GPU Web 2020-07-20

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

Tentative agenda

- minBufferBindingSize default value #874 (Myles)
- getMappedRange default arguments #908 #929 #908 (comment) (Corentin)
- GPUDeviceDescriptor.allowNonDefaultCapabilities #684 (comment) (Kai/Dzmitry)
- Figure out the base vertex input limits #693 (comment) (Dzmitry)
- OOM handling for GPUBuffer mapping #872 (Austin)
- PR burndown
    - Allow BufferSource in writeBuffer/writeTexture #832
    - Add a validation rule for storageTextureFormat in BGL #925
    - Don't use EnforceRange types for flags consts on interfaces #928
- Agenda for next meeting

Attendance

- Apple
    - Justin Fan
    - Myles C. Maxfield
- Google
    - Austin Eng
    - Corentin Wallez
    - Idan Raiter
    - James Darpinian
    - Kai Ninomiya
    - Ken Russell
    - Sarah Mashayekhi
- Kings Distributed Systems
    - Daniel Desjardins
    - Dominic Cerisano
    - Hamada Gasmallah
    - Wes Garland
- Microsoft
    - Damyan Pepper
    - Rafael Cintron
- Mozilla

    - Dzmitry Malyshau
    - Jeff Gilbert
- Mehmet Oguz Derin
- Timo de Kort

Administrativia

- Apple WebGPU Specification editorship
    - JF: switching teams at Apple in 1 week!
    - JF: not a web-related role. For now, going to be working out how we'll provide WebGPU editorship from the WebKit team.
    - CW: need to decide whether we need more than 2 editors. Probably 2 per spec will be enough.
    - MM: we'd like to maintain an editor from Apple. Should probably be either Dean or me. Given conversations with Dean, think that Myles should be the one to pick this up.
    - CW: you'd be editor of the two specs? That's a big commitment, but OK.
    - MM: discuss on private email thread?
    - CW: not sure - think chairs appoint editors. Any way is fine.
    - MM: I don't have any preference about how to make this decision.
- Virtual TPAC coming up in September / October
    - This CG doesn't usually go to TPAC. Any concerns with this CG not attending?
    - MM: the details of the TPAC aren't released yet - schedule, plenary day, etc. If there is a plenary day it'd be good for some member of this group who happens to be at TPAC to have a session with an update
    - MM: I plan to participate in TPAC for other reasons, so if this group wants someone to run that, I'm happy to, but if someone else wants to, please go ahead
    - CW: we won't ask for a slot for WebGPU in this case.

minBufferBindingSize default value #874 (Myles)

- *On GitHub:*
    - *KN: 0 would never be a valid minBufferBindingSize*
    - *MM: Presumably the set of resources attached to the shader (via bindgroups) is allowed to be a superset of the resources that the shader actually uses. For the resources which are bound but not used, 0 seems like a reasonable value.*
- MM: mostly stylistic change
- MM: good point came up about how even if only member of shading language bound to buffer is an array, we'll still have 0 be an invalid size
- MM: if not using a buffer, 0's a reasonable default
- MM: for general cleanliness, better to have out-of-band messaging rather than in-band messaging

- CW: not sure how Emscripten will do that
- KN: don't understand comment about resources bound but not used. It's in the BGL, so you'd never be able to create a pipeline using that resource.
- MM: yes, that's right.
- KN: doesn't make sense then to make it possible to include in the BGL if that slot in the BGL can never be filled.
- MM: you're right, wouldn't be a useful feature for someone to put 0 in that field. This PR is 99% stylistic.
- CW: maybe Editors can resolve this in their meeting.

getMappedRange default arguments #908 #929 #908 (comment) (Corentin)

