Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

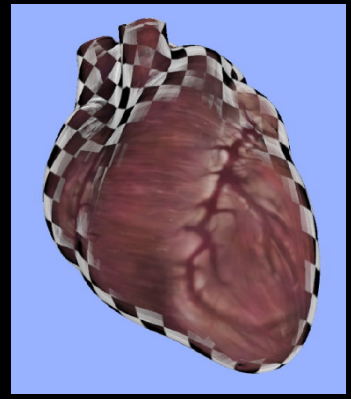
Transitioning to Vulkan for Compute

Bernhard Kerbl,

INRIA, Université Côte d'Azur

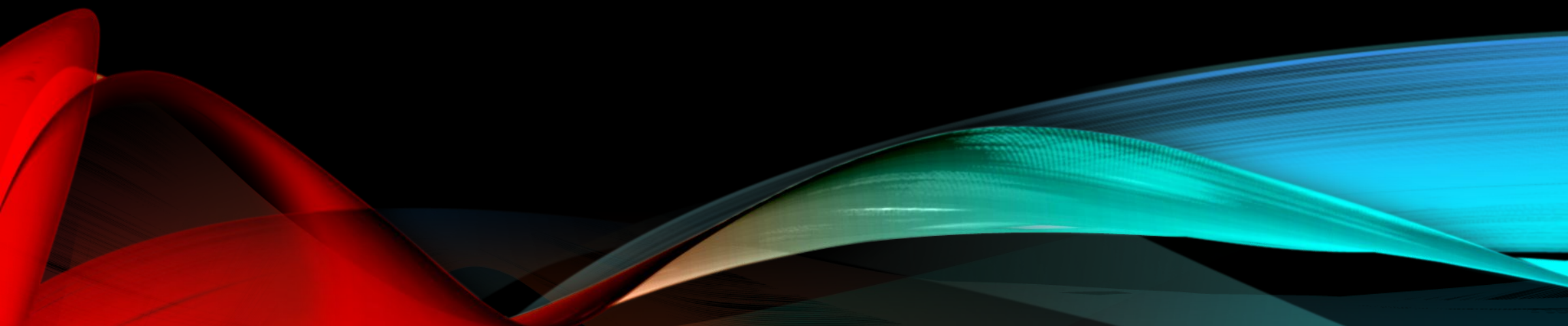
ABOUT THIS TALK

- A case for approaching Vulkan via Compute to curb the learning curve
- Ideas, experiences and advancements with Vulkan as a compute API
- Who‘s presenting it:
  - **2014 – 2018**, PhD at Graz University of Technology, „*Load Balancing for Hardware and Software Rendering on the GPU*“
  - **2019**, Epic Games intern during *Nanite* development
  - **2019 – 2022**, Research at TU Wien
  - **2022 – now**, Research at INRIA, Université Côte d‘Azur
  - **2016 – now**, Teaching (GP)GPU Programming with OpenGL/CUDA/**Vulkan** at Austrian universities

