Dissecting and fixing issues in VK driver with RenderDoc

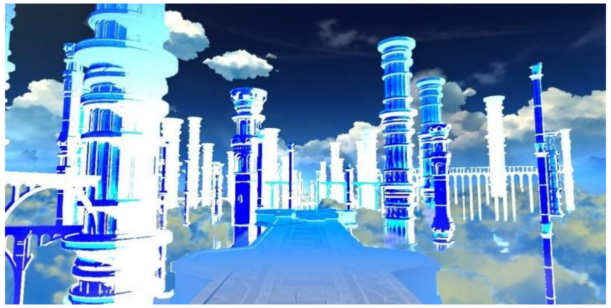
Danylo Piliaiev

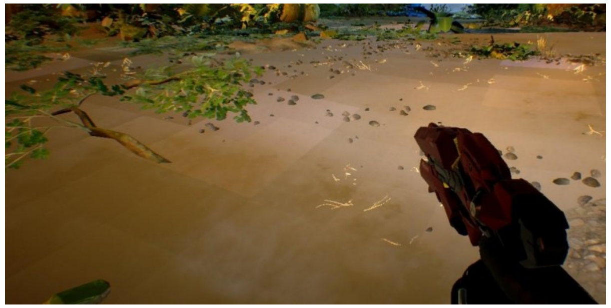
XDC 2021

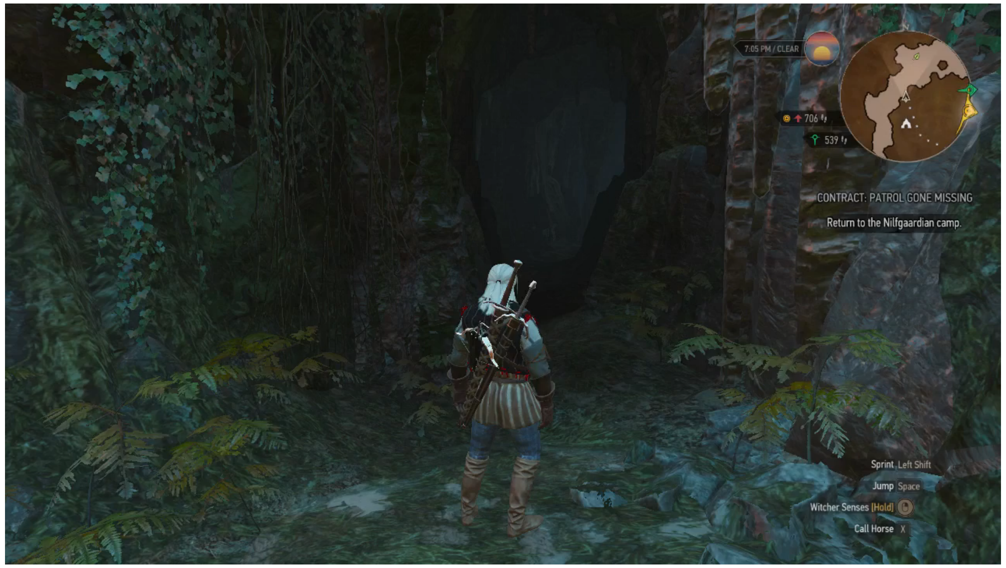
About me

- Worked on mobile video games
- Debugging unruly games since 2018
- At Igalia since November 2020
- Currently improving open source VK driver for Adreno - Turnip
- My blog: **blogs.igalia.com/dpiliaiev**

The problem






7:05 PM / CLEAR
706
539
CONTRACT: PATROL GONE MISSING
Return to the Nilfgaardian camp.
Sprint Left Shift
Jump Space
Witcher Senses [Hold]
Call Horse X

Categorization

- The biggest concern is a temporal stability of issues
  - Stable - easy to capture and easy to debug
  - Unstable - randomly appears and hard to capture
- Orthogonal to it are
  - Incorrect colors
  - Deformed/Missing geometry
  - Hangs
  - Crashes

Causes

- Undefined behavior of an application
- Shader mis-compilation (in NIR or in a backend)
- Missed/Wrong update of GPU state
- Hardware bug

