GPU Web 2019-01-22 San Diego F2F Day 1

Chair: Corentin
Scribe: Ken with some help, David for the shading language side meeting
Location: Google San Diego, Google Meet

TL;DR

- Status updates:
  - Intel: summarized their contributions to the WebGPU efforts and porting the Aquarium demo to Dawn as a proxy for WebGPU
    - Currently using instancing, while change to use dynamic buffer offsets to better represent real applications.
    - Porting a WebGL app to WebGPU with 1-1 translation leads to poor performance
    - Unclear what the incremental porting story will be from WebGL to WebGPU
  - Apple: landing a lot of code for the WHLSL to MSL compiler, implementing current IDL in Safari and has a spinning cube running.
  - Google: working on whc a WHLSL to SPIR-V compiler that fits in 100kb in WASM. Dawn getting close to the WebGPU feature set. Barely starting Chrome integration.
  - Microsoft: trying to figure out how to keep evolving HLSL for the Web and DXC.
  - Mozilla: have hellotriangle running in native based on gfx-rs' Vulkan portability
    - David presented a statement from Adobe on source language vs. IR.
  - has remoting and IPC support.
- Shading language discussion with the whole group:Babylon.js would need support for a client-side compiler in less than 100kb or a mechanism to replace their use of the preprocessor.
  - David presented on the WebGPU shader programming model which should resolve issues independently of SPIR-V or WHLSL.
    - General consensus with this approach.
- Kai presented his experiment writing a WebGPU CTS harness
  - No concerns with the direction taken by the harness,
  - Questions around the Dawn node module that the experiment contains.
- Administrative discussions:
  - Corentin suggested we write the spec and the WebGPU test plan at the same time.

    - Dean created a draft charter for a WebGPU WG that would stamp specs coming from the WebGPU CG.
    - WebGPU is now the Web GPU API to match WebAudio.
- Shading language side meeting:
    - Discussion around data races: it is impossible to detect them at runtime. Could follow LLVM's lead on this.
        - Races can be made safe by always using atomics, though that would be slow.
        - Making races safe in Vulkan would require a SPIR-V extension, GPU vendor support and extensive testing to qualify drivers. Drivers without the extension could be qualified independently.
        - Allowing resources to alias the same storage makes it hard to detect races.
    - Several types of operations require being in dynamically uniform control flow.
        - Tradeoffs between the analysis for validation and the flexibility allowed to shaders and the performance of the lowering of operations.
        - WHLSL had a conservative uniformity analysis but it was removed because it was too complex.

Tentative agenda

- Shading language
- Memory model
- Plans for the CTS
- Plans for the spec work
- Multi queue
- Compressed textures
- Buffer mapping
    - #147 - mapAsync
    - #154 - commandBuffer.uploadBuffer
    - #156 - buffer sub ranges
- Creation of the swapchain
- IDL-athon: see the snapshot tracker
- Produce an example of a WebGPU program
- #167
- Review draft WG charter #15
- Agenda for next meeting

Second Day's Notes

Attendance

- Apple
    - Dean Jackson
    - Justin Fan
    - Myles C. Maxfield
    - Robin Morisset
- Google
    - Austin Eng
    - Corentin Wallez
    - David Neto
    - James Darpinian
    - Kai Ninomiya
    - Ken Russell
- Intel
    - Bryan Bernhart
    - Jiajia Qin
    - Jiawei Shao
    - Yunchao He
- Microsoft
    - Chas Boyd
    - Jungkee Song
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
- Joshua Groves

Intel Presentation

- Slides
- Moving Dawn to descriptors rather than builders
- Aquarium on Dawn native uses large uniform buffers
    - JS: poor performance if ported naively to Dawn due to need to Queue.submit per fish
    - CW: currently setSubData is a device operation.
    - CW: no limits implemented yet. Might support this with dynamic buffer offsets in the future.

- DM: if I were doing the port I'd assemble all the data and do one submit which submitted all the draw calls.

- OpenGL port of Aquarium
- DM: think Aquarium isn't a good candidate for performance comparisons if we're using instancing. The point of WebGPU is to allow many draw calls.
- CW: agree. Dynamic buffer offsets will help here
- KR: Source code for Aquarium on Dawn?
- DM: are you using SPIRV-cross for Dawn? What's the shader input format?
  - JS: GLSL.
  - CW: Dawn takes in SPIR-V. There's a compilation step.
  - KN: this is similar to other ports.
- DM: what WebGPU specific optimizations?
  - JS: multithreading.
- DM: you're the first ISV to port WebGL to WebGPU. Any feedback would be great. What was annoying?
  - JS: it's not that easy to translate a WebGL app to WebGPU one with good performance. First we tried 1:1 mapping version, but perf was very bad.
- MM: what's the story for the lower-level ports?
- DJ: Apple's translated millions of lines of OpenGL code to Metal code.
- CW: There's D3D11On12. Can incrementally replace pieces.
- CB: handles the buffer management for you inside that wrapper.
- MM: incremental updates work fine?
- CB: it's mostly been used as a binary.
- JG: can you use a little bit of D3D12 intermixed in the D3D11 app?
  - CB: yes.
- CB: what's happening behind the scenes in the OpenGL version of the Aquarium app?
- CW: it's translating the WebGL Aquarium 1:1 to WebGPU concepts.
- CB: can debate whether it's a 1:1 port. All we're doing is the buffer management code that was formerly in the driver.
- KR: wouldn't that be one buffer per fish?
- CB: no. Driver ping-pongs them, or dynamically resizes number of buffers.
- JG: number of things you can do there.
- CB: may have to compare well-written WebGL code vs. WebGPU code.
- CW: will need some education effort.
- CB: try to make the old API work and then help them tune performance.
- MM: are we coalescing that it should be possible to implement WebGL on top of WebGPU?
- CW: I don't think so. Will happen eventually because Internet is full of people who do stuff like that.
- JG: we're looking at replacing Canvas2D on top of WebGL.
- CW: don't think this group should handle that. Might happen eventually.
- KR: To make fast the scenario : many draws per queue. Is that predicated on dynamic buffer offsets?

