So as to not be anonymous animals, the doc is shared for writing with the google accounts that are invited to the meeting. If you can't edit let cwallez@google.com know. Also be sure to be in "Edit" mode and not "Suggest" mode.

GPU Web 2020-06-22 VF2F Day 1

Chair: Corentin
Scribe: Ken / Austin / Corentin / Dean / others
Location: Google Meet

Tentative agenda

To be determined, add your topics below:

- ~~Status updates~~
- ~~Roadmap to releasing an agreed upon MVP (Corentin)~~
- ~~Extensions mechanism (integrated in the spec? Experimental vs. community approved.) (Kai)~~
    - What about pipeline statistics?
- ~~Texture view format reinterpretation #744 (Myles / Corentin?)~~
- ~~Add time query on GPUCommandBuffer #870 (Yunchao?)~~
- ~~CreateReadyPipeline #871 (Kai)~~
- ~~Switch primary git branch to "main" (Corentin)~~
- (Rest moved to Day 3)

Agenda/Minutes doc for Day 2

Agenda/Minutes doc for Day 3

Tentative Schedule (in PT):

- Day 1
    - 11:00 AM: Welcome + schedule changes
    - 11:10 AM: Status updates (Apple, Google, Intel, Mozilla, Microsoft)
    - 11:50 AM: 10m break
    - 12:00 PM: roadmap to releasing an agreed upon MVP.
    - 12:15 PM: Switch primary git branch to "main"? + WG
    - 12:20 PM: Texture view format reinterpretation #744
    - 12:40 PM: CreateReadyPipeline #871
    - 12:50 PM: break
    - 1:00 PM: 1:30 PM: Extensions mechanism.

    - 1:20 PM: Case for optional GPUBindGroupLayoutEntry entries #851
    - 1:50 PM: End of Day 1
- Day 2
    - 11:00 Thank Ken for the scribing effort
    - 11:00 AM: ImageBitmap discussion
    - (filler): Add time query on GPUCommandBuffer #870
    - 11:50 AM: break
    - 12:00PM: Start of WGSL part (schedule TBD)
- Day 3 (overflow API and WGSL)
    - Out-of-memory for buffer mapping #872
    - Copying of depth24plus formats #652 (yes, again)
    - Introducing a GPUEncoderBase (Dzmitry/Brandon)
    - Fancy render pass attachment load/store/readonly (Kai/Brandon)
    - Where to store encoder state in the spec? (Kai/Brandon)
    - Can we simplify the data uploading primitives to 2 at most? (mapBuffer, createBufferMapped, writeBuffer) (Ken)
    - Development-only API features (#814)

Attendance

WIP, it is the list of all the people invited to the meeting. **In bold the people that have been seen in the meeting:**

- Apple
    - **Dean Jackson**
    - Fil Pizlo
    - **Justin Fan**
    - **Myles C. Maxfield**
    - Robin Morisset
    - Theresa O'Connor
- Google
    - **Austin Eng**
    - **Brandon Jones**
    - **Corentin Wallez**
    - **Dan Sinclair**
    - **David Neto**
    - **James Darpinian**
    - John Kessenich
    - **Kai Ninomiya**
    - **Ken Russell**
    - Shrek Shao
    - **Ryan Harrisson**
- Intel
    - Brandon Jones

    - Bryan Bernhart
    - **Yunchao He**
- Microsoft
    - Chas Boyd
    - **Damyan Pepper**
    - **Rafael Cintron**
- Mozilla
    - **Dzmitry Malyshau**
    - **Jeff Gilbert**
- Joshua Groves
- **Matijs Toonen**
- **Mehmet Oguz Derin**
- **Pelle Johnsen**
- **Timo de Kort**

Status updates

- Apple
    - Not much progress since last F2F
    - Would like to start converting WHLSL code into WGSL code, but haven't done much on that yet.
    - Mostly busy on other work due to WWDC
    - Myles is very excited about WGSL, and has been putting in most of the pull requests on our side
    - Primary goal: someone can download a Safari Technology Preview, turn on WebGPU experimental, and look at a demo on webkit.org that looks like the community group's API
    - CW: there are a lot of other API changes in Safari, is there a roadmap?
    - DJ: those are easier. On Justin's plate.
- Google
    - Implementing a lot of new features in Dawn. Sent status update 1.5 months ago, since then implemented a bunch more stuff.
    - Since last F2F: exciting stuff is development on Tint.
    - Since last week: got prototype impl in Chrome. Can do a simple demo!
    - http://hello-webgpu-compute.glitch.me/hello-compute-wgsl.html
    - Thanks to Dan, David and others who've been working on Tint!
    - DM: is it going through SPIR-V?
    - CW: yes. Flow: WGSL -> GPU process -> compiled with Tint into SPIR-V -> SPIR-V cross -> MSL -> Metal driver.
    - CTS updates:
        - KN: Development is proceeding
        - KN: Continuing to maintain main branch and GLSL-depdendent branch
        - KN: # tests on GLSL branch hasn't grown much

