GPU Web 2020-10-05

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

Tentative agenda

- Should clear values for integers lose precision, or be emulated on D3D12? #1085 (Kai)
- Capability-querying APIs for GPUAdapter
    - Make GPUAdapter.extensions a setlike interface #1098 (Kai)
    - Add a limit-querying API for GPUAdapter #1100 (Kai)
- Rename "extensions" to "features" #1097 (Kai)
- Method of ensuring GPUShaderModules can contain MTLLibraries #1064 (Myles)
- PR burndown
    - createBindGroup: Require a superset of the layout's bindings #1061 (Corentin)
    - Add filtered texture and sampler binding types #1076 (Dzmitry)
    - GPUColor: remove sequence overload from the union #1079 (Kai)
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
    - Myles C. Maxfield
- Google
    - Austin Eng
    - Brandon Jones
    - Corentin Wallez
    - Dan Sinclair
    - David Neto
    - James Darpinian
    - Kai Ninomiya
    - Ken Russell
- Kings Distributed Systems
    - Daniel Desjardins
    - Dominic Cerisano
    - Hamada Gasmallah
- Microsoft
    - Damyan Pepper
    - Rafael Cintron

- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
- Michael Shannon
- Mehmet Oguz Derin

Administrivia

- Please fill in the VF2F Doodle

Should clear values for integers lose precision, or be emulated on D3D12? #1085 (Kai)

- KN: Propose that we allow only clear values up to an absolute value of 2^24 for integer-typed textures. This is the max safe integer that can be represented by a 32-bit float. D3D represents the clear value of a 32-bit float.
- MM: So how do you clear white?
- KN: By drawing a full-screen rectangle. This is only for int textures.
- MM: if someone writes a native app & they wanted to clear to white, how would they do it in D3D?
- DM: we're talking about integer textures. It's not "white", but INT_MAX, etc.
- KN: You would clear the resource separately instead of using a clear command
- MM: what's "something else"?
- KN: Either a copy or render a fullscreen quad.
- JG: surprising that this would be an issue in the D3D spec. E.g., what does native app do to clear to 0xFFFFFFFF? Would expect that to show up in WHQL testing.
- KN: can we agree first to make this the semantic now? This semantic is forward-compatible.
- JG: want to figure this out first.
- MM: this sounds like picking a bad solution first.
- JG: shocking from an API design decision. If this is D3D's semantic, that's OK, and we can choose a different direction.
- RC: As far as clearing, you can use ClearUnorderedAccessViewUint, but you have to create the texture resource differently with particular flags that make it a UAV. Or you can write a shader to clear it.
- MM: So, do we know what the cost is of marking them as UAVs, or the cost of running a small compute shader to clear it?
- RC: I've asked multiple times - didn't get a clear answer. Depends on driver, hardware, etc.
- CW: you'd lose framebuffer compression if you do that - actually every solution would.
- KN: is there framebuffer compression for integer textures?
- CW: Not sure. At least if you do the clearing with a quad you stay on the tiler.