- DM: No. Simple buffer update proposal would make it fast?
- CW: would suck on mobile.
- DM: you'd want to upload everything before the pass.
- CW: can do that with setSubData.
- DM: how does OpenGL on mobile do this?
- CW: magic. Literally.
- JG: since this is a weird way of doing it, can we rephrase the demo in terms of instancing?
- CW: have to upload data to uniform buffers every frame.
- CB: the fishes are skinned.
- MM: Use compute shaders to compute positions.
- KR: The point of low level apis is to draw lots of batches of small triangles. Compute should not be necessary.
- CW: the problem isn't a feature gap. If you want a uniform buffer for 30k objects then you really want buffer offsets. Here we use instanced rendering.
- MM: how much of an OpenGL app should someone have to rewrite?
- CW: all of it? The rendering code?
- MM: then they could do anything they want. But if we're trying to not require them to rewrite their entire app then we have to give them the primitives.
- CW: don't see a way we can have a seamless way of incrementally turning WebGL
- DM: Dispute that you need dynamic buffer offsets. Use 30k bind groups if you have to.
- CW: we ran out of time before the F2F. We could redo it with bind groups.
- CW: using dynamic buffer offsets is more efficient than using many bind groups.
- DM: you're saying we need dynamic buffer offsets to get this level of perf.
- MM: I did an analysis of this. Creating a bind group is expensive enough that we don't want to do it each frame of the app.
- DM: agree -- not talking about creating every frame.
- MM: question is whether it's practical to create all bind groups up front.
    - 1) Create 30K bind groups up front.
    - 2) Create 30K bind groups during app's lifetime.
    - 3) Use dynamic buffer offsets.
    - Option (2) is probably not acceptable.
- CW: it's a good question and we can answer it with benchmarks.

WebGPU Shader General Items (dneto)

- Slides
- DN: working on whc
- RC: what's the input?
- DN: WHLSL.
- DJ: if we could take Myles' compiler and un-WTF it…
    - DN: we looked at it. It's a lexer.
    - MM: it's a lexer, parser, and 95% complete.

- DN: maybe you're further ahead than we found.
- MM: the last few weeks is when a lot of it landed.
- DN: it's not a lot of wasted work regardless.
- CW: having whc translate WHLSL->SPIRV only makes it smaller and more specialized.
- DN: considered writing it in Rust, too.
- MM: we should collaborate.
- DN: we would like to. We're not very far along.
- DJ: think it's great if there's another one. In W3C you're supposed to have two independent implementations. Just concerned we're doing the same work.
- DN: what do you think about the test suite?
- DJ: our thought with the ref compiler was going to have a lot of rendering tests.
- DN: this is not sufficient but gives good language coverage. Not CTS.
- MM: another way of saying it: our test suite will involve WebGPU and actually draw stuff. More tests are better.
- CW: here, the test suite is for whc. Don't think the point is to take the test suite and run it for WebGPU. WebGPU takes in an undefined language right now. If WebGPU takes in SPIRV then this is at the top of the stack.
- DJ: we were thinking to test the SPIRV we generate (in the ref compiler) by feeding it into some native Android or Vulkan app.
- DJ: another idea: internal-only flag for WebKit that would accept MSL, so we could feed in WHLSL and MSL, and compare the outputs.
    - Could start with GLSL, use SPIRV-cross…
    - DN: yes, use what customers use.

Status updates

- Google
    - Dawn getting close to WebGPU feature set
    - Most features work. Have barely started looking at Chrome integration. Should mainly be scaffolding and plugging ends together.
- Apple
    - DJ: Similar to last meeting but in addition to that MM has been landing a lot of code to generate MSL to WHLSL (only) but not exactly plugged in yet.
    - DJ: adding something else might not be very hard
    - DJ: implementing the WebGPU API as it stands today
    - JF: have been taking snapshots of API per-feature. Recently working on bind groups.
        - Missing: depth textures (depth culling), textures, texture sampling
        - All of these are on the to-do list
    - JF: have vertex inputs, vertex attributes working.
    - JF: after we're feature complete-ish, will update it based on what we decide today.
    - DJ: this demo is something we'd like to project.

    - JF: have been writing WebGPU code without knowing about Dawn - didn't realize it had a Metal backend.
    - CW: our sample code is C++. Kai's been writing JS too.
    - MM: now that we have triangles on the screen, we have some questions about API design. If pipeline layout has thing bound that's not consumed, is that legal? Should go through a program at this F2F that draws a moving triangle and agree upon the API contracts.
    - CW: sounds good. As soon as we take a snapshot we should fill it in with the validation errors because otherwise we won't have interop.
    - Justin's basic demo:
        - https://gist.github.com/JusSn/46814680aeec5a007537970a6d47a050
        - (Ignore the Metal shader)
    - JF: have been pretty verbose about everything in the demo.
    - ...Discussion…
    - DM: view creation for swap chains. Do you think it's OK to do this every frame? I find it non-ideal. It's a device operation, shouldn't require those to be frequent.
    - MM: agree. A web site shouldn't know whether the swap chain is double- or triple-buffered.
    - DM: maybe we should return both the texture and a view during getNextTexture?
    - MM: that's fine.
    - CW: as the user, sort of want to control the view. Think about WebXR. Want to know whether that's a 2D texture array, etc.
    - DM: ok. Another question: using persistently mapped buffers?
    - JF: this is very much a prototype. We'll be getting rid of it.
    - RC: would be great to have this sort of sample code in the spec.
    - DJ: I have an action item to put together the draft spec.
    - RM: would like to discuss memory model and uniformity model for standard library.
- Microsoft
    - RC: gave feedback on various issues and PRs
    - Brandon and Brian making contributions
    - CB: no updates for this week. Few weeks more.
        - Trying to figure out how to keep evolving this language (HLSL) both for the web and for what's going on in the DXC repo.
- Mozilla
    - DM: have Hello, Triangle running. Not going to show it.
    - Have native library that implements WebGPU IDL from ~1 mo ago.
    - Has a C API. Our sample uses glfw for window system integration.
    - Using gfx-rs for its backend. Targets a single backend, runs on all platforms.
    - Allows us to rapidly update the IDL.
    - Mozilla prototype:
        - native C (glfw wsi)
        - IDL from ~month ago