        - KN: However, eager to stop having an extra branch. Painful to maintain two branches.
        - KN: Want to make a call for WGSL to focus on features that will get it to a usable state - an MVP - which will allow use in the CTS. Even if those things change again in the future. Want implementations to be usable and syntax/features to be "close enough" to get rid of GLSL branch.
    - Shading language update
        - DN: tomorrow we'll be presenting to WGSL side of things about things we think are blocking that transition.
        - DN: Our team's converting Dawn samples to WGSL to see things we might not have thought of.
        - DN: Flow from SPIR-V to WGSL is getting better so that things will get automatically translated for this transition. Getting closer to round-trip milestone.
- Intel
    - Yunchao presenting
    - After bind group optimizations on D3D12 from Bryan @Intel, WebGPU Babylon.js demo is 40 FPS, WebGL version is 30 FPS!
        - Finally faster!
- Microsoft
    - Making number of contributions to Chromium's Dawn backend
    - Better context loss handling
    - General cleanup
    - Optimizing ticking, updating things for various backends
    - Code reviews, attending WGSL meetings, providing feedback
- Mozilla
    - DM: Slide presentation overview of what works. Hacks blog article
    - DM: Waiting for textures to be part of WGSL before experimenting with direct WGSL ingestion.
    - Demos in-browser and native demos
    - If folks use wgpu-rs, they can get WebAssembly support for free!
    - Servo can render things. The triangle for example. That's a milestone.
    - CW: amazing to see. Great that FF and Chrome are interoperable. As soon as WGSL has textures will be fantastic.
    - DM: miracle to me that in the past 2-3 months, these examples haven't broken!
    - slides
- WebGPU spec update:
    - slides

Roadmap to releasing an agreed upon MVP.

- CW: Chromium side is feeling some pressure to ship.
    - Made good progress, having people experiment with the API

    - Shipping an MVP is important. Have had some feedback from implementers and people experimenting, but they haven't gone very far. No feedback from large applications targeting WebGPU yet. To get that feedback, we need to give them something reasonably stable and works across multiple browsers - i.e., an MVP. Then we'll get investment from people
    - MM: do those people need it to be on by default:
    - PJ: that would be best.
    - CW: it would help. Chromium has a mechanism to do that - "origin trials" - turn it on by default but off again after a certain number of months. Should still be representative of the final API though.
    - JG: my concern is "un-shipping" something from the web. Working with WebXR group now, shipping something to the web, and then having to be really careful about changing things.
- MM: what timeline did you have in mind? WGSL is pretty far from ready.
- CW: yes, we don't want to ship without WGSL.
- DN: have a lot of spec writing. Bones are all there. Feature-wise, we're close to shippable things. Know there are things Apple wants that we disagree on whether they're necessary.
- MM: I'm concerned about that because shipping a product without a spec that remotely resembles it, is concerning.
- DN: I think we can make a lot of progress on spec text in the coming months.
- JG: I recommend trying to relieve the pressure of getting experimental stuff out there is by providing builds or providing flags.
- CW: it's hard to get important customers to care about things that are behind flags.
- JG: I don't believe that if we have imp't customers investing in this that we'll change it. You only get one shot at having people spend a lot of time on something. If we made e.g. min bindgroup size a requirement, requiring rewrites of engine, they wouldn't want that. They just want the finished product so they can work on it, but we don't have that.
- CW: how do we get to something big customers can use?
- JG: MVP.
- CW: what are the mechanisms by which we can get to MVP?
- JG: if you want to ship it, we have to ship it - in order to ship, you get to an MVP that we could ship, and then finish it up in to a V1 that we actually ship. We're on trajectory for that, don't think there's a shortcut we can do which won't misrepresent our offerings to people.
- MM: what sort of outcomes do you want from this conversation?
- CW: mechanically, browsers can do what they want, but it's bad practice for them to do that. Hoping to motivate people to make list of features they want for MVP / V1 and we call that WebGPU V1. We can add endless features / optional features to WebGPU, and standards can fall into the trap of endless polish, but I don't want that for WebGPU.
- MM: we can certainly make a list of features we want for MVP. Prioritize certain things. Would help us motivate our work if we knew the timeline you were looking for.

