gfx-rs: lessons learned

Dzmitry Malyshau

Rust Graphics Meetup - 02 Oct 2021

Intro

What is gfx-rs?

RIP 2014 - 2021

As many projects, it started with something that is very far from what it ended.
We will go through case studies and extact lessons. With some memes to help.

Epoch: pre-ll

2014 - 2018

Let's try to minimize a chance for errors!

- vertex layouts
- resource bindings
- render targets

Macros

```
gfx_defines!{
    vertex Vertex {
        pos: [f32; 4] = "a_Pos",
        tex_coord: [f32; 2] = "a_TexCoord",
    }
    pipeline pipe {
        vbuf: gfx::VertexBuffer<Vertex> = (), //<-- init-time portion
        out_color: gfx::RenderTarget<ColorFormat> = "Target0",
    }
}
...
let data = pipe::Data { // bind-time portion
    vbuf: vbuf, // it was long ago!
    out_color: window_targets.color,
};
```

Result

Users constantly misunderstanding the difference between macro-derived structures: one for initialization, one for run-time binding.

Macros were also very difficult to write and debug, not properly supported by tools.

Lesson-1

Stay away from macros, unless strictly necessary.

We used them because we were still learning the language (way before 1.0), no best practices were known.

Function-style draw calls

```rust
/// Draws a `slice::Slice` using a pipeline state object,
/// and its matching `Data` structure.
pub fn draw<D: pso::PipelineData<R>>(&mut self,
    slice: &slice::Slice<R>,
    pipeline: &pso::PipelineState<R, D::Meta>,
    user_data: &D)
```

Most state is explicitly provided for each draw, then compared against the current to generate the state changes internally.

Result

Users saving the fat "data" objects and mutating them between the draws. Effectively, re-introducing the same state we tried to remove!

Their responsibility to fail, but still our problem.
A good API shows the trail to success, not placing traps on the way.

Also, deducing the internal state changes means the overhead is not explicit, and is harder to reason about.

Lesson-2

Functional style isn't universally better. It needs to be practical and match the target domain. Computer graphics is inherently stateful, and there is a performance cost to trying to change this.

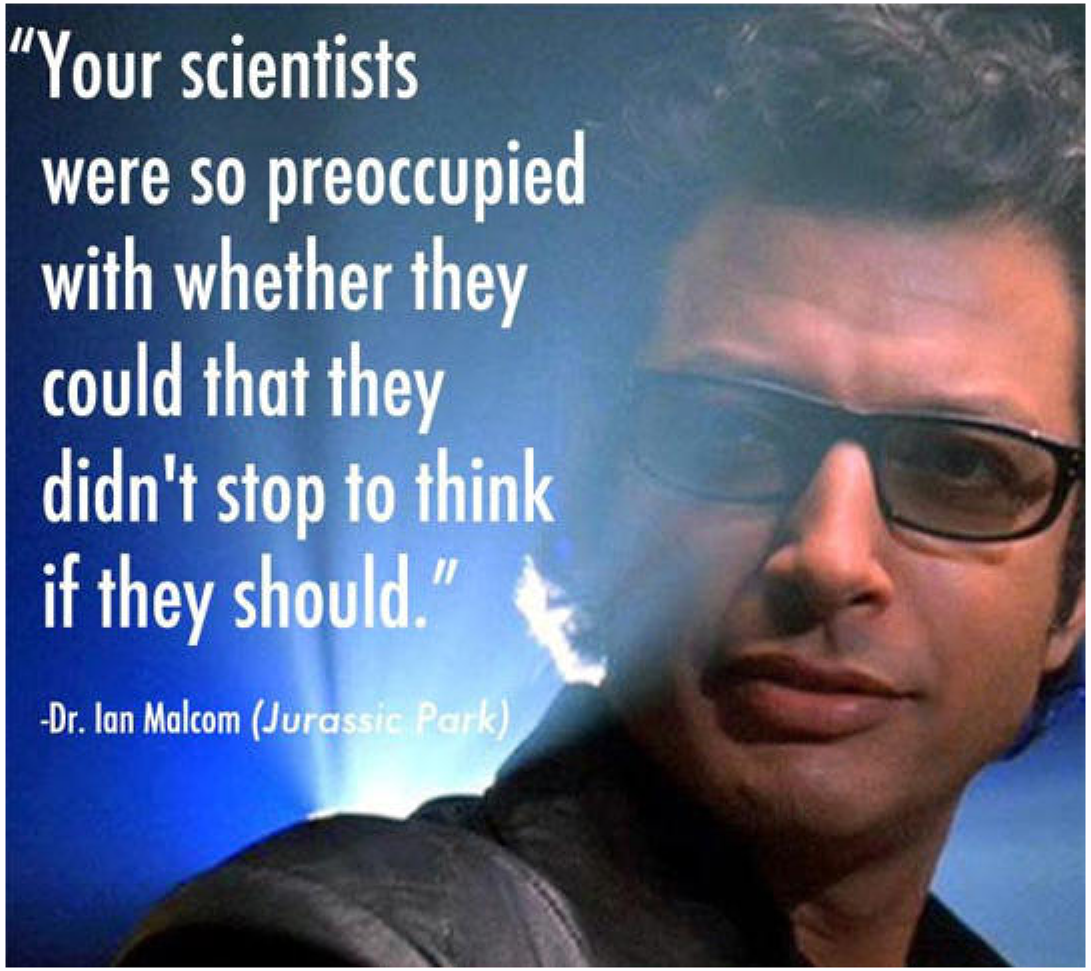
Stack-based approach might have been better: Luminance has some of it.