        - based on gfx-rs/portability
        - remoting/IPC support
        - features:
            - swapchain acquisition and presentation
            - resource creation/recycling
            - bind groups
            - render/compute passes, dispatch/draw calls, vbuf/ibuf
            - command buffer recycling
            - transitions
        - all 3 backends
        - missing:
            - error handling
            - buffer updates
    - Error handling is nonexistent so far.
    - Next step: integration into Gecko.

DN: Statement from ISV “On the value of IRs over HLLs for GPU APIs”

[Slides]

Eric Berdahl, Senior Engineering Manager, Mobile Video Products, Adobe

I have been considering your question about WebGPU, shaders, and potential Adobe applications built on the WebGPU platform. As I always begin when answering questions about "Adobe", I am compelled to observe that Adobe is a large engineering community with multiple opinions and answers on most questions. As such, I am reasonably confident that there exists an Adobe group that would not terribly care about the shader source for a WebGPU-based application and would use the tech that's available.

Closer to my heart are the Digital Video and Audio applications (things that rhyme with Premiere Pro, After Effects, Audition, Premiere Rush, and the like) and the Digital Imaging applications (things that rhyme with Photoshop and Lightroom). Applications of that caliber will be reserved about deploying an application on a technology stack which requires all shaders to be represented in string source, as opposed to a technology stack which allows shaders to be represented in an intermediate representation.

One aspect of this reservation is IP protection. I've heard it stated that IRs are relatively easily cross-compiled to one or more high level languages -- ala spirv-cross' support for glsl, MetalSL, and HLSL -- thereby blowing past IP protection. This argument is misguided and unpersuasive. Any machine code representation (even x86 machine code) can be translated to a high level language, but that translation does not recover the expressiveness of the algorithm and ability to understand what a developer has authored. Further, such translations and attempts to recover the higher level semantic encoded in the individual operations resembles reverse engineering (to a greater extent that just "View Page Source" does in Chrome, for example) and therefore provides both a social barrier to that path and possible reactions and responses from a developer so compromised.

While IP protection is an important part of why IR assets are valued, over source strings, for shader representations, quality is at least as large an issue, if not larger.

Modern GPU APIs have moved toward IR representations and away from embedded source strings. For example, each of the major GPU APIs introduced or revised in the past 5 years, Metal, Vulkan, and DirectX, have IR representations they consume. My experience (in the context of Adobe's DVA applications) has been that compiling shaders to IR at app-build time has been a net increase in stability and quality for our applications. To be sure, there remain quality differences between drivers from various vendors (i.e. drivers are differently picky and finicky about what shaders they accept and reject, even shaders that are spec-conforming, and for which reasons). However, when the shader asset is an IR, as opposed to a source string, working around driver bugs and idiosyncrasies has been faster and easier than when the shader asset is a high level source string.

Even more attractive is if that IR is one I can validate in a non-web context. For example, if the IR consumed by WebGPU were also directly consumable by Metal, Vulkan, or DirectX, I am more likely to attempt to build one of my apps for the web and significantly more like use that API when I do.

Short answer: I am significantly more likely to build and deploy an app using a web GPU API in which my shaders can been provided in a binary form than I am to use a similar API which requires shaders to be specified by a string of high level source code.

- DN: Adobe also gave a talk about this at SIGGRAPH last year. This is a summary.
- MM: we'd like to address some of these points.
- RC: I also talked with an ISV (David Catuhe, BabylonJS). At first he didn't care about the shader input format. His shaders make heavy use of the preprocessor and GLSL. Corentin and Kai spoke with him; at first, didn't really care. Later, had email conversation; told him about shipping a compiler and preprocessor. He changed his

opinion. If it's more than 100K it's a non-starter. He has a strict size budget for Microsoft properties.