- CW: don't think WebGPU V1 can ship before end of this year and that would be aggressive. Hoping we can agree on things in the spec and CTS. Maybe Q1/Q2 next year for shipment.
- MM: we can work with that.
- CW: If we say H1 2021 then WGSL will have everything we need.
- MM: as long as these timelines are fuzzy we can work with that.
- PJ: from app dev's perspective it sounds good. As long as it isn't another 3 years.
- CW: that's what we're hoping for for WebGPU. For MVP / V1 feature set (depending on what we call it) should we make an issue for tracking these things?
- MM: there's a WGSL project with a milestone for MVP. Could track it that way.
- CW: WebGPU editors, should we have a Github milestone for MVP?
- MM / DM: yes.
- CW: we'll start populating that soon / this week. Shouldn't be that much.
- DM: thought the API was pretty close and that we were now discussing lower level things like texture format reinterpretation.
- CW: this is a feature. Multi-threading. There are still some special parts to figure out - not very many, but a couple.
- DM: OK.
- CW: feels like we're 90% of the way there, but has felt that way for the last 1.5 years.
- MM: think the biggest hurdle is the shading language. Agree with DM that the API is pretty close.

Switch primary git branch to "main" (Corentin)

- CW: current naming has offensive / negative connotations
- Any feedback on renaming it? May require closing all open PRs.
- KN: we can leave the old stale PRs open against "master".
- DM: you can switch the target of the PR. Can just create a new branch from master, leave it as the default, and the PRs will update themselves.
- KN: I don't know how to do that.
- PJ: you can also set up branch rules, block merges to the old branch.
- MM: don't want two competing main branches.
- CW: I'll volunteer to do that for the admin repo. KN will do it for the CTS.
- CW: any volunteer for gpuweb?
- DM: I can do it for the main repo.

Texture view format reinterpretation #744 (Myles / Corentin?)

- MM: you can create a texture with a particular pixel format, and then create a view. The pixel format of the view doesn't have to match that of the texture itself.
- MM: WebGPU has textures and views. Do we want to expose this to WebGPU? If so, what are the rules?
- MM: Kai made a wonderful spreadsheet with all of the 3 APIs. Very illuminating.

- MM: Metal one was based on the docs which have some errors.
- KN: I don't understand what the differences are yet but we'll get to it.
- MM: the difference is: everything on the chart was right, but all green boxes should be marked Y instead of Free, except those on the diagonal (????)
- KN: so you can't change the format at all without using the MTLTextureUsagePixelFormatView flag?
- MM: right.
- KN: even between regular and sRGB?
- MM: correct.
- KN: is that from the Metal debug layers?
- MM: yes.
- KN: I'm suspect that the layers are wrong because the documentation is pretty clear.
    - > *Don't set this option if the texture view just needs to read the component values in a different order. Instead, create a texture view using a swizzle pattern to specify the new order. Similarly, don't set this option if your texture view only converts between linear space and sRGB; for example, if your texture uses the MTLPixelFormatRGBA8Unorm pixel format and your texture view uses MTLPixelFormatBGRA8Unorm_sRGB.*
      -- MTLTextureUsagePixelFormatView (link is down because of WWDC - Google cache link)
- MM: in Vulkan, CW posted there's this ImageFormatCreateMutable flag. Is it without that flag, the pixel format of the view has to match the texture?
- CW: yes.
- MM: then that's how Metal's flag works.
- CW: given the spreadsheet, Metal's reinterpretation might be slightly broader than Vulkan's.
- MM: really? The spreadsheets are exactly the same.
- MM: in D3D it seems there are no Free conversions, and those are a subset of those in Metal / Vulkan. Thinks it leads to natural result:
- MM: no reinterpretation is always possible (covers diag of matrix)
- MM: if you add a creation flag to your texture, reinterpretation's possible in some cases, where those cases are the intersection of the 3 APIs.
- CW: that works, would be a good API that's easy to reason about. In talks with Jason Ekstrand from Intel (direct via email), Vulkan CreateMutableFormat bit is a big hammer. Lot of texture reinterpretation that's free in the hardware. Free to reinterpret RGBA8 int/uint/snorm. But sRGB for some Intel HW generations, it has to disable compression completely. Unfortunate, because they'll want to reinterpret between formats that aren't sRGB. Will mean the toggle will have worse perf on that hardware. It's a data point for one HW vendor, but others have similar issues.
- MM: you mentioned two bits in your post. CreateMutableFormatBit and CreateBlockTextureViewCompatibleBit also. Is there something Intel suggests to use instead?