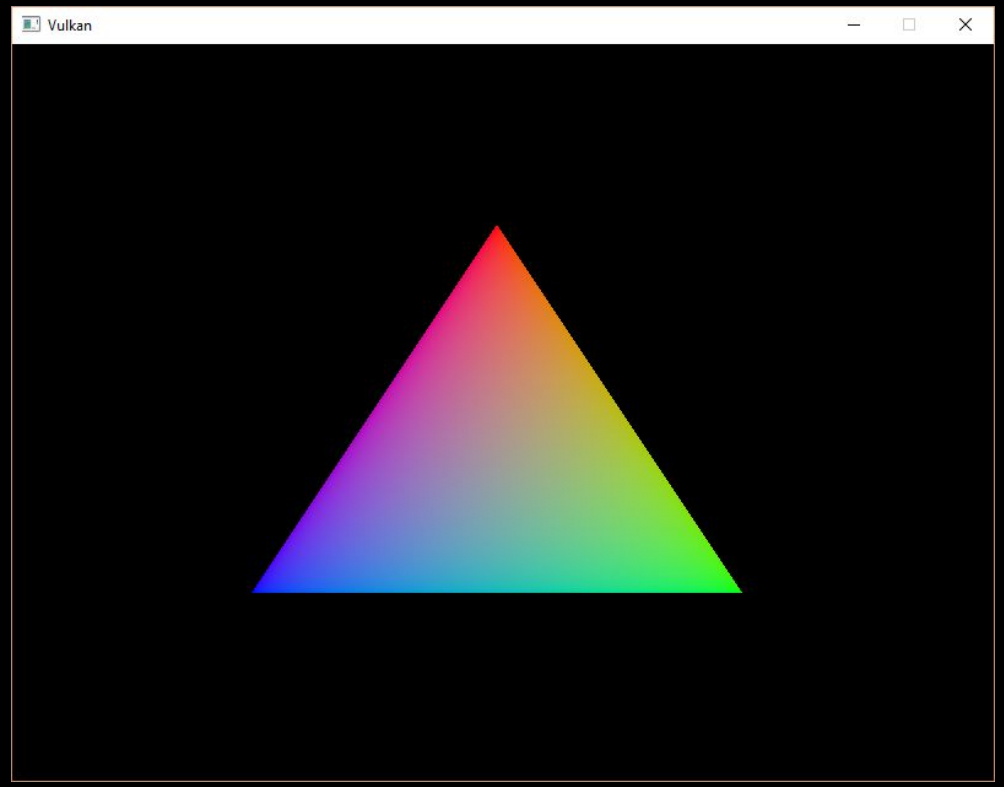


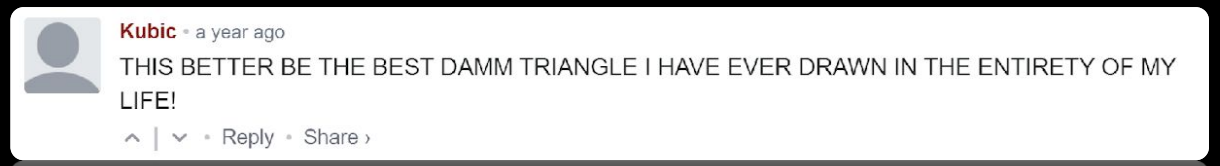
LEARNING AND TEACHING VULKAN VIA COMPUTE

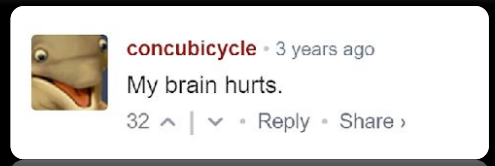
Benefits and Roadmap

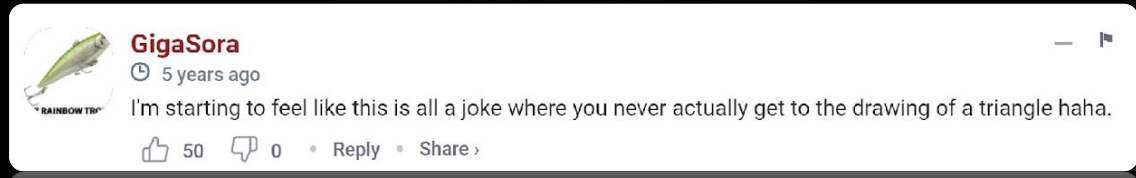


„HELLO TRIANGLE“
WITH VULKAN FOR GRAPHICS









SIMPLIFYING WITHOUT COMPROMISE

1. **Start with compute**
   - Parallel GPGPU compute jobs have wider applicability than "just" graphics
   - Significantly reduced setup!

2. **Embrace the SDK's provided `vulkan.hpp`**
   - C++-style API (RAII, default constructors, …) without compromising Vulkan's versatility
   - Reduces verbosity by a lot, cleaner and easy-to-read code

3. **Exploit Vulkan's tools for error checking and validation on-demand**
   - *Vulkan Configurator* to replace validation layer setup
   - *RenderDoc* to fix hard-to-debug errors