- JG: is 300KB OK?
- RC: 100K. Currently the whole library is around 500K (compressed, minified) for all of BablyonJS.
- CW: that's why the design goal for whc is to fit nicely into 100KB of compressed wasm. It's size-efficient esp. if you don't have strings. Think we can do it.
- CB: did he say why he needs to do recompilation on end users machines?
- RC: he's specializing based on the use on clients' machines.
- CB: that's something we're working on. Need data parallel architecture that is bad at branching. Handling specialization well is key. Looking at cases where we're going to ask the driver to specialize things frame-to-frame under app control. Some of these specializations will involve recompilation.
- CW: some of these will be addressed with specialization constants, but the question is how they affect bind groups. Also they require a full recompilation at the driver level.
- CB: we have access to some of the mechanisms that native APIs do, but it's something we'll have to keep in mind. But something unique about GPUs probably basically forever is that branches (even without divergence) will be costly.
- RC: one other piece of feedback: would be against doing shader compilation on the server. One / many of their customers wants to download onto the client, and they do everything on the client. Customers don't want to have to point to a central CDN, either.
    - KN: we haven't been considering compiling on the server; it's a non-starter.
- JG: want to clarify, we talked about hosting the compiler on a CDN so it can be shared among web sites.
    - KN: right, but you don't have to use the CDN.
    - JG: it can be a tradeoff between hosted library size and putting it on the CDN. Could save on wire transfer size by using a standard CDN copy of it. Improves quality for many people who don't have both requirements./
    - RC: his feedback to me was, even if it's on a CDN, the bandwidth costs are high./
    - CW: main concern was the compiler size. We've taken that into account in our plans.
    - CB: it boils down to cost. Running on the client is free from a hosting perspective.
- BB: are you going to put Adobe's feedback somewhere?
    - CW: we could ask for a written statement if it's helpful.
- MM: would like to address some of the points on Adobe's slide.
    - MM: most of these points we've heard before. For issues like IP exposure, JavaScript has already solved it: people minify their scripts. Making a language use bytes instead of characters is not the only way to solve this problem.
    - CW: do we give up on view-source-ability as a goal?
    - MM: you can't have both. They're at odds with each other. view-source is an integral part of the web.
    - RM: some of the idea is that they can minify if they want to, but others can keep their code readable.

- MM: correct.
- MM: stability and quality are not properties of binary vs. text languages.
- KN: they're a property of SPIR-V and IR. We're talking at a concrete level: WHLSL vs. SPIR-V.
- DJ: are we talking about IRs or compilation targets? The point applies just as well to WHLSL in terms of stability. Why is WHLSL not achieving this?
- CW: there's no real-world example showing that WHLSL has this property. SPIR-V has ~5 years of people building up an ecosystem around it while WHLSL is more at the prototype level. It's true that JS is farther along, but it's orders of magnitude more people working on it than on WHLSL. It's not possible to say that these properties translate from the most popular language (JS) to the WHLSL language.
- DN: level of abstraction was useful for Eric from Adobe to work around incompatibilities.
- MM: I'm curious about that. This post says "easier". Is it easier to work around something where you have to rewrite the source.
- KN: the change to a line of code has large effects due to recompiling. If you know the code you're feeding into the system is closer to what runs on the GPU, you get more repeatable results. E.g., bad codegen. It's more repeatable with the IR rather than the shading language.
- MM: you mentioned we should compare specifics. Don't think SPIR-V is really low level enough.
- CB: in our tools, you'll see the HLSL source code, then IR (maybe SPIR-V), then maybe lower-level IR, then GPU instruction set. Then you can see what the mistake is. At that point you're fixing your source code rather than hacking an intermediate binary. That's better than hacking the binary. I admit the SPIR-V ecosystem is probably farther along.
- DN: Eric's experience was that he was an early adopter and we were putting lots of workarounds in for him. We ended up doing that. We've been pushing those fixes through the ecosystem. His main point at SIGGRAPH was that in principle he could have done this at the SPIR-V assembly level, but he got the compiler writer to fix it.
- KN: to reword Eric's proposal, because he has some level of control over the compiler, he can control what workarounds it has. No variability. He can pin to it, be sure that there won't be other workarounds breaking other things. Just because he's doing the workarounds at the high level doesn't mean it's not useful to know that *this* source code will result in *this* thing being given to the driver. You can guarantee this if you own the compiler and not if you don't.
- MM: the workarounds are driver-specific.
- DN: the workarounds are generic. You might be pessimizing, but you can make the tradeoffs. He's using the same source code and the same compiled SPIR-V for all the environments.
- KN: does he ship SPIR-V shaders?

    - DN: yes. And not the compiler.
    - DJ: what does he do to run on Metal?
    - DN: he has another entire path.
    - MM: that’s unfortunate for the web.
    - DN: that’s not his point.
    - DJ: what does he do for D3D?
    - DN: he has another compiler. It’s not his favorite thing.
    - DJ: where does he ship SPIR-V?
    - DN: you’ll have to discuss these details with him.
    - DJ: it’s interesting that we could be talking about 0.1% of Adobe’s deliveries, or 99%. We don’t know.
    - KN: just because the user base is small doesn’t mean that the implementation should be bad.
    - DN: is WebGPU forward- or backward- facing? Are we trying to target new apps? I asked him, what impact does shader language choice have on your decision to ship. It’s a forward looking question.
    - DN: also, these are Eric’s opinions and not Adobe’s in general.

WebGPU shader programming model

- slides
- RM: many of these questions are independent of language. For example, whether it’s SPIR-V or WHLSL, we’ll need a memory model and it might as well be the same one. A few points:
    - When you say “UB is OK for C++ because it’s part of a larger system”, it’s mostly because you only run C++ code that you trust. You don’t run untrusted C++ code downloaded off the Internet.
    - The main reason we’re so focused on no UB here is for safety reasons. For example, INT_MIN/-1, don’t want it to be necessarily one answer and have a cost on other GPUs. But can’t overwrite other stuff.
    - CW: so you’re saying there are UBs that are safe, and some that are unsafe?
    - RM: mostly, yes. Also, I would like the spec to point out to people that write the impls where they should be careful. Was a bit afraid that the spec has lots of UBs and it’ll say to implementers “make it safe”.
    - CW: DN was saying, we should put the line “here” - everything below has to be UBs / safe, and everything above should not be UBs. Your argument goes very much in the same way: we have to define this programming model and here are a few places to start.
    - RM: OTT does not talk about external memory model, only the local environment for scoping.
        - DN: got it, thanks.