Practical example

Video of practical example

Wrongly compiled fragment

```
_243 = clamp(_243, 0.0, 1.0);
_279 = clamp(_279, 0.0, 1.0);
float _290;
if (_72.x) {
  _290 = _279;
} else {
  _290 = _243;
}
color0 = vec4(_290);
```

Bad GPU assembly

```
mad.f32 r0.z, c0.y, r0.x, c6.w
sqrt r0.y, r0.y
mul.f r0.x, r1.y, c1.z
(ss)(nop2) mad.f32 r1.z, c6.x, r0.y, c6.y
(nop3) cmps.f.ge r0.y, r0.x, r1.w
(sat)(nop3) sel.b32 r0.w, r0.z, r0.y, r1.z
```

Good GPU assembly

```
(sat)mad.f32 r0.z, c0.y, r0.x, c6.w
sqrt r0.y, r0.y
(ss)(sat)mad.f32 r1.z, c6.x, r0.y, c6.y
(nop2) mul.f r0.y, r1.y, c1.z
add.f r0.x, r0.z, r1.z
(nop3) cmps.f.ge r0.w, r0.y, r1.w
cov.u r1.w, r0.w
(rpt2)nop
(nop3) add.f r0.w, r0.x, r1.w
```

What to do?

- Going line by line both assemblies look correct
- The only suspicious instruction is:

```
(sat)(nop3) sel.b32 r0.w, r0.z, r0.y, r1.z
```

- Could it be that `(sat)` modifier doesn't work?
- Manually edit the assembly to test it!

Editing GPU assembly

- Unfortunately we cannot edit the assembly through RenderDoc, there is no Vulkan extension for this =(
  - Could we make one?
- Some drivers including Turnip have the ability to replace shader assembly by its hash:
  - For Turnip/Freedreno - `IR3_SHADER_OVERRIDE_PATH`
  - For Intel's drivers - `INTEL_SHADER_ASM_READ_PATH`
  - For radeonsi - `RADEON_REPLACE_SHADERS`