- CW: similar to above. Also somewhat of a PR burndown but there are multiple PRs open.
- CW: when you have a buffer mapped, and call getMappedRange with no arguments, would be useful if it gives you the whole range.
- CW: how do we specify the semantics of the call having no arguments?
- CW: "the offset defaults to 0", and the offset's relative to the range in mapAsync.
- CW: or, out of bounds messaging. If undefined, it defaults to the start of the range that was mapped.
- MM: that second one sounds like it makes more sense. Default arguments turned into undefinedes if they aren't supplied.
- KR: think it's important to handle this case in Emscripten which doesn't have default arguments.
- ...could do with C style mangling and different functions…
- CW: offset comes before size in WebGPU, can't change order of arguments. If you want to map (4..16) and want bytes (8..12), what arguments do you give to getMappedRange? (4, 4) or (8, 4)?
- KR: Should be talking about the subrange you have defined in mapAsync.
- DM: But it's still an offset on the buffer, and all the other offsets deal with absolute offsets from the buffer. It would introduce another frame of reference. no strong reason to do that.
- KR: once you hand back a mapped range shouldn't you speak according to that.
- DM: We still have the same buffer, it's not a new type that's returned.
- MM: I think your premise may not be correct: about avoiding additional frames of reference. We may want additional frames.
- JG: in general I agree, and like the analogy to ArrayBufferViews. However, because this is on the buffer and not on some intermediate mapped buffer thing, it makes sense for it to be buffer-based esp. if we get to the point where we have multiple mappings of the same buffer live. If 2 ranges of buffer mapped, getMappedRange should be able to talk about either of those. Not paint ourselves into corner API wise.

- MM: That's compelling. the only other option would be to put getMappedRange on an intermediate object.
- KR: Okay I thought there was already an intermediate object. I agree it shouldn't be relative to an implicit active mapping.
- CW: Okay, so passing no arguments gives you the whole "active" mapping -- this is also an issue if there is more than one.
- JG: probably not too bad to say that if you want to join these two at the hip, make your own intermediary object. In example, you'd have to do mapAsync(4, 4) and then getMappedRange(4, 4).
- MM: making it seem like an intermediate object is more compelling. Why doesn't Promise resolve with some intermediate object?
- CW: getMappedRange is also used for buffers mapped at creation.
- MM: can mapping at creation also return one of these things?
- CW: we had that before.
- JG: it'd be a different API than we have now.
- MM: possibly better.
- JG: I enjoyed this API more compared to the previous one. As long as you get your mapping right, getMappedRange won't throw or be nullable.
- CW: what about: getMappedRange with no argument defaults to (0, size of buffer). If you use mapped at creation, that's what you want. If you want the simple thing, mapAsync the whole thing, getMappedRange gives you that too. If you want multiple slices, it's fine - you were going to start passing in small ranges already.
- MM: sounds fine to me.
- DM: that's OK.
- CW: resolution: getMappedRange with no args => (0, size of buffer). Offset is relative to the start of the buffer, not of the active mapping.
- JG: to be specific - both arguments are probably optional. If you spec offset, leaving size undefined but have defined offset => default should be not the size of the buffer but size of buffer - size of offset, which is what we have for WebGL.
- KN: I agree but had proposal at end of #908 about not having this behavior at all. Let users supply the size.
- KR: it's been useful in WebGL to only be able to specify the offset and be able to drop the size, have it be default (size of buffer - offset).
- KN: I don't feel too strongly about this.
- MM: Ken had a use case, sounds pretty strongly in favor of Jeff's proposal.
- CW: someone should see whether the spec implements the semantics we want.

GPUDeviceDescriptor.allowNonDefaultCapabilities #684 (comment) (Kai/Dzmitry)