    - DN: I think we agree more than was apparent before. Needed to use the right concepts and terms. Harder case: programmers will be able to create data races. Have to be able to say that you have one, and we have choices to make.
- CW: how do we move forward with creating a programming model?
- DN: we find all the areas of UBs in the languages we're talking about, and then resolve them. A lot of these will be brought to the larger group because they're API decisions. Have to just get concrete.
- MM: for discussion on SL and memory model, we're also interested in handling races. Would like to talk to CB about what fxc does.
- CB: you mean offline?
- MM: sure.
- KN: for example, writing to block memory in non-uniform control flow. That's the specific problem Intel brought up yesterday.
- CB: I don't know that fxc does anything fancy, but other tools inject things into your source code
- KR: Intel found issues where FXC would fail compilation when barriers are done in non-uniform control flow.
- CW: isn't this UB in the SPIR-V spec?
- DN: yes, similar to clang. If you can prove things it can generate the error. Harder cases are where it's dynamic, and you run into the halting problem trying to solve it. GPU-ASAN would be awesome. It's a common thing to identify statically constant control flow and we should do it.
- MM: when it determines non-uniform control flow does it determine every if-statement, or do value tracking?
    - CB: no idea off the top of my head.
- DN: trying to make the experience generally better but knowing you won't solve all cases?
- RM: for uniform control flow, looking at several possibilities. Currently, all shader languages say that if you have derivatives, etc. in non-uniform control flow it's undefined behavior.
    - 1. At compile time, have a static check that this happens.
    - 2. At runtime, have a check before the dangerous thing.
    - 3. Get a promise from the driver folks that the behavior's safe.
- CW: derivatives in non-uniform control flow are interesting. The ALU will contain unit values.
- MM: it's really dangerous potentially. Could be values from another program.
- RM: did some experiments with this. Tried taking it out of the spec, but problematic.
- MM: also problematic for GLSL.
- DM: HLSL compilation warns on that.
- CW: with cross-lane operations we might be able to do run-time checks.
- DN: and how far do you trust that? If you ballot, and then call a function in between?
- CW: you'd just enclose the thing.

MM: some things we should decide on.

- 1) In Metal, the same facilities are used for binding a bind group, and attaching a vertex buffer. Need for them to not fight each other.
    - a) Web authors can't use the same number for both
        - DM: at the spec level they're different namespaces.
        - MM: that's what I mean.
- (this conversational track somewhat abandoned)

CTS Harness

- Slides
- KN: Standalone CTS for now, eventually want to integrate in WPT and be able to run from their infra.
- KN: Need to be easy to write test: Vulkan CTS isn't good enough because tests are quite difficult to write.
- KN: Need to support the combinatorial explosion of tests, inspired in part by the way dEQP does combinatorial explosions of tests.
- KN: Tests are asynchronous and should be possible to run in parallel in the Web harbess
- KN: Tests run in node against Dawn, useful to test Dawn but also to have the CTS run in IHV test harnesses.
- KN: Had dawn.node working but it broke for some reasons so can't show that.
- KN: Obviously can run tests in the browser against WebGPU but we don't have WebGPU in Chrome so that doesn't work currently.
- KN: Eventually want to be able to trace the tests so that it is easy to run them in native implementation. Should be possible to write mostly traceable tests. Experience in dawn end2end tests shows that the case.
- KN: Written in Typescript
- KN: <shows the web page>. Three test suites for now:
    - An actual CTS
    - Unittests for the CTS
    - Examples of warning / errors, etc.
- KN: Each test suite is in a folder: If a file is .test.ts it is a test file, if just .ts things can be loaded from other modules to reuse code.
- KN: Each of these files is a JS / TS module and pulls in the dependencies.
- KN: There is a build step to run TS on the Web, it also generates a listing for the test suite and pulls tests / description for each file.
- KN: Test.ts files has a description imports then defines and exports test group.
- KN: Then tests are added to test group, tests use a fixture and there will be a "WebGPU" fixture. The test proper is an async function that takes the fixture.
- KN: Within the fixture there is a logging object that contains the status of the test.

- KN: The parameterization system can contain a list of parameters and a list of values for each. Params is serialized in JSON and becomes part of the test name. The test function can use the parameters from the outer scope.
- KN: This is very useful to test a variety of usage patterns concisely. Like “here’s the offset I write to” and “the value I expect at the end”.
- RC: How does one use the params in the test?
- KN: The second test example has a stub test and can use the params object as it is a closure.
- DJ: Example of how you know what combination of arguments you get?
- KN: Params is also passed to the fixture constructor. Not totally fleshed out yet, and you can generate many tests by doing a for loop. The params argument is mostly to set the test name.
- KN: Describes the options combinators.
- KN: Build system works the usual Javascript way. Can run in the browser with a build step, or in node without a build step.
- KN: The browser harness looks at the test listing and can load the test modules lazily.
- CW: Loads lazily?
- KN: Could but doesn’t right now. It would help avoid startup cost of loading every single file.
- KN: Have GET options to filter test, run automatically, etc.
- KN: Have a dummy WebGPU implementation in the browser for now (webgpu.ts defines the interface but doesn’t implement anything).
- KN: You can debug typescript in the devtools and support is amazing.
- KN: Can run things in Node, can run in Dawn or fallback to the dummy WebGPU. You get a JSON with a list of the results.
- KN: In node you can debug using the browser devtools by adding some node command line args and everything works too.
- KN: More in the CTS but wanted to focus on how tests are defined and how they run.
- KN: Questions, feedback?
- BB: we’ll probably look into integrating this into the driver CI.
- KN: great. Have done this with a few IHVs with WebGL. There is some value in running the tests through Chrome, but for driver CI it would be easier to not have the full Chrome stack. That’s why better to run with Node instead of browser implementation.
- RC: if you do this the Node way, how do you make it call into Dawn?
    - KN: there are a few ways to call into C/C++. First, bindings. That’s what I’m doing now. Ex., CreateBufferDescriptor.
    - CW: this is more to prove that the test suite can run on a real WebGPU impl, because this is as close as we have. In the medium term we’ll probably run in the browser. In the future people may want WebGPU-Node. Shouldn’t be that much work to maintain.
- DJ: wouldn’t you use this as your Dawn test suite?
    - KN: depends on how much effort to maintain. Currently only implements ~5 WebGPU entrypoints. If lot of work to maintain, maybe not.

