Metal Compute Shaders And C++

What is Metal?

- Shading language released by Apple in 2014
- Provides Graphics / Compute capabilities
- Most recent release in 2022 (v3.0)
- Not Cross-Platform

Why Metal?

- Apple silicon processors have capable GPUs
  - "APU" style architecture
  - Large amount of memory

Why Metal?

- Apple silicon processors have capable GPUs
  - "APU" style architecture
  - Large amount of memory
- Apple is killing off their other GPGPU options

Why Metal?

Transition to Metal

If you're using OpenCL, which was deprecated in macOS 10.14, for GPU-based computational tasks in your Mac app, we recommend
Metal Performance Shaders for access to a w

Learn about Metal ›

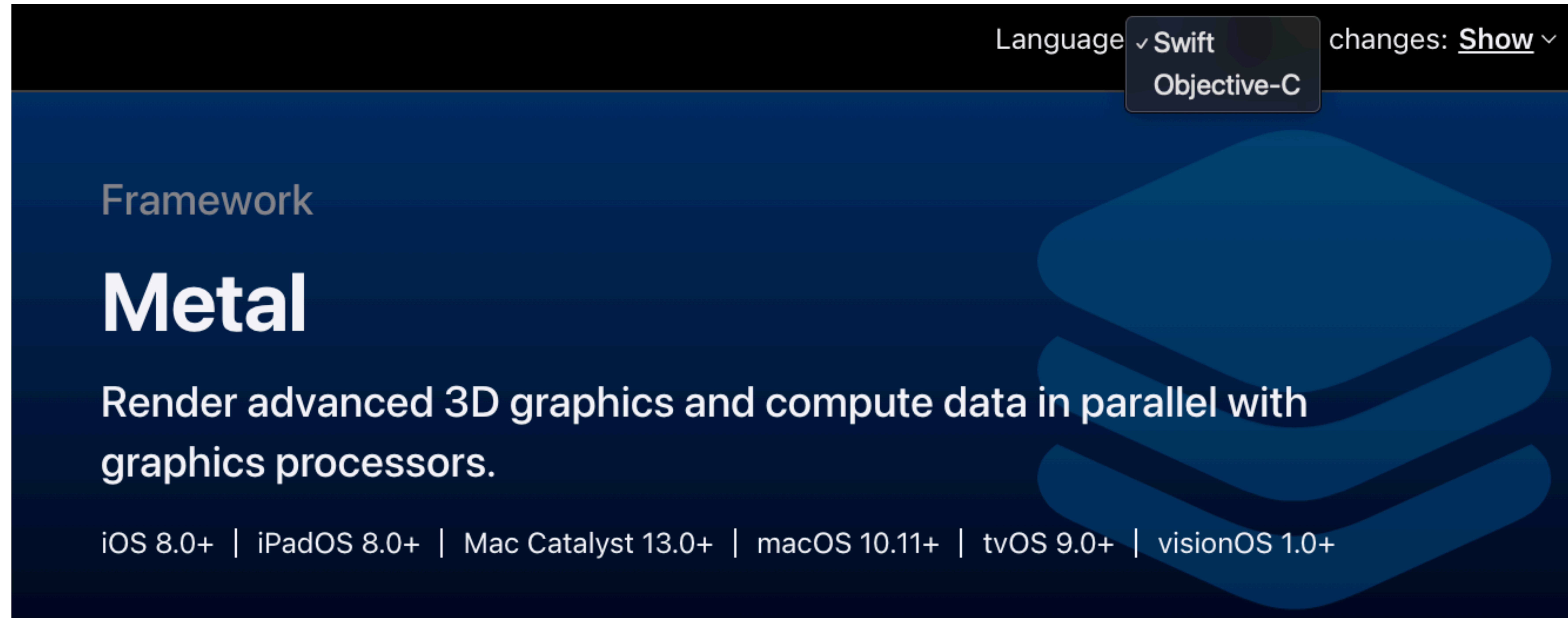
Developer

OpenGL Programming Guide for Mac

Retired Document

Important: OpenGL was deprecated in macOS 10.14. To create high-performance code on GPUs, use the Metal framework instead. See Metal.

Recent Developments



Until recently, Metal was primarily aimed at Swift and Objective-C

Recent Developments

- In 2022 Apple released "Metal-cpp"
  - some headers that provides a way to interact with Metal from C++
  - they provide a few examples of using Metal with C++, but ...
  - those examples are almost exclusively the **graphics** pipeline

Recent Developments

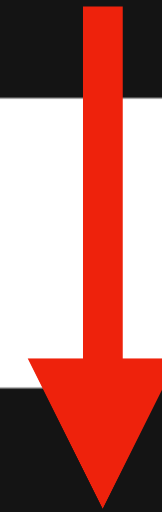
- In 2022 Apple released "Metal-cpp"
  - some headers that provides a way to interact with Metal from C++
  - they provide a few examples of using Metal with C++, but ...
  - those examples are almost exclusively the **graphics** pipeline