- CW: the proposal for reinterpret was #811 . Vk introduced extension, core in 1.2 - during texture creation, list all compatible texture view formats. Driver can decide whether needs to disable compression or not. On Intel e.g., sRGB presence would disable compression. On Metal, RGBA8 / BGRA8 could be handled with swizzle instead of mutable format bit.
    - VK_KHR_image_format_list.
- https://github.com/gpuweb/gpuweb/pull/811
- VK_KHR_image_format_list
- DM: the swizzle won't affect copies - only the shader.
- CW: copies are not affected by texture view formats, because they take in textures.
- DM: OK.
- MM: this seems like possibility of expansion for the future. Catering to one specific device manufacturer we can do in the future, but for the MVP we should focus on the 3 native APIs.
- CW: this extension was promoted to core Vulkan 1.2 - went through standardization with all vendors. Can also have benefits for Metal / D3D12.
- MM: downside is having to know up front all the things you'll use the texture for before you do it.
- CW: agree it's a more complicated API to use.
- MM: think there's a possibility for these APIs to work together. ...
- KR: Would it makes sense to be more restrictive at first, this would be the same as the rest of the WebGPU API where things are specified upfront.
- MM: In the proposal I made, you also predeclare you will use some amount of reinterpretation. The difference is how much detail is provided. Is it a big switch like in Vulkan 1.1 and Metal, or is it many many switches. Seems like one switch that matches the Vulkan that exists, and Metal, is going to be better, especially when they have the same semantics.
- KR: Knowing there are performance pitfalls for some of them, it seems better to use the benefit of foresight. Don't want to having performance pitfalls or problems in the future.
- MM: Also possible for Intel to release a new version of a chip that does not have this problem.
- CW: The extension has been promoted to Vulkan 1.2. It is part of the future spec.
- MM: Vulkan 1.2 on Android essentially does not exist in the marketplace.
- CW: it's coming eventually. The Vulkan WG designs the API with input from all HW vendors.
- MM: I'm making the same kind of argument. Vk 1.1 had one flag, Vk 1.2 had additional flexibility. That's the proposal I'm making.
- CW: at this point I think both proposals are reasonable. Either doing view formats now or the single boolean now and view formats later, either way sounds good on my side. I'd like to have input from other people in the room.
- DM: if we have single boolean, then rules will be D3D12's since they're the intersection?
- MM: think that'd be the way it ends up. Don't think there's anything D3D12 allows that Metal/Vk don't.

- DM: we have little problem, in D3D12 the view casting rules depend on what you'll be using it for. RGBA8 can be used a Int32 only for UnorderedAccess views, not shader resource views. And you can swizzle ShaderResourceViews, but not UnorderedAccess views. We don't have that distinction in WebGPU. Not straightforward for how these rules will be encoded in the current API.
- MM: I'm going by the spreadsheet, not much experience with D3D12.
- …(question)
- MM: I thought we were going to ignore the yellow boxes and only pay attention to the green ones.
- KN: the yellow boxes came from a comment from DM. Special section in D3D spec. For UAVs in particular, can make R32 that can point to bunch of formats. Can't do that on any other backends.
- MM: OK, so ignore yellow box.
- DM: thought we can do this on all other backends. You'd want to do that, actually.
- KN: suppose it could enable some stuff.
- MM: to enable the yellow box we'd have to decide between SRVs and UAVs. Is it worth exposing that distinction to WebGPU?
- KN: not sure we have to add anything to the API. We don't have texture views which are SRV/UAV, but we do know how they're bound. So would be compatibility between BindGroup / BindGroupLayout.
- DM: think it'd be easiest to have this distinction on the texture view. Read-only flag moving to GpuTextureViewDescriptor.
- MM: when you want to attach texture view to BGL, and BGL is marked as R/W (not R/O), then this type of reinterpretation is possible?
- DM: right. Can use U32 view of RGBA8 in R/W storage, or write-only storage.
- MM: sounds reasonable to me.
- RC: does this mean there'll be some platforms where content will work, and surprisingly not work on others?
- CW: that's not the case for any WebGPU proposals.
- MM: we'd come up with rules that would work on Windows, and enforce those rules on macOS / iOS.
- RC: that's my preference. Relaxed rules can be an extension.
- MM: to accept the yellow box, have to also move R/O into BGL.
- DM: no, I was describing one in TextureViewDescriptor. View is only used as read-only. Realize it's incomplete. readOnly:false and want to reinterpret RGBA8 as R32, can't do that with SRVs in D3D12.
- CW: mechanism could be, texture view has all usage of texture by default, but can say: usage is only storage, and if you do that, can also reinterpret per the yellow box.
- DM: that's not what D3D12 has. Storage in D3D12 …. but we're using SRVs (?)
- MM: until new proposal is made, we shouldn't accept the yellow box.
- DM: sounds OK