  - CW: may be value running this in CI.
  - DJ: this is awesome, it’s what we asked for. Thought we wanted to not run with the whole stack.
  - KN: in browser CI we do want to run with the whole stack because that’s what we ship. Node stack could still be useful.
  - CW: WebAssembly is not an option right now because debuggability is poor.
- CW: would love to have feedback on the code itself. Maybe see if it can be wired into Safari’s impl? Currently using MapReadAsync, but the rest are JS.
- KN: there are some pure JS tests that just test parameterization, etc.
- DJ: what happens if your parameterization contains the expected result?
- KN: I do JSON.stringify on it right now, but probably will send it in as inputs / outputs.
- RC: is the code you write to get from JS object to Dawn, how complex?
  - KN: In Node, manually written C++ using N-API to turn V8 objects into C++. Messy but not so complex. In Chrome, autogen.
  - RC: how much of bindings can be shared?
  - KN: across browsers, probably none. In Chrome, we will use JS-C++ bindings autogenerated from IDL.
  - CW: we might have code generators eventually. It’s one-off code in general.
  - CW: as soon as we have multiple WebGPU implementations it would be great to start writing tests in this framework.
- DM: so to be clear we (gfx) would need to just produce bindings to node.js?
  - CW: yes.
  - KN: Or if we implement the same C interface, we could easily share the bindings.

Spec and CTS

- CW: would be great to have a Bikeshed suite, and test plan. When you make spec changes, you make a change to the test plan, too. That way we can review both in concert. Test plan should ideally be tracking tool for what tests are implemented.
- DJ: have only gotten Bikeshed ready; no other updates. Was going to take sketch IDL and start expanding it with prose around each object, function, etc. Boilerplate introduction, scope, etc. An example. Eventually, will be more than one example.
- DJ: things missing most: how to handle errors as you’re running through program. That’s not described in the IDL.
- CW: Kai produced general guidelines about errors doc, but have to be concrete. First few times we review these things, will be discussion about what’s missing.
- DJ: so the goal is to return objects even if they’re inactive and delay things as much as possible until the point where you need to validate?
- CW: yes. FF and Chromium will have remoting impls, and doing double-validation will be too costly.
- KN: it’s not really delaying, just when the GPU process will execute it.
- CW: spec will prob describe the error handling in terms of fences and visibility. May make things complicated, but will probably be a mode for synchronous validation.

- CW: let's get the spec skeleton in place and figure out validation errors. Kai has a proposal to change a couple of things. Goal of F2F is to reach a snapshot so we have interoperable spec / implementations.
- CW: do we have to have a call for spec editors?
- DJ: think it's OK to handle it like we have the IDL for now. If moved into an official W3C recommendation track we'd need names for editors.
- JS: once Bikeshed spec is ready, I can give feedback on IDL, etc.
- CW: if you have IDL feedback, please give it now as it's easier to respond to it.
- WebGPU snapshot tracker
- CW: would be nice to have a process so folks have to look at pull requests. Right now have a lot in limbo.
- DJ: could agree that a PR shouldn't sit open >1 week without 1 person from each browser looking at it.
- CW: SGTM. Also homework deadlines.
- DJ: should we discuss WG charter?
    - https://github.com/gpuweb/admin/pull/15
- DJ: took latest W3C charter, removed most stuff. Boilerplate, plus scope on CG (which is still a draft, oops), and put in lines about:
    - Majority of input to WG will come from CG. No major development will be done in WG.
    - Web Payments used to have a WG and "interest group".
    - KN: WebAssembly still has both CG and WG.
    - DJ: ok, will look to see how the charter works.
    - CW: will have to have lawyers look at it.
    - DJ: should be OK, taking things from less constrained environment to more constrained.
    - DJ: people should also review the CG charter. Preemptively predicted WHLSL and/or the programming model for SPIR-V.
- CW: GPU for the web group. Web Audio group specifies Audio* objects.
- DJ: agree. The only reason we aren't called the WebGPU group is that someone took that already. But we should drop the "Web" prefix from the APIs and object types.
    - CW: colloquially we should still call it the WebGPU API.
- CW: Rename to remove all the "state" suffixes from RenderPipelineDescriptor?
    - DM: might not be consistent to just remove "state".
    - RC: I think we should keep it as "rasterizationState". It's not rasterization or a rasterizer, it's its state.