4. **Pay attention to quality-of-life improvements and API features**
   - E.g., shader debug `printf` is in core since API 1.3, works without extensions

```cpp
int main() {
    const vk::ApplicationInfo applicationInfo("Hello World", 0, nullptr, 0, VK_API_VERSION_1_3);
    const auto instance = vk::createInstanceUnique(vk::InstanceCreateInfo({}, &applicationInfo));
    const auto physicalDevice = instance->enumeratePhysicalDevices()[0];

    int family;
    const auto qProps = physicalDevice.getQueueFamilyProperties();
    for (family = 0; !(qProps[family].queueFlags & vk::QueueFlagBits::eCompute) && family < qProps.size(); family++);

    constexpr float priority[] = { 1.0f };
    const vk::DeviceQueueCreateInfo deviceQueueCreateInfo({}, family, 1, priority);
    const auto device = physicalDevice.createDeviceUnique(vk::DeviceCreateInfo({}, deviceQueueCreateInfo));

    const std::string print_shader = R"(
    #version 460
    #extension GL_EXT_debug_printf : require
    void main()
    { debugPrintfEXT("'Hello world!' (said thread: %d)\n", gl_GlobalInvocationID.x); })";

    const auto compiled = shaderc::Compiler().CompileGlslToSpv(print_shader, shaderc_compute_shader, "hello_world.comp");
    const std::vector<uint32_t> spirv(compiled.cbegin(), compiled.cend());
    const auto shaderModule = device->createShaderModuleUnique(vk::ShaderModuleCreateInfo({}, spirv));
    const vk::PipelineShaderStageCreateInfo stageCreateInfo({}, vk::ShaderStageFlagBits::eCompute, *shaderModule, "main");
    const auto pipelineLayout = device->createPipelineLayoutUnique(vk::PipelineLayoutCreateInfo({}));
    const vk::ComputePipelineCreateInfo pipelineCreateInfo({}, stageCreateInfo, *pipelineLayout);
    const auto [status, pipeline] = device->createComputePipelineUnique(*device->createPipelineCacheUnique({}), pipelineCreateInfo);

    const auto pool = device->createCommandPoolUnique(vk::CommandPoolCreateInfo({}, family));
    const vk::CommandBufferAllocateInfo allocateInfo(*pool, vk::CommandBufferLevel::ePrimary, 1);
    const auto cmdBuffers = device->allocateCommandBuffersUnique(allocateInfo);
    cmdBuffers[0]->begin(vk::CommandBufferBeginInfo{});
    cmdBuffers[0]->bindPipeline(vk::PipelineBindPoint::eCompute, *pipeline);
    cmdBuffers[0]->dispatch(8, 1, 1);
    cmdBuffers[0]->end();
    device->getQueue(family, 0).submit(vk::SubmitInfo({}, {}, *cmdBuffers[0]));
    device->waitIdle();
    return 0;
}
```

Vulkan Setup

Shader

Dispatch

```cpp
int main() {
    const vk::ApplicationInfo applicationInfo("Hello World", 0, nullptr, 0, VK_API_VERSION_1_3);
    const auto instance = vk::createInstanceUnique(vk::InstanceCreateInfo({}, &applicationInfo));
    const auto physicalDevice = instance->enumeratePhysicalDevices()[0];

    int family;
    const auto qProps = physicalDevice.getQueueFamilyProperties();
    for (family = 0; !(qProps[family].queueFlags & vk::QueueFlagBits::eCompute) && family < qProps.size(); family++);

    constexpr float priority[] = { 1.0f };
    const vk::DeviceQueueCreateInfo deviceQueueCreateInfo({}, family, 1, priority);
    const auto device = physicalDevice.createDeviceUnique(vk::DeviceCreateInfo({}, deviceQueueCreateInfo));

    const std::string print_shader = R"(
    #version 460
    #extension GL_EXT_debug_printf : require
    void main()
    { debugPrintfEXT("'Hello world!' (said thread

    const auto compiled = shaderc::Compiler().Com
    const std::vector<uint32_t> spirv(compiled.cb
    const auto shaderModule = device->createShade
    const vk::PipelineShaderStageCreateInfo stage
    const auto pipelineLayout = device->createPip
    const vk::ComputePipelineCreateInfo pipelineC
    const auto [status, pipeline] = device->creat

    const auto pool = device->createCommandPoolUn
    const vk::CommandBufferAllocateInfo allocateI
    const auto cmdBuffers = device->allocateComma
    cmdBuffers[0]->begin(vk::CommandBufferBeginIn
    cmdBuffers[0]->bindPipeline(vk::PipelineBindP
    cmdBuffers[0]->dispatch(8, 1, 1);
    cmdBuffers[0]->end();
    device->getQueue(family, 0).submit(vk::Submit
    device->waitIdle();
    return 0;
}
```