- CW: how about: go with boolean flag that gives D3D12 green boxes. If hardware vendors complain, we can later add view format-like mechanism. TBD how we allow the yellow box. Would like to allow it, but don't know how yet.
- MM: sounds great.
- KN: found the link for that piece of doc on Metal documentation. Says: "don't set this option if the texture view just needs to read the component values in different order, or needs to convert between sRGB and not". Very explicit you don't need this for sRGB.
- MM: the ordering's for swizzling. I could be wrong about the details of sRGB vs. non-sRGB. If you try to write the program and run it, the machine will not let you run it.
- KN: the debug layers might not technically be correct.
- KR: if that's the case we could file a bug about the debug layers.
- MM: or against the documentation.
- MM: when the docs are back up I'll have some stuff to raise tomorrow or the next day.
- CW: can you also work with documentation or debug layers to figure out the real rule?
- MM: yes.
- CW: does the result of this change the discussion we just had?
- MM: no. We'll have to run with the debug layers on, so if they're more restrictive than the docs we have to live with that.

CreateReadyPipeline #871

- KN: idea to reintroduce this has been floating around for ~1 year. I originally proposed to replace createReadyPipeline with ErrorScope. Use ErrorScope with filter=none to say, wait for enclosed commands to finish. If you include createPipeline with that, can wait for ErrorScope to resolve, and it'll tell you pipeline's finished compiling.
- KN: This is the only reason the GpuErrorFilter=none exists. There aren't a lot of cases where you want to wait on resource creation, some object in order to know it's ready to use. Most creation is fast. For large memory allocations, you'll use out-of-memory ErrorScope anyway.
- KN: this PR reintroduces createReadyPipeline - we had this earlier. Returns Promise. You'll know you won't use pipeline before it's ready. No hitches when you wait on a Pipeline that's being compiled. Can take ~1s to compile a pipeline, so this is an important hitch for devs to avoid.
- MM: does this do anything that devs can't do otherwise?
- KN: no.
- MM: so this lets devs do the right thing - more natural, idiomatic to write the correct code (writing stuff before the pipeline's done compiling).
- KN: Two benefits. One is what you said, and the other is that it simplifies the concept of error scopes. We only care about returning an error scope when we know a validation or OOM error has occurred. We don't have to make sure pipeline compilation gets included inside the error scope.
- CW: more specifically: without createReadyPipeline, engines have incentive to move pipeline compilation off-thread, but then get into issue where one ErrorScope

encompasses work on multiple threads. Pipeline compilation result can finish out of order. If we want to put this in the spec it's hell, if we don't it's hell for the impl. createReadyPipeline allows saying in idiomatic way, this can be done async, and we don't have complexity either in spec or impl.

- MM: your second point Kai is that it enables simplification. Think that's only possible if we get rid of our current way of compiling pipelines, is that true?
- KN: By having createReadyPipeline, we can remove the requirement that surrounding the creation in an error scope will wait for the pipeline to complete. We only need to know that validation completed. Admittedly this is not that different, but it removes the requirement.
- MM: so if we accept this PR the only way to compile a Pipeline will involve a Promise?
- KN: the only way to compile async, yes. ErrorScopes don't block on the internals of createPipeline.
- CW: The reason we can't remove the synchronous version is that a lot of applications out there are not going to be WebGPU first, and the first time they need a pipeline, they will compile it because they work in a very GL or D3D11 way. It's bad for them because they have hitches. But we can't remove it really. Even Vulkan and D3D12 applications use this today. createReadyPipeline also removes one of WebGL's biggest problem where they compile a ton of shaders at load time and lose the context. With Promises, they can pace how fast they compile stuff.
- MM: for app that wants to compile synchronously, can they just say Promise.then()?
- CW: that's not synchronous because if you do that inside requestAnimationFrame the .then() runs after the frame.
- MM: you're talking about the case where the pipeline compiles quickly? Then .then() still executes before the "update the rendering step" though.
- CW: the pipeline can go to the GPU process asynchronously. When GPU process is done it sends back the signal that it's done. Could be 10 frames from now.
- MM: app that wants to do this, it's a sunk cost.
- CW: that's the intent of createReadyPipeline. Not practical to just allow this. There are definite use cases for pipeline compilation inline / synchronously.
- MM: createReadyPipeline sounds awesome, think we should use it everywhere and get rid of the current way of creating pipelines. Makes it more likely authors will write correct code.
- KN: there are plenty of cases where you have a trivial pipeline that you want immediately. If your engine isn't architected around async then it's hard to do correctly. There' an example in the PR which renders the object all-black temporarily. That use case is useful for an app that can't just decide to render frames, or an object.
- KR: In WebGL, we've heard from Unity that they simply cannot deal with a fully async version. They need the synchronous version for correctness of some games.
- CW: another example, want screenshot with filters applied. Screenshot's rare, so you wouldn't precompile the pipelines. You'll do it on demand, synchronously. It's OK if it hitches.
- KN: you want the snapshot to be of that frame, and not a later one.