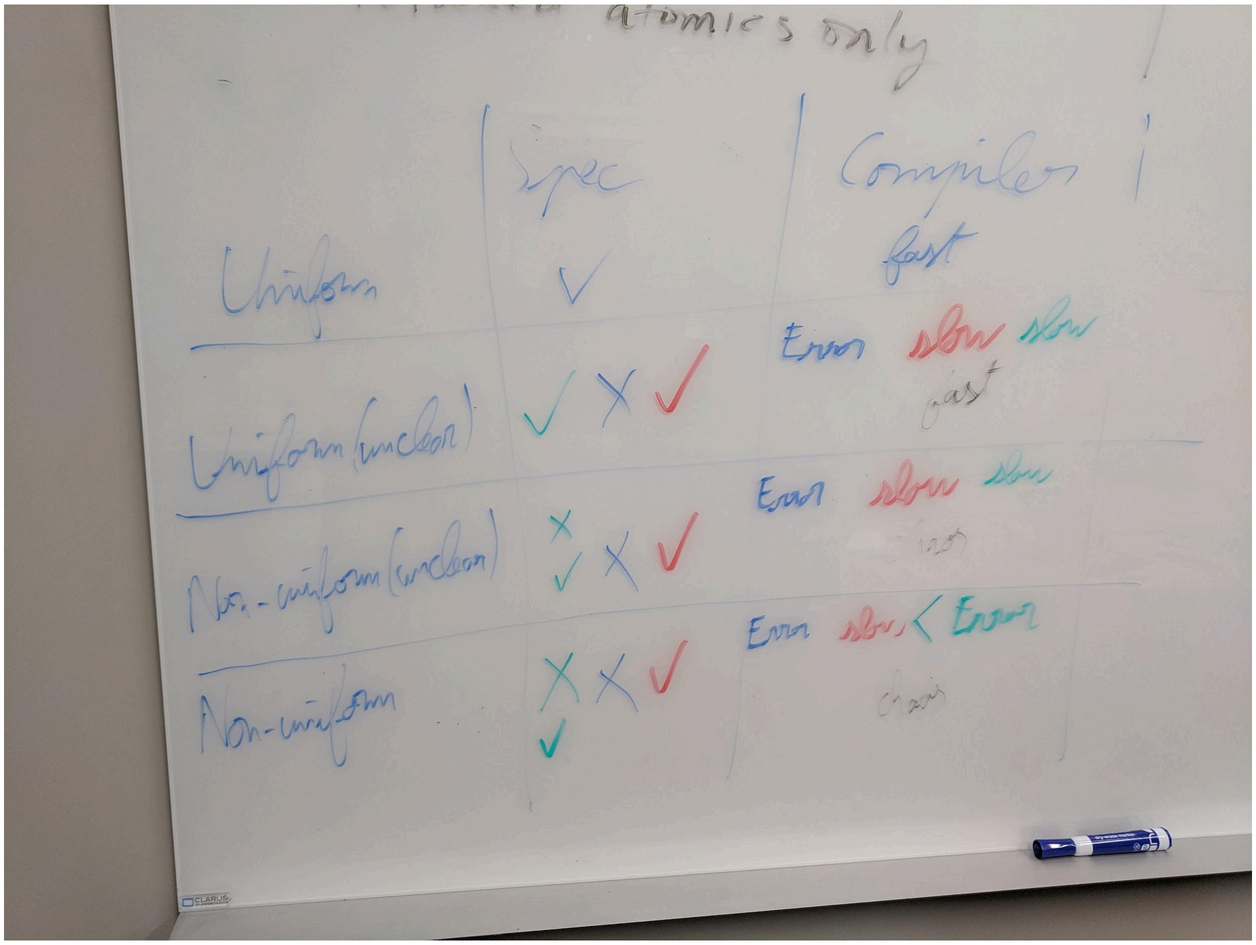
Shader programming model side meeting

Robin, David, Myles

- WebGPU will have only relaxed atomics, because MSL does.
- Control barrier with mem fence flags: always act as rel_acq.
- Vulkan does not have seq_cst.

- MSL: barrier that is only mem fence (not control barrier): must be in uniform control flow
- Myles: Don't have to decide now, but suggest not doing subgroup ops for MVP. Reconsider for future version.
- David: Agreed. IIRC subgroup ops were an extension to OpenCL 1.2. We can live without them for a long time.
- Races: Need to deal with them
    - Very hard / costly to detect them at run time.
    - Robin suggested following LLVM's lead and:
        - W/R race yields arbitrary bit pattern value read
        - W/W race yields arbitrary bit pattern value written; or an LLVM undef value (which can yield arbitrary bit pattern on each use)
        - LLVM community itself is arguing about semantics of undef ? https://lists.llvm.org/pipermail/llvm-dev/2016-October/106182.html is potentially relevant
        - Both W/W and W/R races have the same question of arbitrary bit pattern value vs "undef" that can evaluate to any arbitrary bit pattern value (and change which one over time)
    - David: In the limit: make all accesses of shared locations atomic accesses because they never induce a race with each other. It's probably stupidly slow.
    - Myles: We can run that experiment. Good to get the numbers.
    - Myles asked: what would you do if you had to make races safe in Vulkan.
    - David:
        - Write an extension defining certain bounded behaviour (such as above), while consulting IHVs to see if at the hardware level it probably will work out.
        - Then you write conformance tests. They act as a receipt.
        - Then you ask vendors to support the extension.
        - But you also have to write a bunch of stress tests (beyond CTS), to ensure better quality.
        - And since security is at stake you can't trust that implementations are perfect, so you have to qualify implementations yourself rather than rely on submitted conformance results.
        - This requires negative tests, many of them.
        - End result is similar even if you target old drivers that don't officially support the extension: you qualify them yourself.
            1. (And this is why military and medical equipment is expensive: because you qualify the system very extensively.)
- Uniformity:
    - Need uniform access to resources: buffers, textures.
    - Need uniform control flow for derivatives.
    - Need uniform control flow for barriers
    - (Need list of other ops where uniformity is required)

- (Myles thinks we don’t need arrays of resources in the shader. David suspects fine for MVP)
- Discussed motivation for the uniformity constraint: mental model is because you’re combining accesses to a shared hardware port.
- If you don’t have uniformity then you effectively do mutually-exclusive execution via a switch statement or loop. It’s a slow path, but effective.
- Discussed provability of uniformity property. For many cases static analysis will find correct answer for sane code. Always will be cases where your algorithm is fooled (halting problem).
- Discussed the design space:
    - Analysis cases:
        1. Provably uniform (for derivatives, resource access …)
        2. Uniform but unanalyzed
        3. Non-uniform but unanalyzed
        4. Provably non-uniform
    - Specification can either: Permit or Forbid (either by required error result, or yielding undefined behaviour)
    - Compiler (implementation) can: Error, Slow code, Fast code, Chaos code



atomics only

	Spec	Compiler
Uniform	✓	fast
Uniform (unclear)	✓ X ✓	Error slow slow fast
Non-uniform (unclear)	X ✓ X ✓	Error slow slow [illegible]
Non-uniform	X ✓ X ✓	Error slow < Error chaos