Run-time knowledge

But what if we don't know this at compile time?..

Armed with type system, we can do it at run-time!

```rust
impl<'a> DataLink<'a> for RawRenderTarget {
    type Init = (&'a str, format::Format,
        state::ColorMask, Option<state::Blend>);
    ...
}
```


"Your scientists were so preoccupied with whether they could that they didn't stop to think if they should."
-Dr. Ian Malcom (Jurassic Park)

Result

Even more *confusion*!
Users basically followed the magic incantations without fully understanding them.

Lesson-3

Always build types over plain data, not the other way around.
If unsure, start with simple data and functions. Make typed layer optional.

In this case, the user should choose between using typed layer or not typed. Instead, they picked between the typed layer and the raw typed abomination over it.
Somewhat related to Vulkano.

Pipelines

In the API users defined graphics pipelines. This matches the concept in modern APIs, but at that point gfx pre-ll only had OpenGL support. Later it got DX11.

“We are ready for the next gen!”

We can eliminate a class of errors by requiring users to provide all information every time... Does not solve the problem.

Result

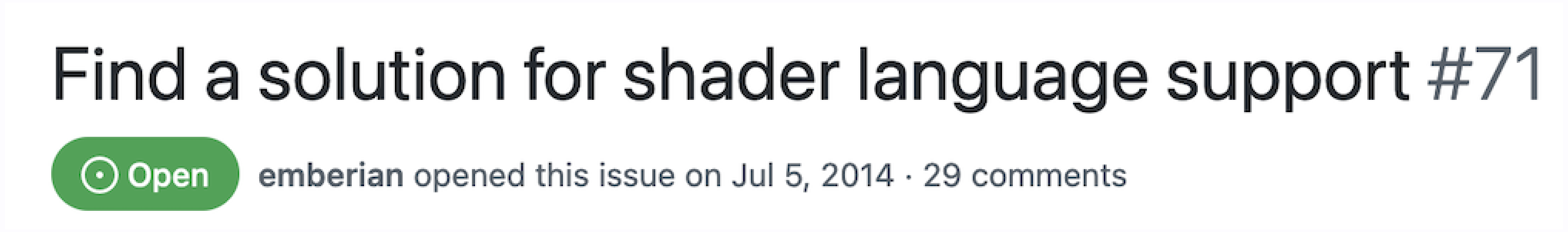
pre-ll pipelines never ended up being translated into the *real* graphics pipelines of modern APIs. There were other obstacles preventing the modern backends to be written (object lifetime on GPU, descriptors, etc).

Lesson-4

Don't design for the unknown future. Design for the known today instead.

There are many reasons for the design to need to change, and clarifying the future is just one of them.

Very similar to the advice you can read on various posts and threads, e.g. on HN.


Find a solution for shader language support #71

Open emberian opened this issue on Jul 5, 2014 · 29 comments

Lesson-5

Users need solutions. A GPU abstraction without a common entry point for shaders is just a part of a solution.