- This weekend, I bit the bullet and tried porting some Metal-Objective-C compute examples to Metal-C++

Initial Port

Objective C

```
[computeEncoder setComputePipelineState:_mAddFunctionPSO];
[computeEncoder setBuffer:_mBufferA offset:0 atIndex:0];
[computeEncoder setBuffer:_mBufferB offset:0 atIndex:1];
[computeEncoder setBuffer:_mBufferResult offset:0 atIndex:2];
```



C++

```
computeEncoder->setComputePipelineState(_mAddFunctionPSO);
computeEncoder->setBuffer(_mBufferA, 0, 0);
computeEncoder->setBuffer(_mBufferB, 0, 1);
computeEncoder->setBuffer(_mBufferResult, 0, 2);
```

Metal-cpp made porting easier than expected

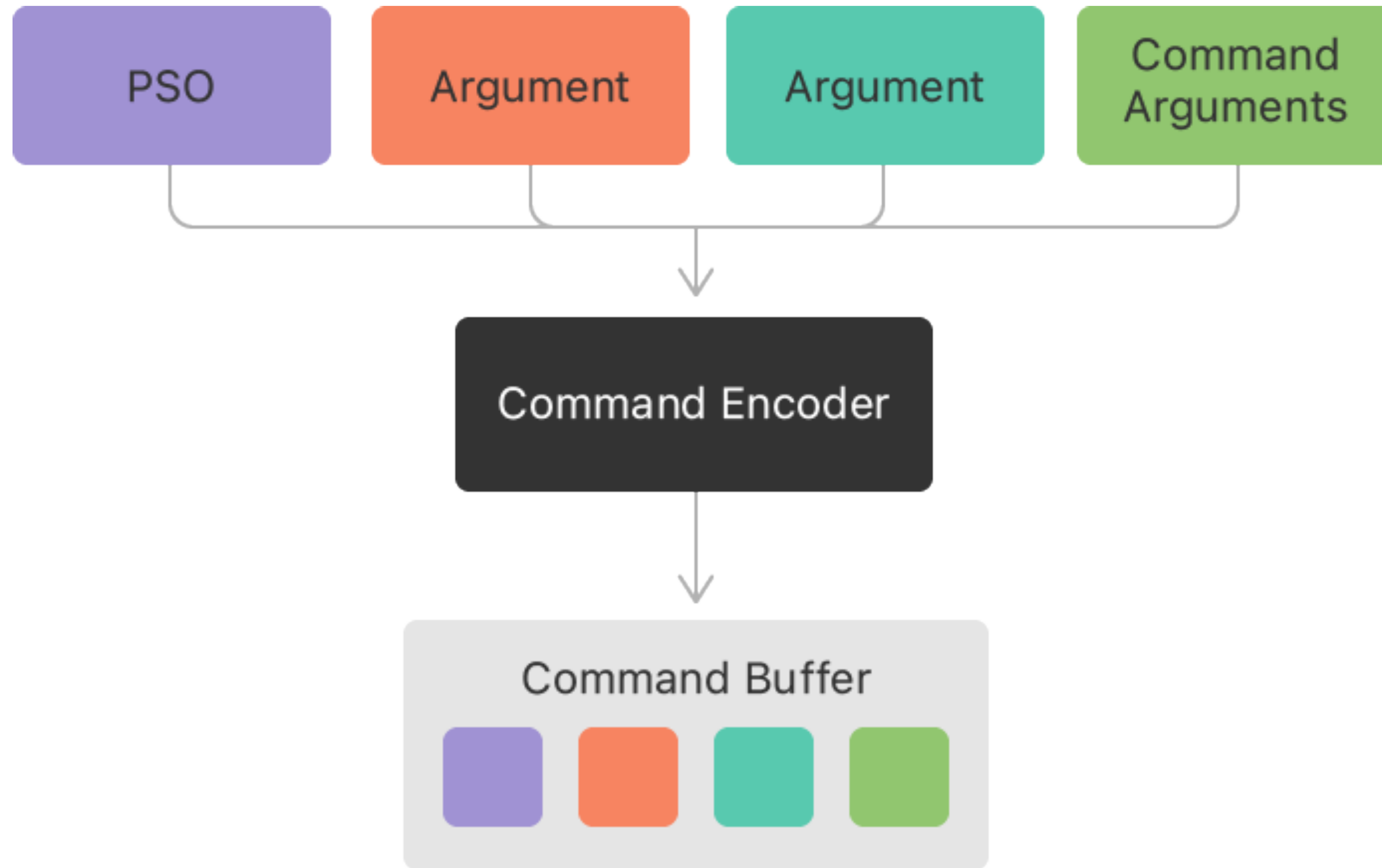
Example Code Walkthrough 1:

https://github.com/samuelpmish/metal_cpp_compute/blob/main/examples/saxpby.cpp

Metal / CUDA Cheatsheet

Metal	CUDA / C++
kernel	__global__
threadgroup (attribute)	__shared__
grid*	grid*
threadgroup (launch parameter)	block
device	device
library	translation unit
CommandQueue	stream
CommandBuffer/ComputeCommandEncoder/ ComputePipelineState	<<< ... >>>
Buffer (container)	cudaMalloc