- Conservative path (blue): Spec permits only shaders with provably uniform. Compiler matches analysis from spec. Generates only fast code, and results always correct.
- Loosest path (red): Spec places no uniformity constraint. Compiler generates code in all cases. But only provably uniform case can run fast.
- Forgiving path (green): Spec forbids provably non-uniform cases. Compiler generates code in all other cases, but only provably uniform case can be fast. (comment by Robin: I suspect there are very few programs that are provably non-uniform, for any reasonable analysis)
- Simple compiler case (black): Spec permits only uniform case but does not specify analysis; non-uniform behaviour is chaos case. Compiler does no analysis, assumes behaviour is actually uniform, only generates fast code appropriate for uniform. At execution time, live with chaos if behaviour is in fact non-uniform.
- Discussion:
    - Often a spec will forbid the slow path cases to avoid unbalancing performance or perception of IHVs. (Avoid performance cliffs) E.g. desktop GPUs can spend more silicon/power to make the more complex case run fast.
    - Bias is toward enabling features that are uniformly good. Expand that set over time as implementations increase functionality.
    - There is a cost in spec complexity to specifying an analysis; from above, requiring a matching analysis in the compiler increases implementation complexity.
- Robin had written an rules in the WHSL spec to analyze uniformity in a conservative manner (i.e. one-sided error: if it says “uniform” for a value or control flow, then it is correct, but might say “maybe” when it still is actually uniform):
    - Analysis lattice: bottom → Uniform → Non-uniform (top)
    - Seed with constants and uniforms as Uniform. Certain builtins as non-uniform. Variables are untagged.
    - Then use conservative combining rules (like type judgements)
        - Conservative, e.g. if X is non-uniform in, then Y = X - X is uniform in real life (since it’s zero), but combining rule for subtraction is too simple to look at the actual values, so it judges Y as non-uniform.
    - Use rules about reconvergence (borrow from SPIR-V divergence / uniform reconvergence rules)
    - Run analysis… Profit.
    - Analysis was removed from WHLSL, as too complex for the spec.
    - TODO(david): Look at it for a second opinion.
        - https://github.com/gpuweb/WHLSL/commit/5c97af8cbfb8b666ce0df731b65d93b26ce120a4

                - https://github.com/gpuweb/WHLSL/commit/85fb37b2845eaa5c4fcb7c6f8d8c7315f71b4c3f
            - Myles: Analysis was in terms of WHLSL. If WebGPU prog model requires that analysis then would need to redo analysis in terms of SPIR-V.
            - David: Yes. Expect it's not fundamentally different.
            - Robin: Structured control flow in SPIR-V?
            - David: Yes, in Vulkan and proposed for WebGPU.
            - Robin: Not expecting a fundamental blockage there.
            - TODO(david): Replicate static uniformity analysis in terms of SPIR-V.
        - If a barrier is placed in non-uniform control flow: Must the implementation hang?
            - Good question. Sounds painful?
        - That implies negative testing here too.
        - Myles: Our HW test friends are not going to like this…

- At some point David and Robin discussed the complexity introduced by having different resources actually alias the same underlying storage. Makes it hard to infer races. This is one reason Vulkan has the concept of a "reference" to represent the I/O interface to a resource, and to manage synchronization/availability/visibility across that interface.
    - Expect to recommend to the CG to disallow aliasing of the same storage via different references to the same pipeline execution instance.
- TODO: Enumerate design points to determine: include "undefined behaviour" list, optional capabilities, obvious parameterization. But the hard parts are races and uniformity.
- Likely can progress between Robin and David and report back to the CG.

In a later email, Robin wrote:

> Here are the two commits in which I removed the static analysis of uniformity from the WHLSL spec:
> https://github.com/gpuweb/WHLSL/commit/5c97af8cbfb8b666ce0df731b65d93b26ce120a4
> https://github.com/gpuweb/WHLSL/commit/85fb37b2845eaa5c4fcb7c6f8d8c7315f71b4c3f
> … and details about the tricky bits.

Also:

> there is a third significant issue beyond races and uniformity: fast-math.
> Metal (and I suspect Vulkan and HLSL) enable the fast-math flag of LLVM:
> https://llvm.org/docs/LangRef.html#fast-math-flags
> This might be problematic because of the two sub-flags:

`nnan`

> No NaNs - Allow optimizations to assume the arguments and result are not NaN. If an argument is a nan, or the result would be a nan, it produces a poison value instead.

`ninf`

> No Infs - Allow optimizations to assume the arguments and result are not +/-Inf. If an argument is +/-Inf, or the result would be +/-Inf, it produces a poison value instead.

Poison values in turn can lead to undefined behaviour if they flow into observable stores (or some other things, the spec is a bit fuzzy about them).
I am a bit afraid of a carefully crafted program generating NaNs/Infs to convince LLVM to erase bounds-checks (for example). I am not entirely sure of what can be done about this: we can check the cost of removing these two flags in Metal, but from what I understood on Tuesday, the Vulkan implementation on Android phones is not easily updatable. And inserting checks at runtime to prevent NaNs and Infs from being created sounds both tricky and horribly slow.

Agenda for next meeting

Agenda was briefly discussed in
https://docs.google.com/document/d/1g5wLCzJFZNIqKUhzoygOc8V53voGGccu-NPd5kqnYz4/

Second Day's Notes