GPU Web 2018-09-17

Chair: Corentin Wallez
Scribe: Ken Russell (thanks Ken!)
Location: Google Meet

Minutes from last meeting

TL;DR

- Lots to talk about at the F2F
- Texture view investigation #79
    - PR accepted, pending discussions on format compatibility and how to expose the constraint that Texture Storage views have a single level.
- Data on how expensive GetAdapters and GetDevice are #67
    - Mozilla and Google's numbers are different.
    - Even in the best case getting adapters can takes 8ms. Don't want applications to stutter because of it.
    - Consensus: make getAdapter return a promise, createDevice is synchronous but the device can be lost right away (or some other type of error)
- Issue #78 - subgroup support
    - Important extension, but the discovery mechanism would probably be different.
    - Discussions around how extensions are exposed.

Tentative agenda

- Agenda for the F2F
- Texture view investigation #79
- Data on how expensive are getAdapter and createDevice are
- "Don't care" mechanics (stretch)
- Subgroup support #78 (stretch)

Attendance

- Apple
    - Dean Jackson
- Google
    - Corentin Wallez
    - Dan Sinclair
    - Kai Ninomiya

    - Ken Russell
- Intel
    - Yunchao He
- Microsoft
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
- Elviss Strazdiņš
- Joshua Groves
- Mehmet Oguz Derin
- Tyler Larson

https://github.com/gpuweb/gpuweb/wiki/WebGPU-F2F-Meeting---27-Sept-2018

Agenda for the F2F

- CW: discussing timeline for MVP
    - As we develop more code around WebGPU, would be best if API didn't change constantly. If we can reach an MVP soon it would be good.
    - Obviously wouldn't be the only MVP.
- CW: shading languages.
    - DM: will probably be another update from Apple on WHLSL.
    - CW: yes, we need to plan for a time slot around it.
- KR: Since we are meeting face-to-face, should we discuss more contentious topics like push constants?
    - CW: yes
- CW: Overall roundup
- CW: not sure if we're going to solve the system integration with the rest of the web this time. Should go over the specifics again.
- CW: test suite
- CW: DM from your gfx-rs work (since you're implementing all of Vulkan) do you think we're missing anything from WebGPU?
- DM: not off the top of my head. You'll run into these things when you prototype things.
- CW: would like to revisit fences. There was strong feedback to the Vulkan working group that numerical fences are good.
- JG: line item for numerical fences due to feedback.
- DM: we have feedback on the Vulkan advisory panel. We just need to parse and condense it.
- CW: seems this F2F is going to be more about the remaining small points of the API, shading language and more organizational things (MVP CTS, etc.). Should also discuss the creation of the working group.

- DM: would like to see if we can collaborate on native bindings. If you had a header file it would be great if you could share it. But all autogenerated for Dawn project?
- CW: yes. What we could do is provide a snapshotted version of it.
- DJ: we would like some time to discuss WHLSL prototype.
- DM: do we want to discuss dynamic buffers (when you bind them you supply the offset)?
    - CW: is that valid in D3D12?
    - DM: should be
    - CW: my understanding is that it's a Vulkan concept. Can bind uniform buffer with dynamic offset. Would be a root constant?
    - DM: so you'd emulate this with a few constants in the table? I see some applications doing that. Whether they can work around this is another question. Maybe ask ISVs for feedback on this?
    - KR: Which applications are doing this?
    - DM: Dota2
    - DM: to be more precise: there is a giant buffer, they update parts of it with persistent mapping, then you issue a draw call that uses this offset. Allows you to avoid changing any of the bindings for the persistent buffer.
    - DM: doesn't seem immediately applicable to us because we don't have persistently mapped buffers.
    - CW: does make sense if there's a staging buffer and they use it as a ring buffer.
    - JG: would be great if we could expose persistent mapping for the web with explicit flush.
    - CW: think it's slightly different. Two parts: 1) this app doing persistently mapped things. 2) has to use dynamic offsets because every frame it's uploading uniforms packed into a new buffer.

Texture View

https://github.com/gpuweb/gpuweb/issues/79

- CW: investigation #79 about texture queues, generously made by Intel. It's very non-controversial. Investigation doesn't talk about texture views for render targets.
    - DM: I left a few notes which aren't addressed. For example what are the compatible formats?
    - CW: think it's orthogonal. Need to do a detailed format investigation at some point. Will be tedious.
- DM: level count issue?
    - CW: on Vulkan and D3D12, if you access texture as storage / uav, you can only access the first mip level. And often you can only specify the first mip level and not the count. For views to be used as storage, idea is to only allow 1 level. Or ignore level count for storage textures. The latter would be my preference.