Conclusion

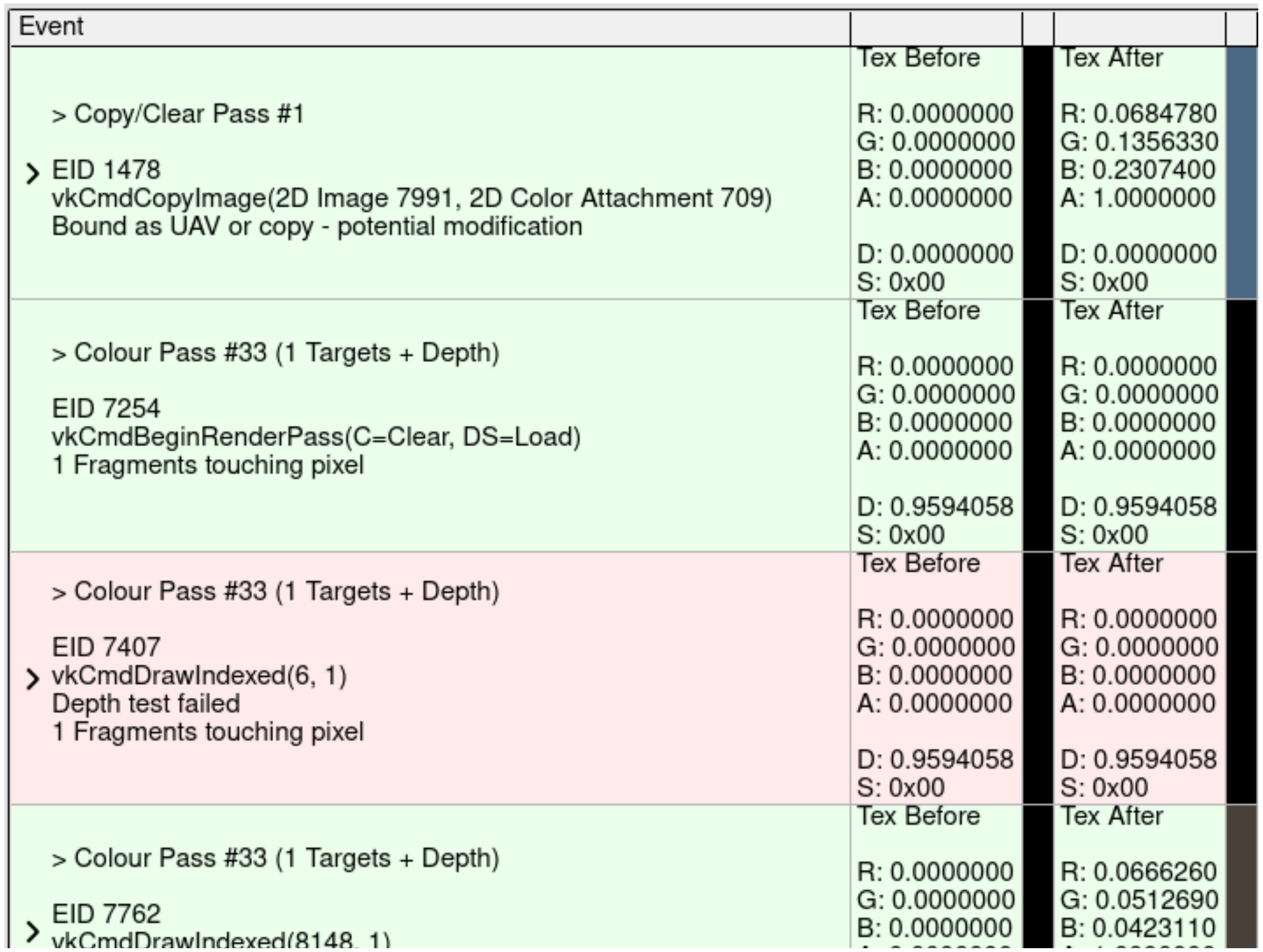
- Removing the saturation modifier doesn't change anything
- Moving it to the instructions which produced the arguments for the `sel` resolves the issue
- Verdict: saturation is not supported for `sel` instruction
- Fixed in **MR#9666 "ir3: disallow .sat on SEL instructions"**

Useful RenderDoc capabilities

- Viewing and being able to export any input in the pipeline
- All of the pipeline state is visible
- All varyings between shader stages are visible, thanks to the most cursed extension - `VK_EXT_transform_feedback`
- Software shader debugger
- Printf support in shaders
- Python API for automation and plugins!

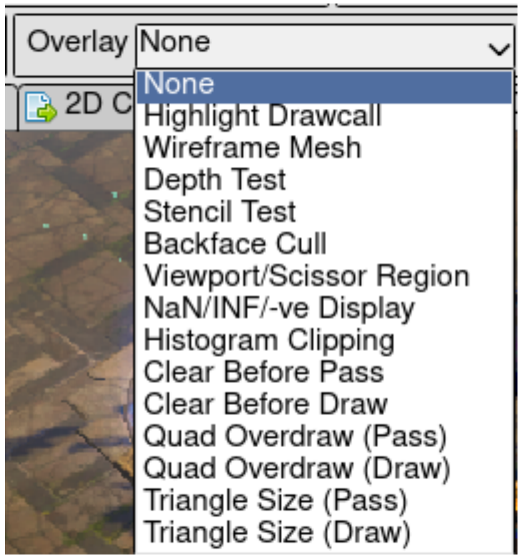
Pixel history

- Pixel history allows to quickly find which draw call wrote the fragment to the FBO