Vulkan Setup

Microsoft Visual Studio Debug Console

```
'Hello world!' (said thread: 4)
'Hello world!' (said thread: 1)
'Hello world!' (said thread: 7)
'Hello world!' (said thread: 3)
'Hello world!' (said thread: 5)
'Hello world!' (said thread: 6)
'Hello world!' (said thread: 2)
'Hello world!' (said thread: 0)

C:\projects\gpusim-vulkan\build\01_HelloGPU\Debug\01_HelloGPU.exe (process 5604) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically
close the console when debugging stops.
Press any key to close this window . . .
```

er

VULKAN VIA COMPUTE – A ROADMAP

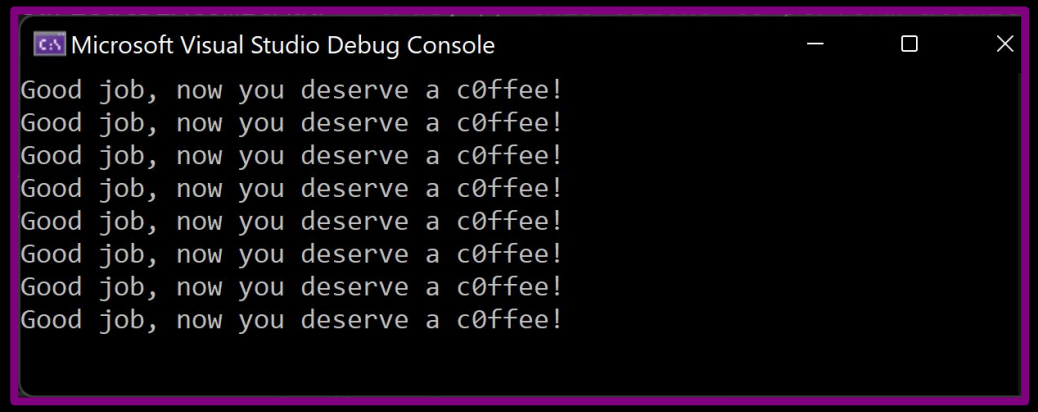
- "The Road to Vulkan" and "Vulkan all the Way" by J. Unterguggenberger

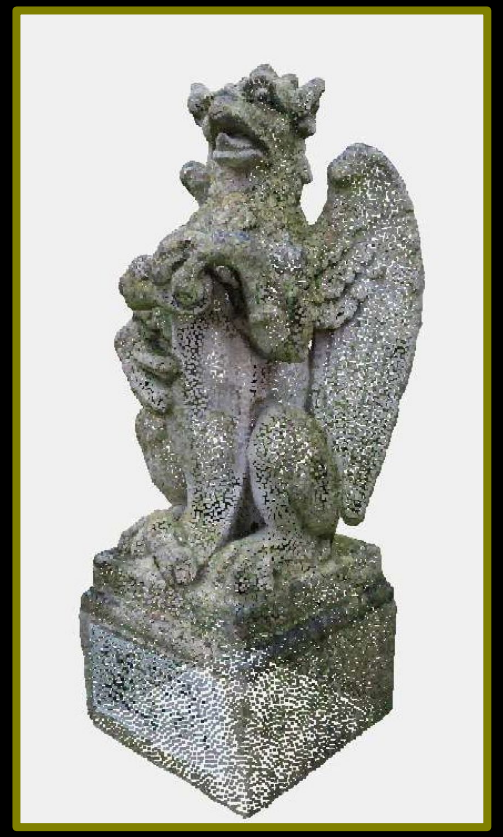
- In 2022, prepared undergrad course for GPGPU programming using Vulkan
  - 39 hours over the course of 4 months
  - 11 students with little-to-no experience with low-level GPU APIs

- A suite of 12 hands-on coding tasks, starting from zero
  - Supported by **`vulkan.hpp`**, *VkConfig* and *RenderDoc*
  - All remaining coding steps written by them
  - Small tasks (~30 min each), all with tangible results
  - Plan to extend and release in coming year, currently no curated recordings

LIST OF CODING TASKS

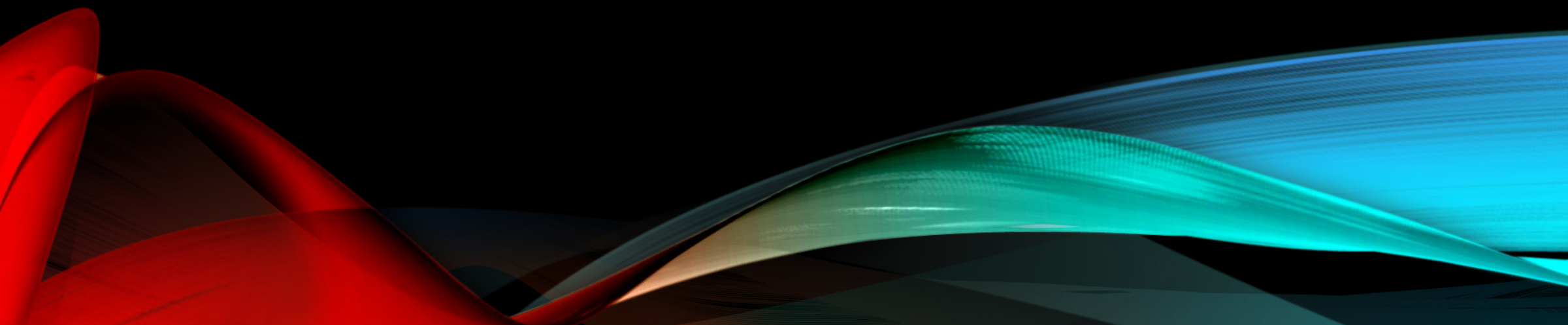
1. My Device(s)
2. Hello GPU
3. Copying
4. Uniform buffers
5. Storage buffers
6. Edge detector
7. Atomics
8. Point cloud renderer
9. Shared memory
10. Matrix Multiplication
11. Reduction
12. Final task of choice (Ray Tracer/Cloth Sim/MLP)