    - JG / CW: more discussion (didn't capture)
    - DM: didn't see where we said it would be a storage view? My understanding was, if we add this extra piece of information and add "usage mask" of view, then you'd say that your level count has to be 1 for just these specific cases?
    - CW: not sure if that's the only advantage it brings. Seems tiny.
    - DM: it's mainly to help the D3D12 backend. If it doesn't know how the view will be used then will be SRV UAV.
    - CW: will be bind group creation time, not before.
    - DM: because you don't want to create UAV if not necessary? What stops you from creating all views at view creation time?
    - CW: in D3D12 when you want to create a SRV UAV DSP or RTV, you have to put even the CPU allocation in a heap. Either the texture view owns the heap for its type of things, or when you use the view for something then that thing knows what it needs. For render targets, you'd create a 4-element heap and allocate the things inside it. Should be cheap.
    - DM: ok i see, that's one way to do it.
    - CW: have the level count, just ignore the level count for UAV.
    - DM: will it work for UAV? Would argue for firing an error if you try to use it as UAV. Seems strange here.
    - CW: either way would be fine.
    - CW: were these the only two concerns about this PR / proposal?
    - DM: I left this a week ago... need to look at it.
    - CW: we can give people another 3-4 days, if no other comments, this is tentatively accepted and put it in the IDL.
    - DM: that's all I've got, fine with merging as described.
    - CW: YH could you talk with Jiawei and make the proposal as well?
    - YH: make pull request you mean?
    - CW: yes, on sketch IDL.

CW: data on how expensive GetAdapters and GetDevice are #67

- DM; my understanding: if we don't need to query all features and formats, it's essentially free to create a device.
- CW: that's weird because we don't have the same results.
- DM: I was testing this only on the graphics abstraction, no web stuff.
- CW: we were also testing loading the DLL. Would only do that if we were starting WebGPU.
- DM: that's something we wouldn't include – we always load it. The logical device creation is the expensive part. If you aren't forced to do that then there isn't much to spend time on. Curious to know where 40-60 ms comes from.
- CW: loading the DLL and querying adapters.
- DM: so you're not creating the logical device there?

- CW: no, and maybe we should. Not doing device discovery yet – creating the device, assume it's good. We load the DLL, create the factory, and get the adapter. (in these 40-60 ms)
- DM: i'll try to load the library then. Still weird because Vulkan Linux has 3-8 ms and it's also loading the library.
- CW: Linux is faster at loading dynamic libraries. Sorry Rafael. D3D12 has more layers than Vulkan on Linux (and NVIDIA). Shouldn't matter because most of the cost should be in the driver. Safest to say that GetAdapters can be slow and that we don't really know about logical device creation.
- JG: these all seem that they take significant portions of a frame, at the lower bound. General guidance would be to make them Promises.
- DM: only going to be slow on the first call. Not going to enumerate devices for each WebGPU application.
- CW: you might though.
- JG: even in the warm start case, 8 ms is long enough that you miss a frame. So if in ideal case it takes 8 ms, it's slow enough that we should probably just return a Promise.
- DM: no, you won't have to do any work at all the second and subsequent times.
- DM: what's the app that would suffer the most? Say you have 100 ad banners and the first one will load slower.
- JG: consider Facebook, when Messenger uses WebGL. Loading Facebook, partway through we realize we have to load more DLLs. Scrolling down, the page is janking.
- CW: I think it makes a compelling argument to make GetAdapter return a Promise. Now if that is a Promise, what about CreateDevice? DM's argument about Maybe monad (contagious internal nullability) being able to remove Promise on logical device because that's what we're doing with buffers / textures – interesting, but a lot of details. Do you still get an error stream if you do that?
- DM: why not?
- CW: the object's invalid, maybe it gives you an error stream and maybe not? Lot of details to sort out.
- KR: what about lost devices?
- CW: maybe you don't get a Promise but maybe your device is lost right away.
- KR: For WebGL regretted not going the async path. Context can be blacklisted but still returned. For WebGPU setup should make APIs faillible like promises then can have error callbacks for the rest.
- DM: it's the general model of the API
- KR: Trying to port synchronous code, this would break their assumptions.
- JG: is it a common case where somebody's going to create a device and synchronously query whether it's there?
- CW: I don't think there are synchronous queries on devices by design.
- JG: the query's on the device. Adapters can have multiple devices.
- CW: the adapter has the maximum limits.
- JG: not in the sketch API.
- DM: that's the idea maybe, but pushed back to V1.