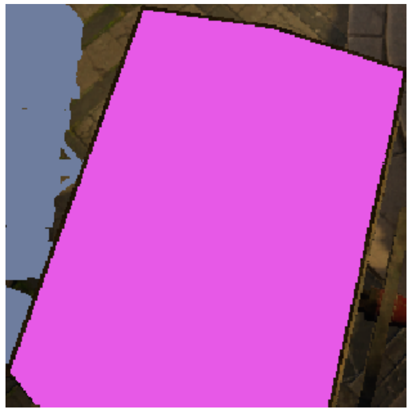
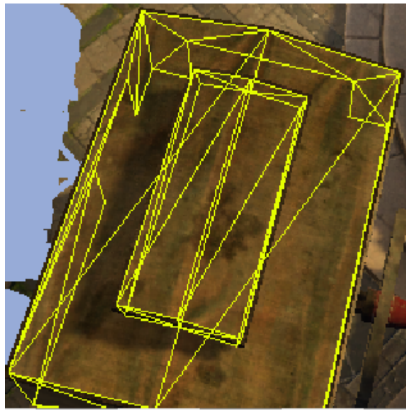
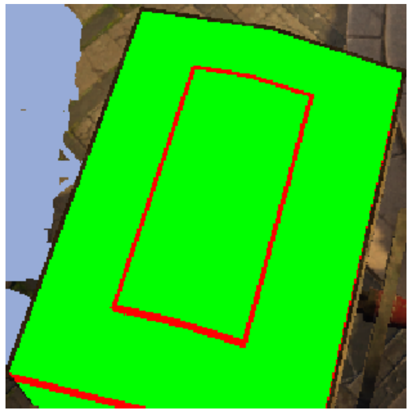


Event				
> Copy/Clear Pass #1 EID 1478 vkCmdCopyImage(2D Image 7991, 2D Color Attachment 709) Bound as UAV or copy - potential modification	Tex Before R: 0.0000000 G: 0.0000000 B: 0.0000000 A: 0.0000000 D: 0.0000000 S: 0x00	⬛	Tex After R: 0.0684780 G: 0.1356330 B: 0.2307400 A: 1.0000000 D: 0.0000000 S: 0x00	🟦
> Colour Pass #33 (1 Targets + Depth) EID 7254 vkCmdBeginRenderPass(C=Clear, DS=Load) 1 Fragments touching pixel	Tex Before R: 0.0000000 G: 0.0000000 B: 0.0000000 A: 0.0000000 D: 0.9594058 S: 0x00	⬛	Tex After R: 0.0000000 G: 0.0000000 B: 0.0000000 A: 0.0000000 D: 0.9594058 S: 0x00	⬛
> Colour Pass #33 (1 Targets + Depth) EID 7407 vkCmdDrawIndexed(6, 1) Depth test failed 1 Fragments touching pixel	Tex Before R: 0.0000000 G: 0.0000000 B: 0.0000000 A: 0.0000000 D: 0.9594058 S: 0x00	⬛	Tex After R: 0.0000000 G: 0.0000000 B: 0.0000000 A: 0.0000000 D: 0.9594058 S: 0x00	⬛
> Colour Pass #33 (1 Targets + Depth) EID 7762 vkCmdDrawIndexed(8148, 1)	Tex Before R: 0.0000000 G: 0.0000000 B: 0.0000000	⬛	Tex After R: 0.0666260 G: 0.0512690 B: 0.0423110	🟫

Draw call overlays


Overlay None

None
Highlight Drawcall
Wireframe Mesh
Depth Test
Stencil Test
Backface Cull
Viewport/Scissor Region
NaN/INF/-ve Display
Histogram Clipping
Clear Before Pass
Clear Before Draw
Quad Overdraw (Pass)
Quad Overdraw (Draw)
Triangle Size (Pass)
Triangle Size (Draw)

Limitations

- Synchronization issues tend go away in the capture
- Impossible to inspect inter-frame issues
- Capturing a hang could be tricky or impossible
- You cannot edit shader inputs or any Vulkan call parameters
- Requires a lot of RAM for capture, which may not be there on ARM devices

Transferring captures

- Captures aren't compatible between different GPU vendors, sometimes between GPU generations, and even driver versions
- It poses an issue in the cases where we cannot make a capture on the target device:
  - Due to a hang
  - Application doesn't run on the device
  - Target device doesn't have enough memory

Making a capture

- If the issue appears only for a few frames
  - Make a trace with GFXReconstruct
  - Replay it frame by frame and find the one you want
  - Tell RenderDoc which frame to capture
- If there is a hang causing capture failure
  - Make hang to go away by any means
  - Make a capture