- MM: in order to use one of these pipelines you have to put it in cmdbuf and submit it. If you createReady Promise, then do .then(), you won't submit any work until after your pipeline's created.
- KN: in screenshot case for example you'd capture data / texture contents at a later time.
- CW: I'm not even sure - the Promise.then() won't prevent rAF from getting called. You'll still submit later partial frames from e.g. 10 frames before. createReadyPipeline is the good / recommended way to create pipelines, but unfortunately not sufficient. Sometimes you need synchronicity, and sometimes porting an app that behaves this way.
- MM: Having both paths is OK with us.
- CW: OK, sounds good, thank you.

Extension mechanisms

- KN: already covered a lot of this last week. Generally, my vision is:
- KN: have a concept of multi-browser extensions, like Khronos' multi-vendor extensions.
- KN: if agreement from community group to add an extension, which is always optional and doesn't have to be implemented by all browsers / hardware but is widespread enough that it can go into core spec, it should go into core spec.
- KN: all multi-vendor & ratified extensions go into same document. Not sure about single-extension docs.
- CW: there are multiple builds of the spec, some with no extensions, some with only KHR extensions, and some with all extensions.
- KN: think we'll have very few single-vendor extensions. Probably only highly experimental stuff. E.g., first prototype of a ray-tracing extension would be carried only on single browser, likely.
- KN: if something goes into spec, think it'll be unlikely to be removed or changed significantly. Others can be drafted / experimented with quite a lot.
- KN: proposal:
    - Incorporate optional features into spec. Considering renaming 'extension' to 'feature' similar to Vulkan.
    - Creating device / adapter: i require these features / these limits.
    - In both spec and with visual styling, indicate whether a part of the spec is blocked by a feature being enabled.
    - Pipeline Statistics Query, for example. Methods would be demarcated to indicate they're part of an extension.
    - Then checkbox(s), show / hide these extensions. Like Core Vulkan spec, and Core Vulkan Spec + extensions. Want something similar for WebGPU.
- DM: little concerned about merging the semantics of extensions and features. Extension = a document that extends the API. Can have multiple features, technically.
- KN: There would always be feature flags for enabling the various features. Documents that describe extensions would use feature flags. The IDL doesn't have a concept of

extension, only feature flags. An extension is a document that describes new feature flags that you can enable.

- DJ: By feature flag you mean that you check the interface to know if something exists?
- KN: no. For example, compressed textures. Say we wanted a separate extension doc for PVRTC rather than putting it in main spec. That extension spec would say “this adds a feature called PVRTC, and you enable it the same way you enable BC compressed textures (which are in the core spec) by adding it to the features array when you create your device”.
- CW: sounds like everybody’s happy?
- KN: if everyone’s happy with the direction I’ll make it into a real thing and put it up for review later.
- DM: will the features be completely independent, or will there be interdependencies?
- KN: expect that some features will require other features. Example, loading U16 from buffers is separate from expressing U16 in shaders. In order to load U16 from buffer into a U16 variable, you need both.
- KR: made point about WebGL’s dynamic extension enabling and the fact that it’ll be easier to spec everything upfront. Reject attempts to enable one feature without the one it depends on.
- KN: another example: say we have a feature for rendering to depth24unorm, and another for copying from.
- KN: I think the mechanism we have in the spec today works well for this.
- DM: will enabling a feature affect other features? Remove functionality from other features?
- KN: it’s possible i suppose. I’d like it if it didn’t, but seems possible.
- DM: if an extension is a bunch of features, and they’re additive, that works great. But if an extension changes the core behavior of the spec that becomes problematic.
- CW: is there precedent for this in Vk?
- DM: yes. VkPortability. It’s a good example.
- CW: in that case, we talked about WebGPU compat. If it’s removing stuff and the magnitude of removal is like VkPortability, that should probably be a device creation flag, or feature level. Not in this features list.
- KN: think that’d be ideal.
- DM: but features are device creation flags.
- KN: yes. Just give it a different name.
- DM: maybe we should schedule some homework, look at other Vulkan extensions that make different changes to semantics than just new features.
- KN: looking at feature flags in Vk, not just extensions, then robust buffer access is an example. Required to be supported, not enabled by default, changes behavior of other stuff. In that case it restricts the behavior of other stuff, so non-breaking in that sense.
- DM: everything sounds good, think we should proceed.