Stone Griffin, Downing College, Cambridge by Thomas Flynn, software-rendered. CC 4.0

TRANSITIONING TO VULKAN



WHY WOULD YOU TRANSITION?

- Reduce bloat
  - Use a single API to perform rasterization, ray tracing, AI and physics simulations
- Vulkan has outstanding portability, but maintains important features
  - Base profile allows writing OS-agnostic and vendor-agnostic code
  - Advanced features are queried and added via **extensions**
- More freedom to design your own workflow
- How hard is the transition from pure GPGPU APIs?
  - Often easier than you think

"THERE'S AN EXTENSION FOR THAT"

- Underneath, it's the always same hardware
  - Obvious to adepts, but relevant to novices
  - Hence the ability to use a feature is just a question of *exposure*
  - *Extensions* can be part of GLSL code or activated when compiling shaders

- Extensions expose capabilities outside of the common GPU feature set
  - Atomic floating-point arithmetic? `VK_EXT_shader_atomic_float`
  - Release/acquire atomics and barriers? `GL_KHR_memory_scope_semantics`
  - Subgroup reduction of half floats? `GL_EXT_shader_subgroup_extended_types`
  - Want to use tensor cores*? `GL_NV_cooperative_matrix`

* The specification does not explicitly mention 'tensor cores', but describes the identical interfaces and effects

MORE CHOICES THAN YOU EXPECT

- Any shader language is fine, as long as it can be compiled to SPIR-V
  - *GLSL, HLSL, Rust, Metal, WGSL…*
  - Need CPU/GPU-agnostic code? Try *Slang*!

```glsl
#version 460
#extension GL_EXT_debug_printf : require

void main()
{
    debugPrintfEXT("Hello from thread: %d\n",
    gl_GlobalInvocationID.x);
}
```

GLSL

```hlsl
void main(uint3 id : SV_DispatchThreadID)
{
    printf("Hello from thread: %d\n", id.x);
}
```

HLSL

MORE CHOICES THAN YOU EXPECT

- Any shader language is fine, as long as it can be compiled to SPIR-V
  - *GLSL, HLSL, Rust, Metal, WGSL…*
  - Need CPU/GPU-agnostic code? Try *Slang*!

```hlsl
RWStructuredBuffer<int> message;

[numthreads(32, 1, 1)]
void main(uint3 dispatchThreadID : SV_DispatchThreadID)
{
    uint n = dispatchThreadID.x;
    float A = (1.0 + sqrt(5.0)) / 2.0;
    float B = 1.0 - A;
    A = pow(A, n);
    B = pow(abs(B), n);
    if(n % 2 == 1)
        B = -B;

    int fib = int((A - B) / sqrt(5.0));
    message[n] = fib;
}
```

HLSL/Slang

```glsl
#version 460
layout (set = 0, binding = 0) buffer MessageBuffer
{
    int data[];
} message;

layout(local_size_x = 32) in;
void main()
{
    uint n = gl_GlobalInvocationID.x;
    float A = (1.0 + sqrt(5.0)) / 2.0;
    float B = 1.0 - A;
    A = pow(A, n);
    B = pow(abs(B), n);
    if(n % 2 == 1)
        B = -B;

    int fib = int((A - B) / sqrt(5.0));
    message.data[n] = fib;
}
```