Validation layers

- Validation layers are the easiest way to find Vulkan misuse
- Not every error or warning corresponds to a real issue
- Some useful validations are in the "warning" category - make sure to enable it!
- Enable GPU-Assisted validation to catch many of out-of-bounds issues in shaders

Driver debug options

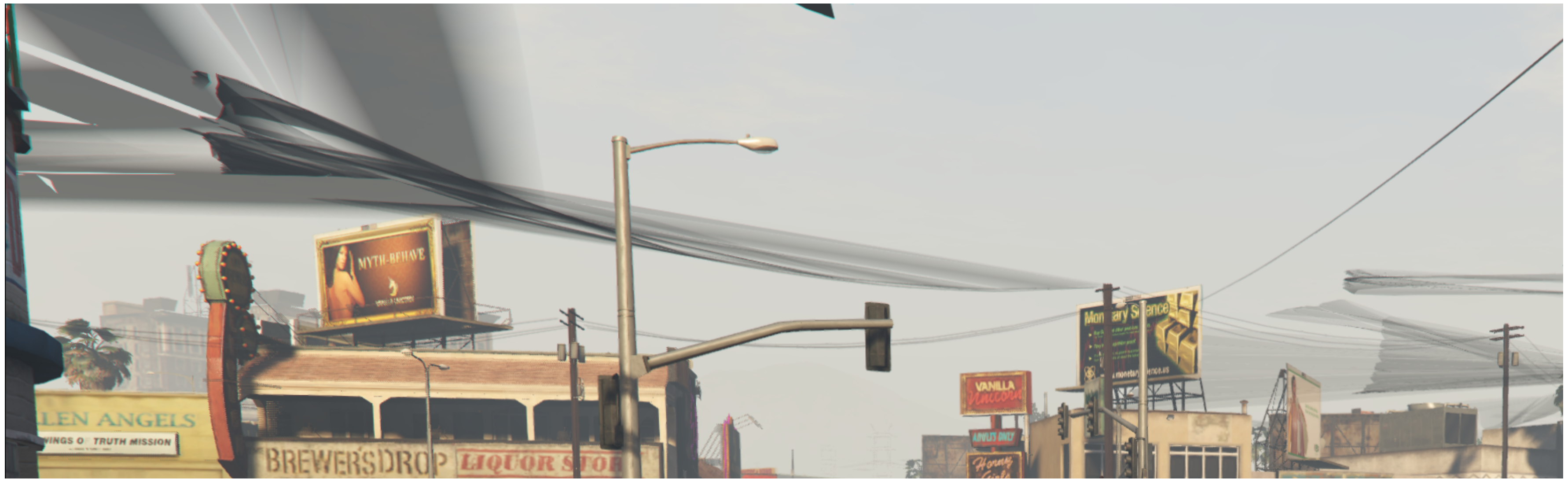
- Debug options to speed up your investigation
  - Synchronize every call
  - Re-emit state for every draw call
  - Force spilling for all shaders
  - Substitution of shader assembly
  - Selective disabling complex optimizations

After inspecting the frame

- If you know the problematic draw call:
  - It's impossible to just export it to C code