Case for optional GPUBindGroupLayoutEntry entries #851

- DM: we have BGLs. The contract between resources and pipelines we create. 3 fields of BGL are not required by either of the native APIs. Texture component type, storage texture format, min binding size.
- DM: we decided min buffer binding size has to be optional.
- DM: wanted to raise awareness: we need to know where we draw the line between optional and non-optional. Theoretically, all could be optional.
- CW: think the issue is not about changing which ones are optional, but rationale why some are optional and not others?
- DM: yes. If no good rationale, should be optional.
- CW: Idan's benchmark for minBufferSize shows there's a cost when you leave it optional and do a draw-time check. Instead of a comparison, it's an equality check in other entries that could be optional. minBufferSize is a big thing to ask people for.
- CW: reason for defaulting some: minBufferSize is likely to change very often - you add one more uniform to this shader, now have to update a size far away in the code.
- CW: for the others - think it's unlikely to change after the first time you set it. Texture component type: if you have a uint texture, that's what you've got. Different than e.g. a float texture. Easy to know what it is when you make the BGL.
- CW: storage texture formats - sort of the same reasoning. A bit less used than regular textures and UBOs. Maybe that could change with WebGPU since WebGL doesn't have them.
- DM: you mentioned 2 different aspects. Something changes and you need to update it. Another one is, how easy is it to figure out what's valid to use. Second depends very much on what WGSL looks like. If have explicit offsets / strides, it's easy to see what minBufferSize is. Already have the numbers in the shader. A little strange to say it's hard to figure out what those numbers are.
- DM: not like you can add a uniform and say your code will just work. You'll change the size of the buffer you're allocating, etc. You know what your buffer size has to be. You're creating the bindings at some point.
- CW: think of sokol_gfx when floooh did the WebGPU port. Has a 10 MB uniform buffer and sub-allocates out of it. Always makes sure there are 16K bytes free at the end. That's the max allowable size. Have a window which shifts.
- DM: that did come up. They always create the binding size of 64K. They always put 64K as the min buffer binding size. Solved. Not a concern for them at all.
- CW: I agree.
- DM: doesn't need to be resolved. Wanted to point the group to this inconsistency problem.
- CW: we could ask Babylon.js folks if it'd be painful to specify minBufferSize everywhere. We can imagine the structure of our clients' code, but not sure. Should ask.
- DM: sounds good.

Add timer query on GPUCommandBuffer #870

- KN: very simple. Adds option to GpuCommandBufferDescriptor which you optionally pass to GpuCommandEncoder.finish(). Get Promise back, resolves to the amount of time the command buffer took to execute.
- CW: any objections? We agreed to this in previous meetings.
- RC: does this mean we'll have to time every GPU command buffer?
- KN: no, there's an option in the creation.
- RC: ah, sorry, I see.
- CW: sounds like no concerns. Let's merge right away if OK with editors.
- KN: don't want to merge it yet - want to change how it's specced.
- KN: one thought - I don't know how we implement this, but might need to move this to CommandEncoderDescriptor, to know we have to start a timer query at the beginning. Depending on how this is implemented in say Vulkan, and wgpu where they're recording live.
- DM: good point. We will need this on the encoder.
- CW: OK.
- RC had a question later about whether command buffers were one-shot. Since they are, there's no issue.

Add aspect back to GPUTextureCopyView #873

- DM: think this is required because we can't copy both aspects on D3D12. Vulkan as well. Can do two separate copies but not like we're copying d32s8 with two different copies. (?)
- KN: might need this in some cases to be able to specify
- CW: depth or stencil for texture to buffer copies, so we can copy stencil only, or depth only.
- KN: slightly more complex than that. Instead of putting in TextureCopyView, could go in: GPU texture data layout (technically), or somewhere in the list of args for the methods that use it. Buffer to texture copies. texture to texture copies, and writeBuffer. T2T doesn't use buffer data layout. All have arg lists we can append to. Making arg list longer is unfortunate. Makes enough sense to me to put in the texture copy view. We already have mip level in that view. There are 3 axes of sub resources: array layers, mip levels and aspects. From that standpoint makes sense to put it here.
- DM: makes sense, sounds good.
- CW: do we allow copies between two color textures with aspect:'none'?
- KN: there is no aspect:'none'.
- DM: the aspect's an enum. has 'all', but not 'none'.
- KN: briefly considered adding new one, 'color'/'depth'/'stencil', or similar. However the current 'all'/'depth-only'/'stencil-only' can allow copying of multiple aspects at the same

time. Should we allow that? Could add it later. Similarly with multi-level copies, which we don't support now, but could add later.