GLSL

VULKAN WITHOUT *VULKAN*

- Need portability benefits but want to avoid the API at all costs?
  - libvc – Vulkan Compute for C++
  - Vulkan Kompute – Vulkan Compute Framework for advanced GPU processing
  - VUDA – provides a CUDA Runtime API interface
  - …and several others at https://github.com/vinjn/awesome-vulkan

libvc

```
DevicePool devicePool;
for (Device &dev : devicePool.getDevices()) {
    try {
            Buffer buffer(dev, 8 * 1024);
            buffer.fill(0);

            Program prog(dev, "comp.spv", {buffer});
            Arguments args(program, {buffer});
            CommandBuffer commands(dev, prog, args);
            …
```

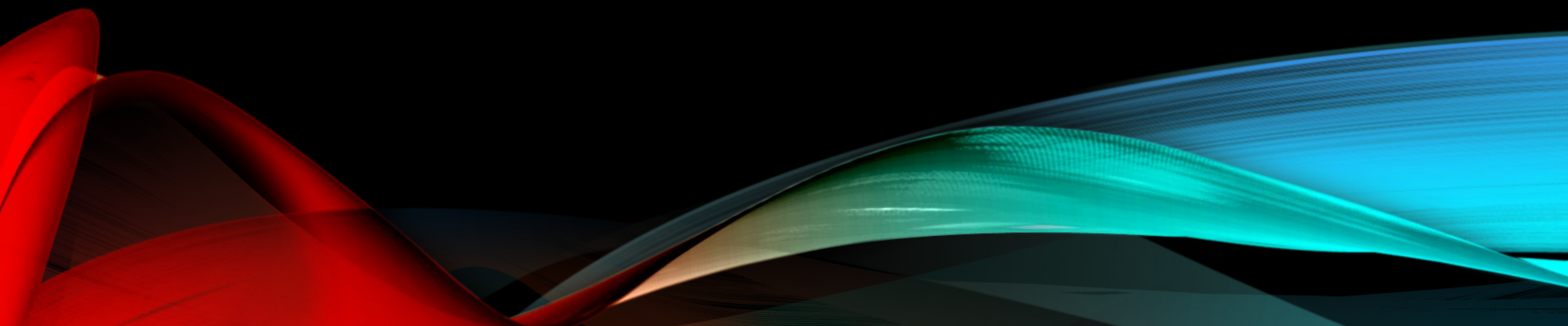
VUDA

```
#if defined(__NVCC__)
    #include <cuda_runtime.h>
#else
    #include <vuda_runtime.hpp>
#endif

int main(void)
{
    cudaSetDevice(0);
    …
```