- KN: good point - might want to clear and then not write the result at the end of the render pass. Still want the clear to work.
- CW: to understand better - do we need to make a constraint in the API, or can we do it for the user?
- KN: possible. We can insert a full-screen quad, or a UAV usage.
- MM: probably compute shader vs. full-screen quad.
- KN: would be full-screen quad. Would be inside the render pass.
- JG: this is when you load a resource as a render target & want to clear instead of loading it.
- MM: So you'd have to swap out the shader to draw a big quad, and put the shader back.
- KN: we'd do it at the beginning of the render pass.
- JG: would be fairly efficient.
- MM: didn't Rafael tell us last week that draw calls are overlapped & run at the same time? Maybe no perf cost.
- RC: dispatches with no barriers are all over hte place. Draws - only thing guaranteed is writing of pixels at the end, so you can do predictable blending. UAV writes can be reordered by driver / HW at will.
- MM: would be good if we didn't have a 2^24 constant embedded in the spec.
- RC: I can ask for an official position on adding those flags, using UAV etc. would have perf cost. Do we have flags in WebGPU about whether you'll use a resource in a compute shader?
- DP: you do not want to set the UAV flags on things you'll be using as render targets. It will disable framebuffer compression.
- MM: OK, that's one of the two options. Even better would be data and a benchmark.
- DM: is it a lot of work for this issue? Nobody will notice this, and won't block MVP.
- JG: it's surprising. Peoples' knowledge won't change much between now and MVP. We know when encoding the render pass whether D3D supports the clear value, and fix things up on D3D12 only.
- CW: in a sense it doesn't matter much either way. Just have to make any decision here. Decision could be, force all impls to do full-screen quad to do the clear.
- MM: it does - the other 2 APIs do support these values.
- DP: I think it's fine for the WebGPU impls to be different in this respect. You're targeting different hardware; can't expect the perf characteristics to be the same on all hardware.
- JG: if we tell users you can't rely on this, would they do something significantly different from our impl? Weird foible in our API where we'd make everyone write the full-screen quad code.
- MM: if we forced web devs to inject their own full-screen quad, on Metal we'd probably try to detect them doing a full-screen quad and change it to a clear color instead. Awful.
- DM: there's a counter-argument; we aren't going to make pipelines for all the formats we need to clear this way. Don't' want to delay the pipeline creation until needed because it'll hitch.
- RC: if we're concerned about just this 0xFFFFFFFF value, could clear to floating-point -1. Concern then is only 2^24 to 0xFFFFFFF.

- JG: is that true? Can we just backfill these?
- MM: ulp of floats is > 1.0 for big floats.
- JG: Rafael may be saying that -1 turns into UINT32_MAX for integer textures. If that's true, we can detect you're using a large problematic number, and instead use the value of the int32 bit pattern that we want for the uint32 bit pattern.
- MM: does that work for -2, -3, -4?
- RC: no, just this one. UINT32_MAX it'll work, but for things between 2^24 and 2^32 hae no control.
- KN: If we can get UINT32_MAX working easily, then it's tempting to say you get up to 2^24 and UINT32_MAX specifically. WAs thinking more about what DM said. It's more complicated. We can't make a pipeline for each format. We need one pipeline for each format, for each render target slot the format could be in.
- JG: doesn't the pipeline know what the format is?
- KN: it knows the format and what attachment slot it's in.
- DM: could be multiple as well.
- KN: can do that as 2 different draw calls.
- DM: ok.
- KN: so this combinatorial explosion is bad.
- MM: this would happen once and we could reuse it forever. There would be a hitch the first time but only the first time for the lifetime of the app. Not per-shader module, per pixel format.
- KN: correct.
- MS: unlikely to change through the lifetime.
- CW: If we can do 0xFFFFFFFF and a lot of small values, maybe it's enough we can do the quad and support the path that won't be used that much. Still be tested though.
- MM: that's a good point. Can we also submit a feature request to D3D?
- RC: you mean to use clear integer formats without CLEAR_UAV_UINT?
- CW: my point was, implement & test this even though it'll never be used. Or have a corner case in the spec. Whatever choice we make likely won't affect things much.
- RC: Right, I think most people clear to zero. Not sure if people clear to large uint numbers.
- MM: I think that's an argument for having this be inside the API. On the off-chance that someone passes in a large value, performance will be bad.
- MS: clearing to white - I've used that a lot with WebGL before. Clearing to a color not full white or full black - have only done that once personally.
- MM: Don't some applications clear to like.. bright purple so you can see cracks?
- KN: yes but we're not talking about typical colors, but integer formats. Unusual to store colors about this.
- CW: In spirit of progress, let's say that the API has no corner cases for this and we just do a quad on D3D12 and submit a warning. Rare enough that it shouldn't matter.
- MM: I think that's good. It will give us data about how bad the polyfill is.
- DM: I'd rather not do this work now. Can do it after MVP.