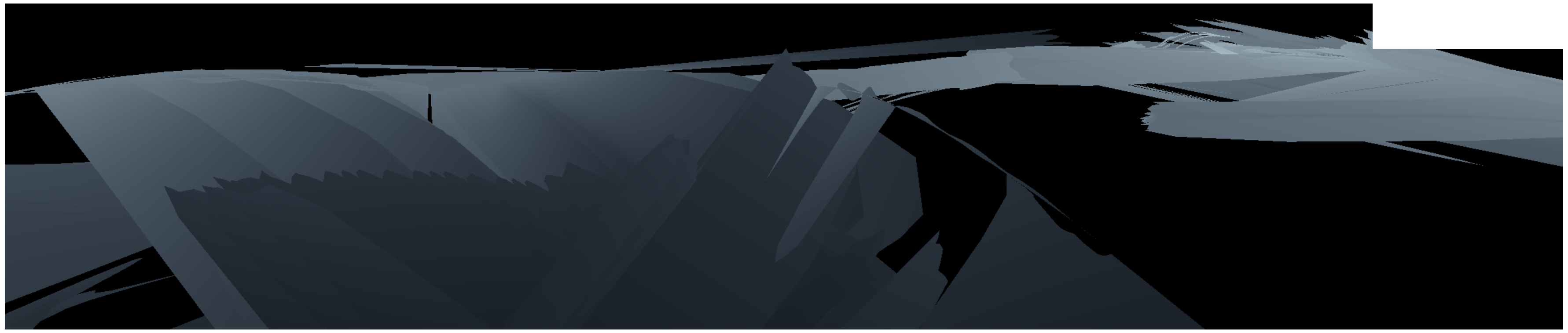
    GFXReconstruct is planning to support exporting a trace as a compilable code
  - It's easy* to manually create an Amber test mimicking the call - you could export all buffers/textures from RenderDoc!
  - Still, use this as a last resort

Recently I had to do this for an issue with tesselation shader in GTA V on Turnip



It was impossible even to make a RenderDoc capture to inspect the draw call due to the lack of RAM

The draw call extracted as an Amber test

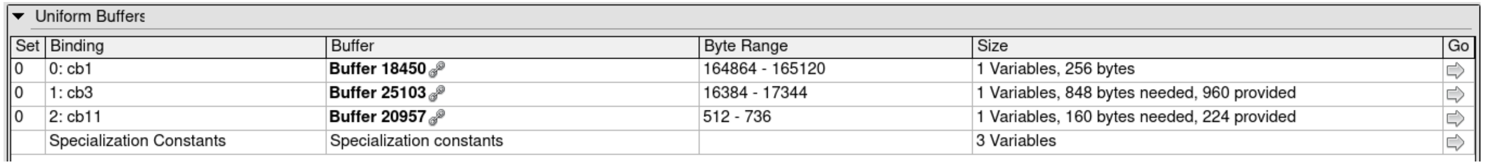


After that it was much easier to debug

Amber reproducer

```
BUFFER indices_buf DATA_TYPE uint32 SIZE 2808 FILE TEXT indices.txt
BUFFER vertex_buf DATA_TYPE int8 SIZE 38556 FILE BINARY vertex_9978.bin
BUFFER cb1 DATA_TYPE int8 SIZE 1048576 FILE BINARY ubo_16835.bin
...
PIPELINE graphics test_pipeline
    ATTACH vert_shader
    ...
    INDEX_DATA indices_buf
    VERTEX_DATA vertex_buf LOCATION 0 RATE vertex FORMAT R32G32B32_SFLOAT OFFSET 0 STRIDE 36
    ...

    BIND BUFFER cb1 AS uniform_dynamic DESCRIPTOR_SET 0 BINDING 0 OFFSET 0 DESCRIPTOR_OFFSET 4421
    BIND BUFFER cb3 AS uniform_dynamic DESCRIPTOR_SET 0 BINDING 1 OFFSET 0 DESCRIPTOR_OFFSET 9216
    BIND BUFFER cb11 AS uniform_dynamic DESCRIPTOR_SET 0 BINDING 2 OFFSET 0 DESCRIPTOR_OFFSET 102
    ...
```



Uniform Buffers

Set	Binding	Buffer	Byte Range	Size	Go
0	0: cb1	**Buffer 18450**	164864 - 165120	1 Variables, 256 bytes	⇨
0	1: cb3	**Buffer 25103**	16384 - 17344	1 Variables, 848 bytes needed, 960 provided	⇨
0	2: cb11	**Buffer 20957**	512 - 736	1 Variables, 160 bytes needed, 224 provided	⇨
	Specialization Constants	Specialization constants		3 Variables	⇨

Other methods to try

- For hangs - inspecting the GPU state dumped on hang
- For shader yielding unexpected result - instrument it to see intermediate results on GPU

  Not available on open-source drivers, but I made a prototype for Turnip
- Compare emitted states to proprietary driver if there is any

The End

Any questions?