CHALLENGES

For Developers and Teachers



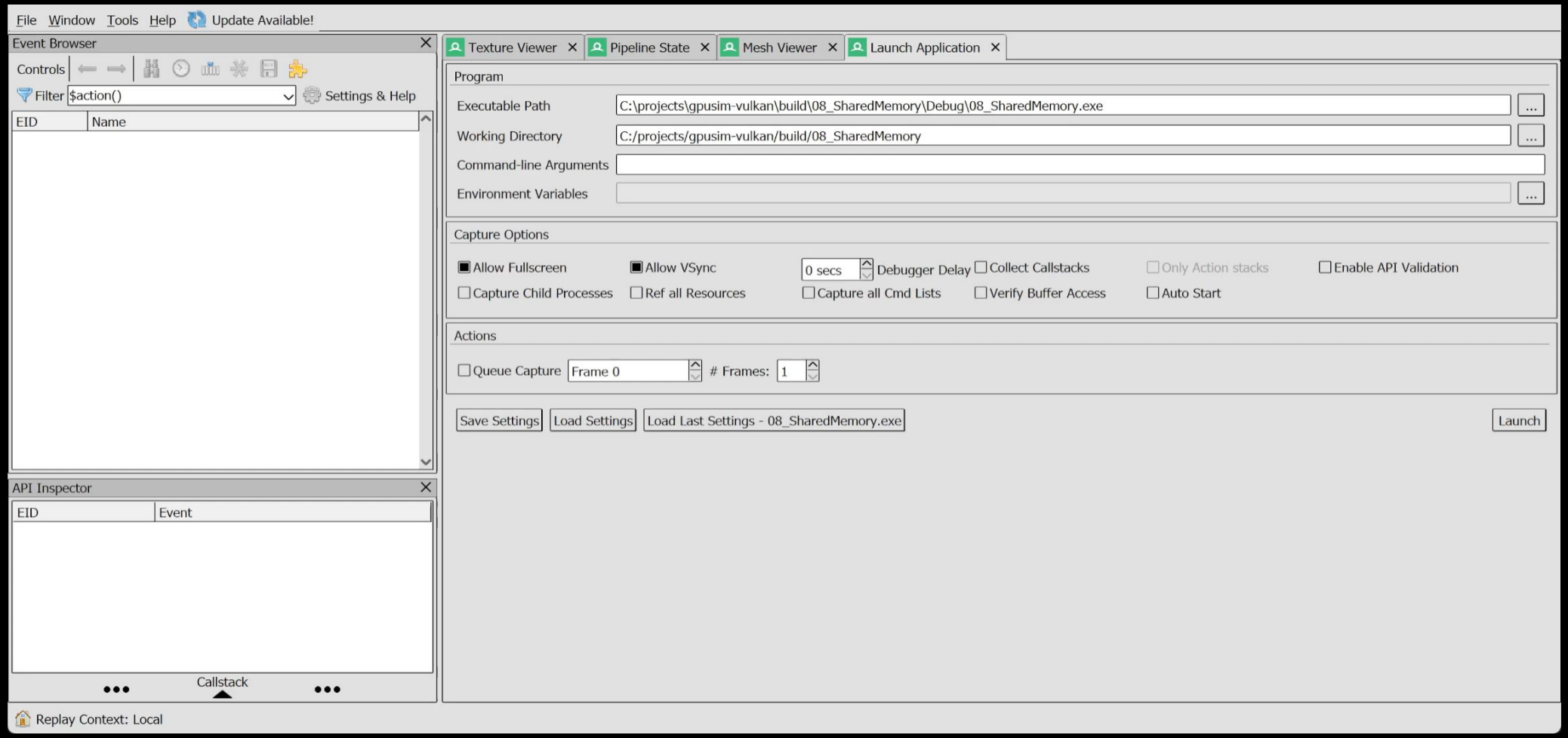
PROFILING AND DEBUGGING

- General-purpose compute shaders can quickly become complex; ensuring performance is often key to success

- Great profiling solutions with *RGP* and *Nsight Graphics*, **but limitations apply**
  - E.g., *Nsight Graphics* does not expose machine code instructions
  - Professional edition and agreement required

- We saw a great intro to *RenderDoc*'s debugging capabilities earlier today
  - Perfect for fixing arithmetic logic or data alignment errors
  - Not yet fully equivalent to the debuggers you have for *ROCm/CUDA*