- JG: need something there to choose between depth and stencil. Would like a default for the non-depth-stencil color case.
- KN: yes. Would be nice to have default for depth-only or stencil-only textures. Could say "all" means "all of the aspects" except for depth/stencil textures, in which case you can spec one or the other. Going to/from buffer, have to spec depth or texture. Don't think it matters to copy both depth+stencil aspects at the same time right now.
- JG: other option, treat it like subresources. Default can be the first subresource. If you want to say stencil subresource, can do that.
- KN: would be OK if for depth/stencil you have to say depth or stencil, and it only defaults for textures.
- JG: think so, but don't want someone to have to check that the format's depth/stencil and then clones two commands to copy both planes. If not too difficult for us to do both, would be the most compatible thing. If you try to copy a texture, it just works. If you want to copy into buffer, have to do some narrowing.
- KN: say you have depth24plus-stencil8, implemented by depth24unorm-stencil8. May want T2T copy of both aspects to be one memcpy.
- KN: And for a future, concrete depth24unorm-stencil8 format, it could be possible to copy that to a buffer. It's already packed. E.g. stencil bits will either be filled in with 0 ('depth-only'), or the stencil value ('all').
- JG: I no longer think this is a simple PR.
- KN: not so complicated without hypothetical formats.
- CW: we'll talk about depth24plus on day 3. Maybe this isn't blocked, but after discussing those we'll have to come back to this one.

Add maxAnisotropy property to sampler #864

- JG: I just left a comment on this. I agree overall, don't have to spec that there's a max size. Might be 16, might be lower than 16. Since this is a quality knob we can choose to ignore, don't think we should validate the maximum. Let them provide the max, it's just a hint.
- CW: so clamp it for them?
- JG: yes, if we clamp to e.g. 4x internally then hide that. Expect we will only do either 1x or max 16x. Validating that it's less than 16 isn't valuable.
- DM: i want to see, you don't need to say that > 16 isn't useful.
- JG: weirder to say that the max is 16, but you specify < 16 sometimes.
- DM: only on some platforms do you get < 16 as the max.
- KN: we're pretty sure there are platforms where the max will be 1.
- CW: some Android devices probably.
- KN: I don't think the add'l validation here buys us anything. Agree with Jeff.
- JG: doesn't hurt, but I fixed so many busy work errors in WebGL implementations because we spec'd it. My impl has different abstractions and gets the number wrong.

- CW: can we add a limit to the GPU limits, and then we clamp the user's value?
- JG: that also takes more time than just treating it as a number.
- KR: Pretty sure there was at least one GPU where if you passed a huge value it crashed the GPU process. Exposing the device's limit could be helpful.
- JG: think we talked about that but figured we didn't have to because it's a quality knob. It's not an MVP requirement, but a nice-to-have. Think we were considering not exposing this by default. If Unity came to us and said they want to know whether anisotropic texture filtering's available, we could add that.
- KN: we could have that without this validation rule.
- JG: but don't do extra validation just because you can.
- DM: do we allow a value of 0 for example?
- JG: we can require >= 1.
- DM: once you have the validation rule you might as well just specify all the validation rules.
- CW: for webgpu.h would be nice if 0 was a valid value, and impl clamped between 1 and 16.
- DM: probably don't want the input to be 0, so we'd have the test anyway.
- JG: if they specify 0, or -Inf, we'll give them the lowest valid value.
- DM: in WASM, 0 wouldn't mean 1, it would mean disable. ?
- CW: No, 1 is the same as disabled. We say: impl is free to clamp between 1 and whatever number they want. Give a float, they clamp to what they like. Would be nice if we could expose max value we clamp to is, but maybe doesn't matter. If we do this, and you pass in 0, you get clamped to 1 and that's the same as disabling the feature.
- DM: what about this not being a validation rule but non-normative text saying most impls clamp between 1 and 16?
- CW: OK.
- JG: will be fun to test. Try 1, try 16. Make sure 16 is "better than or equal to" 1.
- KN: in the spec when we say how the sampling occurs we'll have to say something about the value getting clamped. maxAnisotropy=0 doesn't make sense. Clamp to at least 1, say you get at most this many samples.
- JG: non-normative text is fine for this as a hint. Put in 15 - do we ask for 15, or do we ask for 8?
- CW: or 16.
-
-
-
-
-
-
-
-
-
-

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

Agenda for next meeting