Epoch: gfx-hal

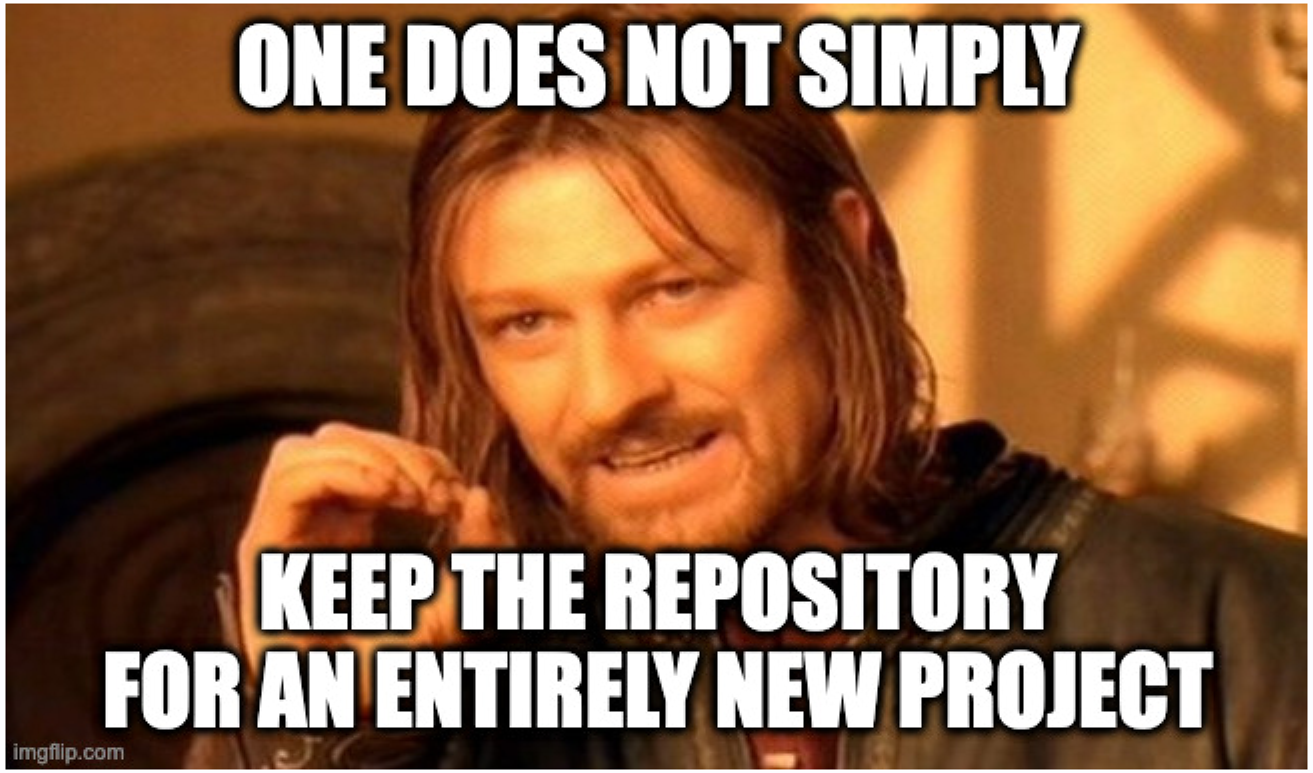
Oct 2017 - Summer 2021

Let's expose the lowest level

API Alignment

Follow **Vulkan** precisely, worship **Khronos**. Participate in Vulkan Portability Initiative

Include all the nasty details, like render sub-passes and explicit memory control (i.e. be unopinionated)



https://github.com/gfx-rs/gfx

Lesson-6

Users *still* get confused on what gfx-rs is.

Branding is important. Heritage needs to be preserved. Replacing the project on the same repository with a different one causes too much pain and confusion in the long run.

Transition killed most of the contributors and users.

Backends

Each backend in a separate crate: gfx-backend-vulkan/metal/gl/etc.
Everything on the user side is generic over `<B: hal::Backend>`.

Result

`<B>` bubbling up all the way to the top of user code, ruining abstraction, crawling into seemingly unrelated things, like `Camera<B>`.