Metal Kernel Launch



Example Code Walkthrough 2:

https://www.smish.dev/math/mesh_relaxation/

https://github.com/samuelpmish/metal_cpp_compute/blob/main/examples/vertex_relaxation.cpp

Example Code Walkthrough 2:

https://www.smish.dev/math/mesh_relaxation/

https://github.com/samuelpmish/metal_cpp_compute/blob/main/examples/vertex_relaxation.cpp

```
sam@mozzarella examples % ./vertex_relaxation
vertex relaxation time (CPU,  1 thread)  776.898ms
vertex relaxation time (CPU, 14 threads) 338.543ms
vertex relaxation time (GPU)              21.7406ms
```

Writing a Metal Wrapper

Goals:

- Try to eliminate boilerplate
- Build in as much type-safety (as practical)
- Make it "feel" like CUDA runtime-API
- Automate lifetime management

Writing a Metal Wrapper

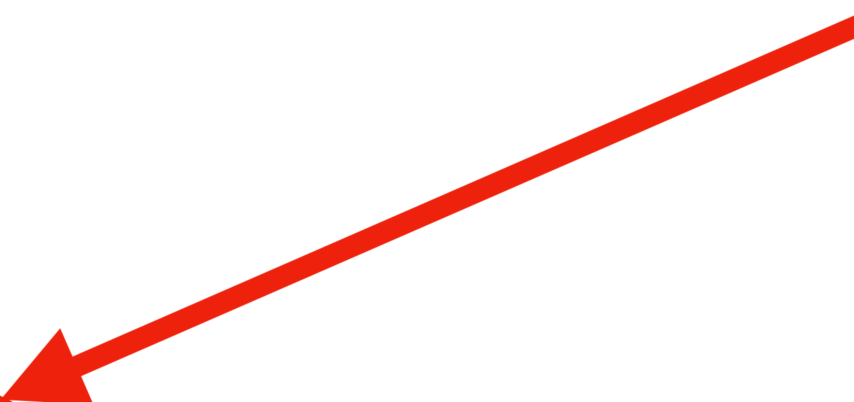
https://github.com/samuelpmish/metal_cpp_compute/blob/main/examples/saxpby_with_wrapper.cpp

Summary

- Easier than I expected
- The good:
    - Performance
    - Programming model
- Some significant downsides
    - boilerplate, type safety
    - no double precision support
    - portability

Summary

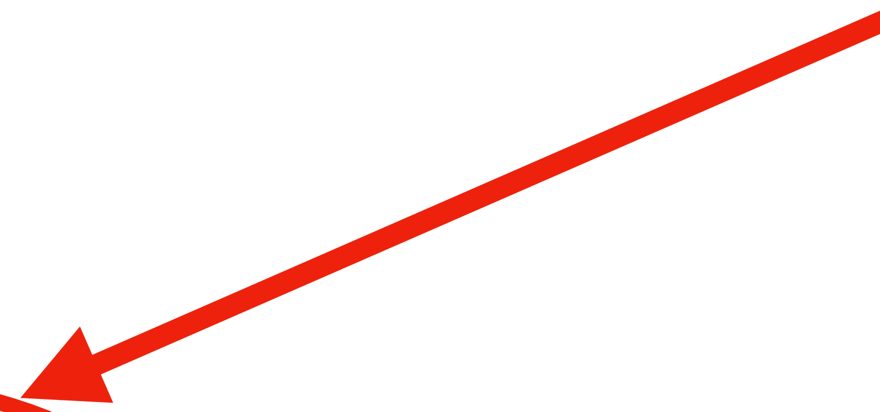
- Easier than I expected
- The good:
  - Performance
  - Programming model
- Some significant downsides
  - boilerplate, type safety
  - no double precision support
  - portability



can write wrappers
to help address

Summary

- Easier than I expected
- The good:
  - Performance
  - Programming model
- Some significant downsides
  - boilerplate, type safety
  - no double precision support
  - portability



can make algorithmic changes
(e.g. compensated summation)
to help address

Summary

- Easier than I expected
- The good:
  - Performance
  - Programming model
- Some significant downsides
  - boilerplate, type safety
  - no double precision support
  - portability

How to handle?

slang

- shading (meta)language
  - slang transpiles to C++/CUDA/Metal/WGPU/Vulkan/D3D
  - supports cool language features:
    - full compute / graphics (raster and ray tracing) capabilities
    - static reflection
    - statically analyzable automatic differentiation (fwd/bwd)

https://shader-slang.com/slang-playground/