- CW: it's a small thing. You don't have to do it now in wgpu. Probably will be tests in the CTS, they'll fail, and we can live with that for some time before shipping.
- DC: does clearing include the alpha channel too? Sometimes I use quads for overlays.
- CW: it includes the alpha channel.
- JG: these are integer formats. RGBA8 uint, would work fine. RGBA32UI might be a problem.
- KN: It affects only 32-bit integer formats. So really a corner case for most people.
    -

Rename "extensions" to "features" #1097 (Kai)

- KN: any complaints?
- CW: context: features are optional pieces of functionality exposed on an adapter. Extensions would be things that add optional features in a separate document/spec. Would be called "Gpu features" in the spec.
- MM: do we have any proposed extensions?
- KN: Not right now, but many we could think of with raytracing
- MM: What's the purpose of having these two as different concepts? Seems like whether they're in the sepc or not is administrative.
- CW: exactly.
- MM: why call them different things?
- KN: Another document adding on top of the WebGPU spec could add more than one feature, or add additional limits to the API.
- CW: right now in the WebGPU spec there are just features. Extensions don't really exist yet. In the future people will want to add things not in the core WebGPU spec and we can call them extensions.
- KN: Can think it of just fixing up the naming. Extensions aren't really extending the API because they're all in the API and in the same document. So "extensions" aren't actually extending.
- MS: extensions would be vendor-specific, impl-specific things?
- CW: or Ray Tracing, assume there's a prototype that's popular / useful. At some point maybe gets merged into the core spec. RT would involve some of this group's time, and be cross-browser effort. For a while it wouldn't be in the main spec.
- MM: from an author's point of view, do they care about this?
- KN: No they don't see it. Extensions don't exist in the API. We're just renaming extensions to features.
- MM: talking about world where both exist.
- JG: I think you can pretend that extensions aren't a thing. You have features which are specified in the core document, and maybe if we're doing experimental work that's not core for a while, someone might call it an "extension" but it's really just an experimental feature.
- KN: yes.

- MM: argument to getContext - those things live in different specs, but they're not extensions.
- JG: I agree. I think we can ignore the word "extension". We're just renaming it to "features" with the understanding that they're optional.
- MM: I don't have an opinion about the naming - just wondering about 2 different kinds of optional things.
- KN: They're not separate things. "Extension" is a document that add features. "Features" are the only API thing.
- JG: there won't be "extensions" anymore. Just documents describing features. "Features" are the only official thing.
- CW: Needs to be a mechanism in the future in which external documents that augment the API. Like for when we have tile shaders.
- MM: philosophy I've got - if there's functionality specific to one backend / pipeline, … ?
- KN: I see it for experimental stuff that hopefully gets into core. Not for single-platform stuff.
- JG: think we can take a moment to merge this PR. Think there's consensus. Change "extensions" to "features" in the current spec document.
- CW: and can figure out what to call external documents referring to features.
- MM: Alright.
- CW: think of it as OpenGL / WebGL naming convention vs. e.g. Vulkan naming conventions.
- JG: will have to discuss it when we come to it.

Capability-querying APIs for GPUAdapter

- Make GPUAdapter.extensions a setlike interface #1098 (Kai)
    - KN: thing we need to discuss - do these address any fingerprinting concerns? These aren't just for the sake of the API, but for these concerns.
    - MM: I commented last night.
    - DC: approach being suggested is based on privacy budget - correct?
    - KN: approach isn't based on it - but with it in mind. Creating API surface to which privacy budgeting could be applied.
    - MM: desirable (from Apple's perspective) for web platform to enumerate as few things as possible about the user's hardware.
    - CW: so interactive queries are better in that sense?
    - MM: yes. In WebGL you can request high-performance vs. low-power. That's an interactive query - thumbs-up. Getting list of adapters - thumbs-down.
    - MM: This is the same as what we have today, except it changes it from a list to a set.
    - KN: doesn't remove the ability to enumerate, but adds the ability to query, which seems good to me. Seems OK from privacy standpoint. Like checking yes/no for every possible extension / feature.
    - DC: will these queries have a cost to the privacy budget?