SHARED MEMORY DEBUGGING



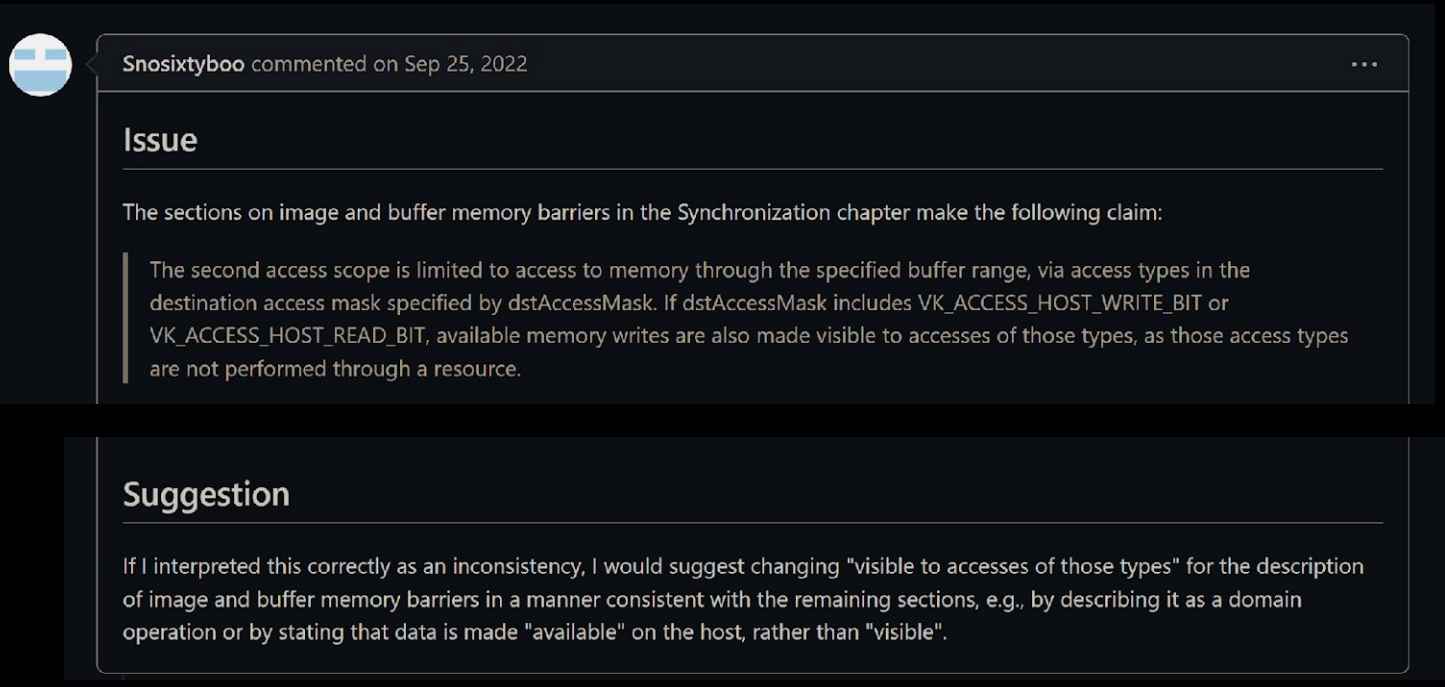
FEATURES AND LIBRARIES

- A complete, C++11-like memory consistency model (shared and global), **but** currently no forward progress guarantees!
  - In practice, however, many GPUs still seem to fulfill it
- Supports wide range of cutting-edge, vendor-specific features, **but** some popular ones missing (e.g., dynamic parallelism)
  - Available extensions often influenced by demand and vendor policies
- In a second, we will hear about *vkFFT* and its speed, which is fantastic news, **but** other APIs offer auxiliary libraries for containers, sorting, BLAS, AI…
  - An opportunity for the community to chip in!

THE DOCUMENTATION

- Full specification is 1000+ pages (intimidating!)
  - Is it just me or does the online version take 20s to load for everybody?
- **Must** be consulted at some point. Reader must be experienced at exploring lengthy documents for the bits they need.
- Mistakes do happen (they are being fixed post-haste!)
- Some statements can be correct, but the wording is unintuitive

EVOLVING VULKAN: BE THE CHANGE!



Snosixtyboo commented on Sep 25, 2022

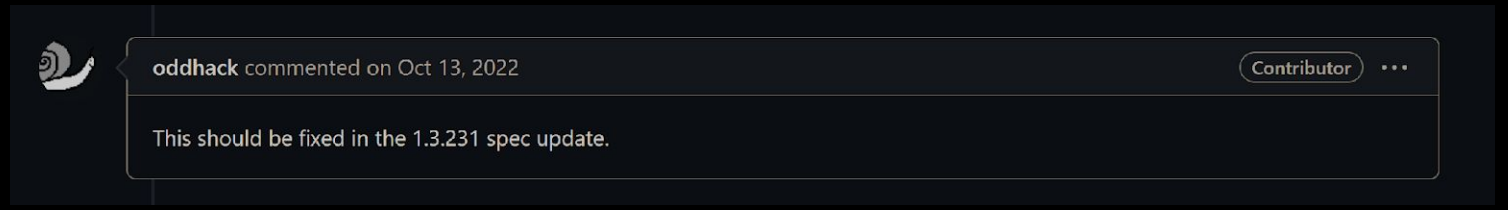
Issue

The sections on image and buffer memory barriers in the Synchronization chapter make the following claim:

> The second access scope is limited to access to memory through the specified buffer range, via access types in the destination access mask specified by dstAccessMask. If dstAccessMask includes VK_ACCESS_HOST_WRITE_BIT or VK_ACCESS_HOST_READ_BIT, available memory writes are also made visible to accesses of those types, as those access types are not performed through a resource.

Suggestion

If I interpreted this correctly as an inconsistency, I would suggest changing "visible to accesses of those types" for the description of image and buffer memory barriers in a manner consistent with the remaining sections, e.g., by describing it as a domain operation or by stating that data is made "available" on the host, rather than "visible".



oddhack commented on Oct 13, 2022 Contributor

This should be fixed in the 1.3.231 spec update.

CHALLENGES OF TEACHING VULKAN

- Synchronization
  - Improved with the arrival of `VK_KHR_synchronization2`
  - Many things to get wrong – potentially tough to verify!
  - Hot take: could we gamify synchronization?

- Descriptor sets
  - Exposing your resources to a shader: it feels like always one step too many
  - Again, things develop: we now have `VK_EXT_descriptor_buffer`

- Curbing mental load: if you are in a position to suggest courses, Vulkan via compute provides a gentler learning curve for mastering a modern GPU API

THANK YOU!

Questions?