Lesson-7

If you don't know how users can solve a problem, it's your problem.

Zero-cost abstraction

```
unsafe fn bind_vertex_buffers<I, T>(
    &mut self, first_binding: pso::BufferIndex, buffers: I
) where
    I: IntoIterator<Item = (T, buffer::SubRange)>,
    T: Borrow<B::Buffer>,
    I::IntoIter: ExactSizeIterator;
```

There are always costs: debug run-time cost, compile time cost, maintenance cost, mental overhead.

Result

Wasting time trying to navigate the types jungle.

Lesson-8

Types can be overwhelming and inefficient. They hinder debugging, compile times, API usability.
Reaching safety with run-time checks is a perfectly valid alternative.

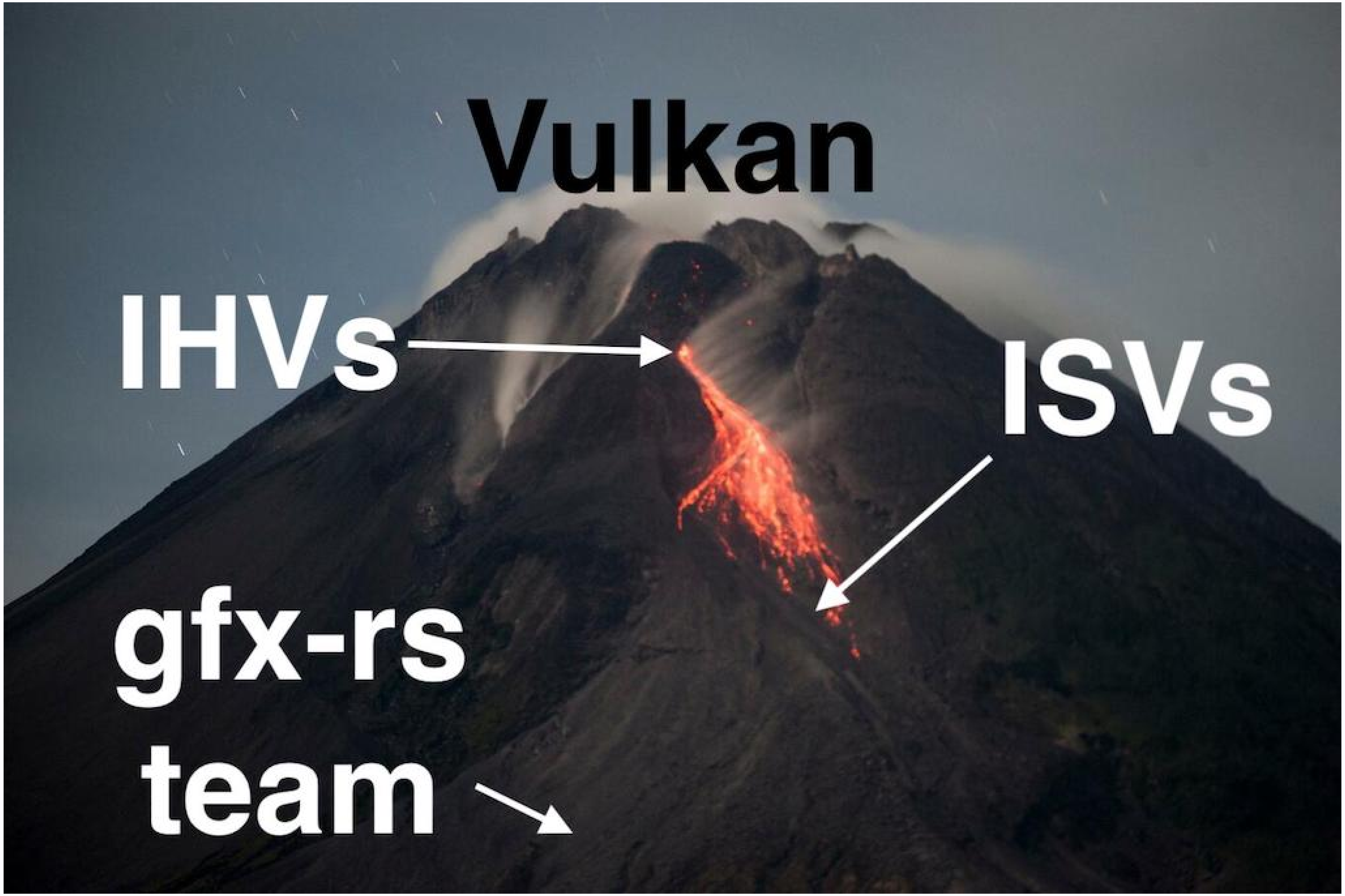
We are implementing Vulkan everywhere
And you are all Vulkan gurus, right?
You know Vulkan, right?
imgflip.com