- KN: it's commented out. We have the structure in there.
- CW: it's commented out because it's post MVP.
- JG: maybe shouldn't be post MVP if we're making decisions on it today.
- CW: OK. Let's say that you get the information about the extensions you have, etc.
- DM: how does it affect our decisions?
- JG: if the first point where the impl tells you the limits are device, people will create device and immediately query the limits.
- CW: we expose limits at the adapter level. App opts into higher limits than defaults.
- CW: only sync APIs are: extensions, limits and adapters. These we know when we create the device. They're not really synchronous, but they're already there when you create the device. Seems that all of WebGPU is asynchronous.
- DM: think we have information we need to blacklist things at the adapter level, not the logical device level.
- CW: going back to sync app use case: is it really bad if at setup they might have to do some asynchronous work? What it looks like on the engine thread is that it gets a device or an error.
- KN: think it's fine. For example if you spin up a pthread and enter the main loop and it's sync from then on, the basic setup stuff can probably be done before pthread main starts at all. Think emscripten can handle this without breaking how native application works.
- KN: there's plenty of other async stuff in setup, like downloading resources. All sorts of web API stuff that has to be done in setup.
- KR: Good point that the underlying APIs are synchronous, we need to solve that at a higher level. We already doing this with partners for WebGL and the investigation will inform discussion in this group.
- KR: will these things be Transferable?
- CW: Devices will be shareable.
- KN: postMessaging it will make a "copy" but reference the original thing under the hood.
- CW: how do people feel about?
    - getAdapter returns a Promise
    - createDevice appears to be instantaneous
    - If createDevice doesn't work, then it's lost from call #1.
- DM: or a different error.
    - CW: correct.
- RC: fine with getAdapter / createDevice being async. Was told that D3D12 device is a singleton per process, so if you create it multiple times you get back pointer to the original. But first time creation might be expensive. As JG says, if you call it and immediately ask for limits, will have to wait for them anyway.
- CW: what about adapter being async but device being sync / instant? Use fancy error mechanism for it?
- RC: you'll have to be sync anyway if you query for limits immediately.
- KN: limits are known already at that point – they came from the user.
- RC: some limits you ask the device for.
- CW: they're known at the adapter level.

- DM: except for D3D12. Advantage for that API. Logical device creation will be free.
- RC: I see. Make both D3D12 and adapter at the beginning.
- CW / DM: yes.
- RC: could do that too.
- JG: then different devices created from same adapter have same limits? When you create Adapter, create Device. Use that for WebGPU. Then if you create second WebGPU device?
- CW: validation code will not compare against the same limit.
- DM: not sure if D3D12 guarantees the same limits. Can make nullable in this case.
- JG: lose the adapter? Limits changed.
- CW: think we're confused.
- DM: maybe RC can get some feedback from MSFT.
- JG: just have to create a device to ask what the limits are.
- CW: agree for Promise for getAdapters, instant for createDevice.
- JG: is it getAdapter or getAdapters?
- CW: you say getAdapter for "low-power", "high-performance", etc. To avoid fingerprinting. Not sure it's worth it.
- RC: if you make D3D device when you get adapter then why are they separate objects in WebGPU?
- CW: in Vulkan can quickly query adapters for what they support.

CW: Issue #78 - subgroup support

- MD: subgroups are useful for compute. Can't comment about graphics pipeline; I mostly experimented with CUDA. There, performance improvements are close to the theoretical.
    - Logical grouping of threads.
- CW: internally there are people asking for subgroup support. Also produced an investigation. The problem is that it's not supported on all APIs. So need a mechanism for exposing optionally. Would be a nice first extension.
- MD: D3D12 lacks shuffle operations, which I find useful in other APIs. OpenGL completely lacks them. Need some optional mechanism to support subgroups.
- CW: in the group, contrary to Vulkan, we don't want to query the device, but the device tells you the supported extensions, and you enable the extension later. Would be an easy extension because it enables new operations in the shading language, nothing else.
- DM: there are also limits that we can modify.
- CW: yes.
- MD: how do we structure the extensions?
- CW: problem is that we don't have a spec right now, so difficult to make a diff against it.
- JG: it's fine. Just include it in prose.
- CW: maybe at F2F we should design what an extension looks like? Mainly prose.
- JG: sure.
- RC: yes, separate document for each extension next to sketch IDL would be good.

- CW: not sure we want to go the object route like WebGL.
- KR: Don't think you want to add properties to the WebGPU objects themselves because it could make them slower.
- JG: it's fixed at device creation time. Create with the extensions you want.
- KR: So you don't want to dynamically enable extensions like WebGL (CW: No) so that might work.
- RC: reason WebGL made you enable them was to avoid people using extensions accidentally. Think it's necessary to both enable extensions and put things on the extension object.
- JG: we went only halfway – e.g. for float textures you can just store off the extension object, but GL_FLOAT was defined on the core context. Here, you have to ask for the extension early. If you don't ask for the extension early we don't let you create float textures.
- CW: probably error / context lost.
- JG: maybe you ask for extensions and you don't get an adapter back because it doesn't support the extensions you asked for.
- CW: maybe most compatible way is to not ask for any extensions, or any higher limits.
- RC: so probably have to ask at the beginning.
- KN: right now, you have to ask for list of extensions when you create the device.
- JG: unfortunately, people will create a device with all of the extensions the adapter exposes.
- DJ: is there any benefit to not enabling all the extensions? Maybe all extensions should always be enabled.
- RC: I don't like that way. People will only test on their machine.
- DJ: think Jeff has a point, people will create the device with all extensions turned on.
- KN: they'll have explicitly done something wrong. Enabling all the extensions requires code.
- CW: also hopefully WebGPU will be useful enough without extensions.
- RC: in real life people will probably use Three.js or Babylon.js. When you start up you don't know what the web developer will ask for (FP textures, etc.). Will be difficult for you as a framework author.
- JG: that's a case where they'd like to enable all extensions up front. Two years from now there will probably be useful extensions?
- KR: should we dynamically enable extensions like WebGL?
- JG: Vulkan requires you to enable them up front.
- CW: let's put this on the agenda for the F2F then, and a sample extension.