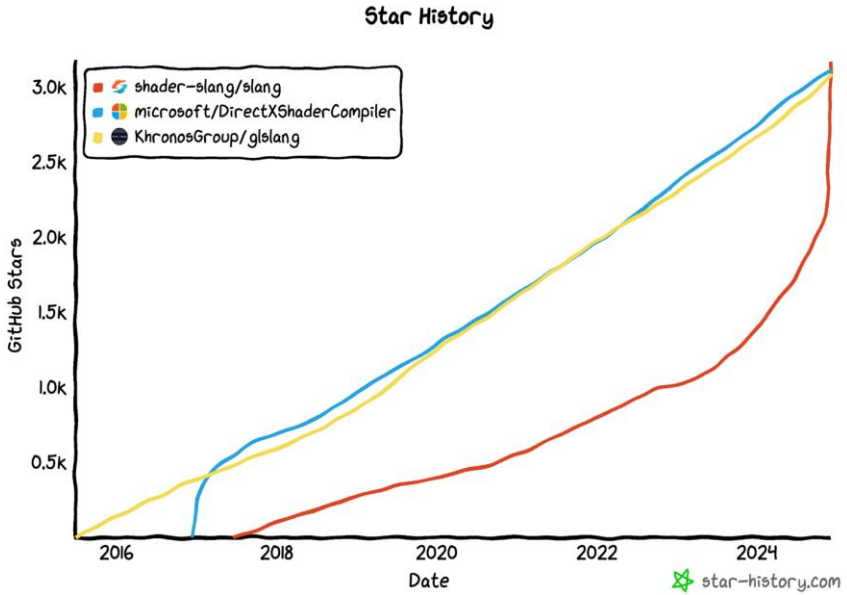
```
#ifdef __slang
#    define WHERE(x) where x
#    define template __generic
#else
#    define WHERE(x)
#endif
```

Try Slang in your Browser!

https://try.shader-slang.org/

How you can get involved today

- Join the Discord!
  - 180 members and counting!
- File an issue or feature request
- Start a GitHub discussion
- Submit a pull request
- Become a committer!



KHRONOS GROUP

Resources

- **Slang resources**
  - https://shader-slang.org/
- **Open-source Slang Repo**
  - Accepting design proposal RFCs, Pull Requests, and Bug Reports
  - https://github.com/shader-slang
- **Discord Discussion Channels**
  - https://khr.io/slangdiscord
- **Playground - try Slang in your browser**
  - https://try.shader-slang.org/

Khronos BOFs at SIGGRAPH Asia

Day	Time / Room	Session Title	Standards and Projects
Tuesday 3rd	1:00-2:00PM, G408	Khronos Fast Forward	Vulkan, OpenXR, Slang, ANARI, glTF
Wednesday 4th	1:00-2:00PM, G407	Slang Shading Language	Slang
Wednesday 4th	3:30-4:30PM, G407	Immersive Web with Khronos and W3C	WebGL, WebXR, WebGPU, three.js
Thursday 5th	2:15-3:15PM, G407	OpenXR Update and Roadmap	OpenXR
Thursday 5th	3:30-5:30PM, G407	Vulkan Update and Ecosystem	Vulkan, Vulkan SC, Slang
Friday 6th	1:00-2:00PM, G408	glTF 3D Transmission Format	glTF, VRM Avatar Format

All BOF slides and videos will be uploaded to the
Khronos SIGGRAPH event page

SIGGRAPH ASIA 東京 2024 TOKYO 3–6 DEC

Khronos BOFs

KHRONOS GROUP

Khronos Information
www.khronos.org
memberservices@khronosgroup.org