Put a ton of trust into Vulkan as well as **ourselves**.

Result

Vulkan is complicated. Really.
There is a lot of complexity, and a lot of tribal knowledge.

And we signed oursevels to be *slaves* of the decisions made by Vulkan working group.


Vulkan
IHVs
ISVs
gfx-rs
team

Lesson-9

Recognize your assumptions and wishful thinking.

If something is open and developed in collaboration, we **want** it to be the best thing in the world. But we should not assume that.

An API developed behind closed doors with tight feedback loop from developers can feature better designs.

Examples: copy alignments, sub-passes, resource states.

Take Vulkan
Implement over all backends
Let somebody else serve users
Let somebody else serve users
imgflip.com

Result

We couldn't build any decent high-level abstraction...
Users were crushing into the complexity and leaving.

Lesson-10a

Put users first. Serve real needs.

Building the foundation without a good understanding of higher levels is a recipe for disaster.

I.e. could bring wgpu-rs *before* gfx-hal.

Lesson-10b

Being universal and unopinionated is *not* a feature. It's an instrument to build something **actually useful** on top. But once it's build, there is an inevitable temptation to avoid compromises. **Opinionated >> Universal**.

Lesson-10c

Low level primitives are as **re-usable** as narrow their API is. gfx-hal API is large.

If gfx-hal had a smaller API, there would be much higher chance for other libraries to pick it up and build on top.

gfx-portability

Vulkan on top of anything, via gfx-rs.

- heavy contributor to Vulkan Portability
- beat MoltenVK in Dota and Dolphin Emultator
- first to run Vulkan CTS on macOS

Me + LunarG
Devs linking to MoltenVK directly
imgflip.com

Result

Nobody cared:

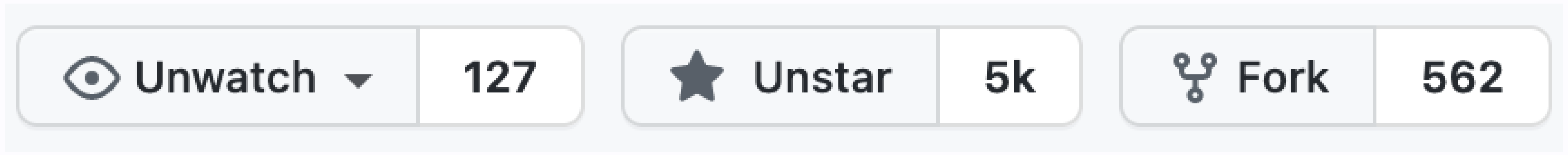
- Rust users don't care about C APIs
- C users just need something that survives long enough

Lesson-11

Fighting against the flow is hard

Nobody woke up in the morning thinking: I am going to make VkPi run superb for my app.