    - JG: Each query will, yes.
    - DC: The privacy budget itself is a Chrome thing?
    - KN: it's a concept that's being explored across some browsers in the web platform. Not that far along yet.
    - DC: Would this replace the user agent string? with a trust token?
    - CW: There's many proposals to improve/change how privacy works on the web. Privacy budget is a proposal. The Trust token is another one. As MM said, there's an interest in making the discovery of things interactive so that however we decide to go forward with privacy, the UA has more control over what happens and exactly what kind of query/fingerprinting what the application is doing. For Chrome though, as far as I know, we're still in data collection phase for Privacy Budget and not 100% going to happen
    - DC: is there a roadmap link?
    - CW: let's talk about it offline.
    - CW: for this extension - everyone OK with making array into set?
    - MM: array to set is definitely in the right direction. Might come back with additional requests. For now we should merge it.
    - KN: would you prefer it's only queryable but not enumerable?
    - MM: Probably, but don't want to commit to that yet. Need to talk to more people about it.
    - KN: if you want information you probably won't get much more from enumerating them. Taking away functionality might guide people to the best path.
    - CW: Let's just land it has a strict improvement now and discuss additional changes in follow-ups.
- Add a limit-querying API for GPUAdapter #1100 (Kai)
    - KN: Similar, but limits are a bit more complicated, so it makes a custom API for it.
    - JG: this one's more complicated. My understanding: my app would like to have 8K textures. You ask, can I have them? It looks at its privacy budget, yes, you can have 8K textures. Another app might ask for 4K textures. Not as big of a hit as telling you, I have 16K textures. Might give you a bunch of bits of information - much more rare capability.
    - KN: that's exactly the intent.
    - MM: that's good. This API adds 2 new entry points - supportsLimit / getLimit. "How big textures can you give me?" Second one is the one I'd prefer not to have, would prefer just supportsLimit.
    - JG: there are a bunch of use cases that just want to know what the value is.
    - KN: hard to split limits into 2 categories, one where you can request the limit and one where you can't. Some apps might just want to use the largest available texture size.
    - CW: yes. E.g. Photopea, uses WebGL right now, wants to move to WebGPU. Want to use the largest texture size.
    - KN: And it would use that to determine tile size, and fewer tiles the better.

- CW: idea would be, you should use supportsLimit as much as you can - but when you can't, you can count it against your privacy budget by calling getLimit.
- MM: I think there are going to be situations where a device supports bigger limits but we don't expose them to the web because of bucketization. If that's true, it seems like a better API would be - which bucket am I in? rather than tell me a specific number.
- CW: isn't that the same though?
- KN: sounds like it would require the browser to bucket the hardware.
- MM: Have we decided that the browsers can't agree what the buckets are?
- KR: We discussed this a lot in WebGL and decided it's rather infeasible to do.
- JG: I think it's less true for WebGPU, but what I worry about is that different impls will have diff constraints on what they can expose. For example, one impl may always be using two of the bind groups, so the max bind group number is two lower. And the other may only be using 1 or 0. Much harder to agree on those implementation details.
- CW: there's impl differences and differences between set of devices we care about. E.g., we'd care about what bucket Pixel phones get bucketed into. Or the vast majority of Android handsets. Ideally, group could look at aggregated data over all WebGPU-supporting devices. Not really feasible. Worry it'll be difficult to have agreed-upon buckets for a while.
- DJ: Would it be possible to have buckets - I don't mind if a browser wants to expose the hardware details of their platform to everyone, but Apple will not want to do that.
- CW: user agents can choose their own buckets is the point.
- DJ: So as long as it's possible to have buckets, then it's okay.
- CW: point is - even if you have buckets, you need to tell the developer, this is the bucket you got. Need a way to ask for limits, or ask for limits from GpuDevice. Adapter, maybe there's a bucketing system - but you need the limits on GpuDevice, to know what buckets you fell into. It's pretty clear there will be buckets for all user agents. But will be hard to agree on common buckets between all browsers.
- DJ: I'd note that we did get some benefit in iOS with where we basically apply limits to a GPU based on the API that they requested. You could have a more powerful device, but not everything was available to the developer. The reason was for compatibility. I'm just saying that as anecdotal feedback.
- KR: So how about we just allow the UA to do its own bucketing or limiting of more esoteric limits/features and let the developer know exactly what they got.
- DJ: think so. Kind of interesting. Difficult topic. Might be the platform that wants you to control that, and not just the user-agent. Maybe on macOS they'd want to suggest you don't expose all the details of the GPU all the time.
- CW: If we ever get to the point where there's different buckets for different browsers, we could perhaps have a global registry of what buckets there are so