Community



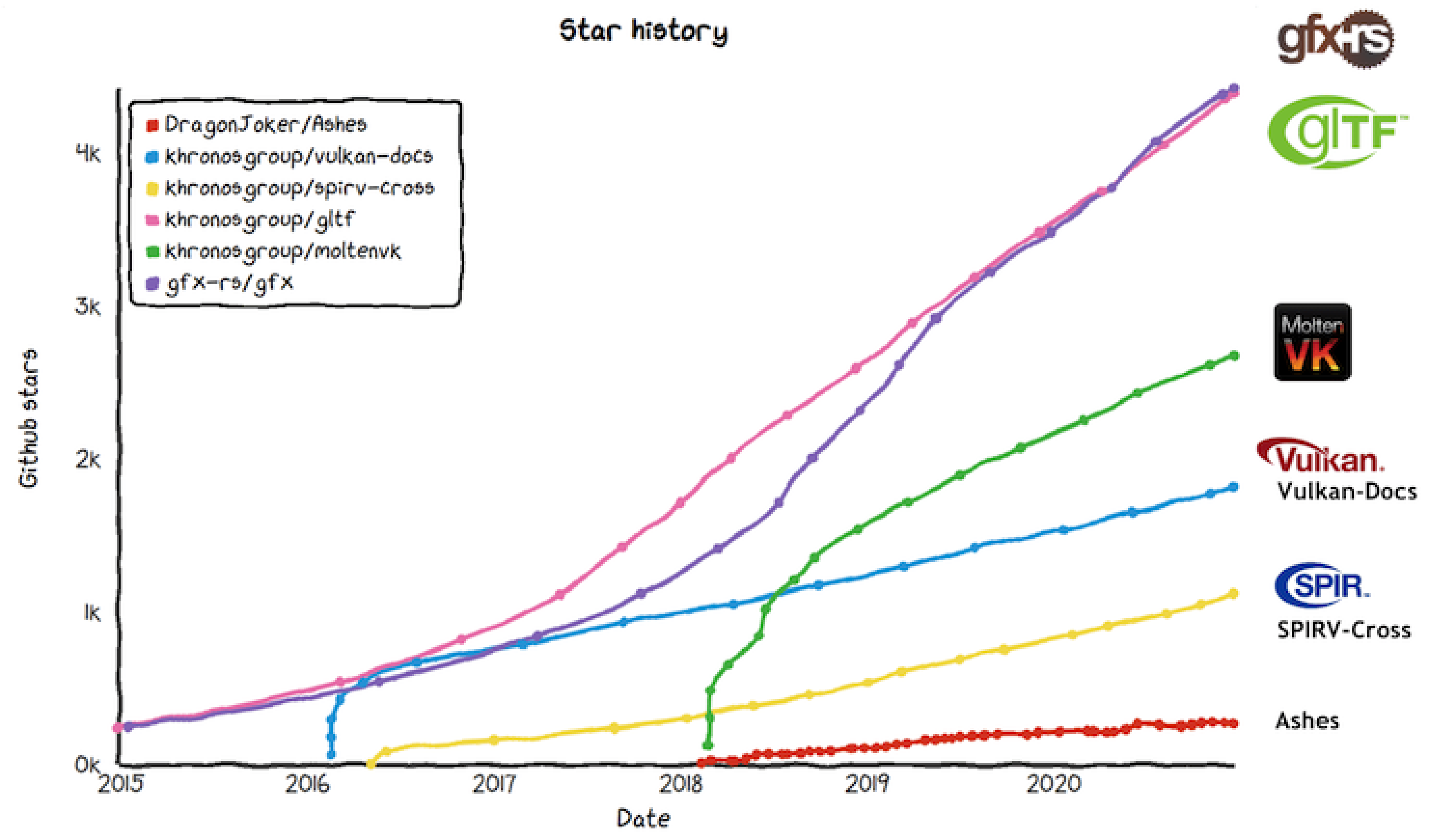
Commits: 6K+

Contributors: 255, and growing

Code: 65K LOC, and growing

Maintainers: 1

Khronos-Related GitHub Projects



https://star-history.t9t.io/#khronosgroup/moltenvk&khronosgroup/gltf&khronosgroup/vulkan-docs&khronosgroup/spirv-cross&gfx-rs/gfx&DragonJoker/Ashes

Lesson-12

Contributor/commit counts and project stats are unlikely important, at all.

Some contributions are harmful. Can't expect the authors to maintain their code. Smaller scope is a hidden advantage.

Epoch: wgpu-hal

Summer 2021 - future

Keep it simple, keep it close

Design

Follow WebGPU where it's reasonable.
Unlike Vulkan, WebGPU is designed to map to other APIs.

Metrics: 2.5-3x less code, faster compile times.

what's next?

gfx-rs repository is in **maintenance** mode.
But the ideas of gfx-rs live on within wgpu-hal and wgpu-rs.

Further reading

- https://gfx-rs.github.io/2021/07/16/release-0.9-future.html
- http://kvark.github.io/3d/api/vulkan/2021/06/20/vulkan-alignment.html
- https://github.com/gfx-rs/portability/issues/250