you can test in development mode. That's a nice thing about WebGPU for opting into limits.

- CW: Going back to the PR..
- DM: if we agree browsers will bucket, then does this help in any way?
- JG: less useful.
- CW: helps if a browser decides to bucket more quietly, with more buckets, and wants to limit the privacy budget spend.
- JG: If you have 16 buckets, supportsLimit could let you make three queries for different limits.. same number of bits. Given that buckets are likely, it makes supportsLimit a little less valuable, but we could add later. One thing API-design-wise is potentially having the limits object - the browser can observe when you read out of that dictionary. We could do the same thing for limits.
- KN: before came up with this design, was going to switch it to an interface to make those observable. Thought it would be better to switch to something that exposes the queries.
- MM: If it were a dictionary, what would the length field mean? Pretending it's a dictionary without it really being one would be a mistake.
- KN: it would be an interface, with attributes.
- JG: no length field. If enumerable, could enumerate across this.
- MM: enumeration is almost surely the wrong thing here. Question is, what's the largest texture size you have, or do you support 16K textures?
- JG: Two use cases. One is where you require a maximum size, and the other is where you ask for it and react to it.
- MM: I understand an app that wants two code paths, one for small and one for large textures.
- KR: How about we take Kai's incremental update here. Apps that want big textures can ask supports 16k or 8k.
- KN: I think having an expressive API is the most important thing and that this is a step in that direction.
- JG: It is most expressive, but I think it goes too far in that direction at detriment to UX of using the API. The most simple thing is exposing a limits object, and you can create a different limits object by looking at the properties.
- KN: I could imagine there being a lot more than 16 buckets. If trying to achieve 1-in-1000 anonymity, would have a lot more than that.
- JG: my opinion here - this a "more perfect" solution but its perfection is a step away from UX. Trying to fit under privacy budgets that we aren't sure will even exist.
- KN: I don't think it's very complicated. I think it's a minor UX hit.

Method of ensuring GPUShaderModules can contain MTLLibraries #1064 (Myles)

PR burndown

- createBindGroup: Require a superset of the layout's bindings #1061 (Corentin)
- Add filtered texture and sampler binding types #1076 (Dzmitry)
- GPUColor: remove sequence overload from the union #1079 (Kai)

Agenda for next meeting

- CW: sorry I didn't post anything on shader module discussion. Having trouble using MtlLibTool without it crashing.
- CW: more on limits.
- CW: GPUShaderModule/MTLLibrary
- CW: PR burndown for now.
- CW: can come back to multi-queue later. These seem more core to the API.
- MM: any movement on multithreading? Kai?
    - KN: no, we talked about it last meeting but commented afterward I wouldn't have time to think about it. We have already done the core of the work, making objects sharable between threads. But still need to go into detail about shared state. Dzmitry has some ongoing work on this, requiring COOP / COEP for shared state.
    - JG: some concerns inside Mozilla about Spectre-related issues.
    - KN: aware of this for a while, but haven't spent time on it.
    - CW: if something by homework deadline we can work on it for next week.
- CW: please fill in the VF2F Doodle. 2 working days left to fill it in!
- VF2F Doodle