- DM: simple flag that prevents you from passing adapter limits into device limits. When they want custom limits they pass the flag. Simple solution to portability issue of the use case.
- JG: they'll step around it.
- DM: yes but they'll be conscious of it by opting in.
- CW: Our plan was to have an extension "lose device on creation", print to console.
- DM: We still haven't talked about extensions, so your idea is still on the table
- JG: feels like something that's more boilerplate for no extra value.
- KN: then they've copied/pasted the right stuff.
- JG: think if they're not reading the spec and not caring about portability we shouldn't care much.
- MM: can we pull the flag out of the device descriptor object to make it harder to copy/paste?
- DM: don't think people will be negatively affected. People with custom limits should know what they're doing.
- MM: think this is good to solve
- JG: shouldn't have footguns, but don't think this is a big one. It's the way WebGL already works.
- KN: I don't think it's a high cost to prevent a footgun.
- JG: think it's almost an insultingly low bar of boilerplate to get what they want to do.
- MM: that's the point - we're making it harder for them to add it.
- JG: there are whole classes of apps that rely on this behavior. This comes from a good place, but you're trying to chip away at the problem of people writing non-portable code. I think this doesn't really move the needle on the portability issue. Since not quantum step forward for us I think we shouldn't do it.
- MM: I disagree - taking little steps in the right direction is better than no steps.
- JG: there are steps associated with each step.
- CW: can you think of a bigger leap toward avoiding this mistake?
- JG: I don't see the need; not one of the biggest problems with WebGL.
- DM: just because it's around doesn't mean it's perfect.
- JG: no, but it does give you an idea of where the problems are.
- CW: in most browsers there will be a place where you only get the base WebGPU because you're an iframe you don't trust. People will notice their WebGPU fails in those contexts.
- MM: I'm a little scared about authors finding "some configurations of some browsers". Think most authors don't test in enough browsers.
- JG: in WebGL we've found they do.

- MM: I'm worried about the successful highly used apps, but also the non-successful and non-highly-used apps.
- CW: should we rethink this and try to make a bigger leap?
- JG: I don't think so - don't think this is the area to find portability issues.
- KN: think we've spent a lot of time on these kinds of issues - does anyone else agree with Jeff's position here?
- JG: if not, the consensus is just to add it.
- MM: you're allowed to object.
- JG: yes, but I'd like to pick my battles and this isn't one of them. Consensus seems to be to add it.
- CW: yes, consensus is to add it.
- MM: should we re-discuss the mechanism?
- CW: since we're at it we can do that.
- MM: can you talk more about the round-tripping?
- CW: from the GpuAdapter - exposes limits and extensions. Can pass that through.
- MM: so GpuAdapater's DeviceDescriptor doesn't have this field, and so if you want to add it in you have to add in this field when passing it back into the API?
- CW: yes. Maybe what's exposed from GpuAdapter should be DeviceFeatures or DeviceGeometry, and DeviceDescriptor inherits from that.
- KN: there's no DeviceDescriptor on adapter - just extensions and limits. Blocked on the other PR (number???) that's been open a long time.
- DM: that's what we want - limits exposed by an adapter.
- KN: we didn't agree on that unfortunately.
- CW: this isn't small enough follow up for this meeting - let's defer to pull request.
- RC: can developers supply arbitrary limits between mins and max's? Are we going to check against and enforce that?
- CW: yes, provided your adapter supports uniform buffer sizes that are more than the default. If you supply a limit, it will be checked against exactly.
- MM: that makes the most sense. Common case: base device will have lower limits than real devices. Will already have code comparing against arbitrary numbers, not the real limits. So we might as well let those numbers be passed in by dev.
- RC: ok, makes sense.

Figure out the base vertex input limits #693 (comment) (Dzmitry)

- KN: we discussed in editors' meeting. Would propose min limit on vertex buffers. 14 covers Metal support, but wanted extra buffers for impl details. Don't remember what we decided upon in the meeting. Maybe 8?
- KN: impl detail - want to expose limits that won't exceed Metal limits when summed together. Adapter Metal-backed, sum of these two limits won't exceed hardware limits.
- MM: for ppl who know more about what games need - is 14 a reasonable number?
- CW: I think there was one concern from people loading glTF files with each vertex attribute packed separately - probably something like ~20 theoretical max.

- KN: right. If that's the case then the glTF impl has to conform to the restrictions.
- DM: think 14 is too high. 8 sounds good for MVP. Don't think the combined limit will work. In VkPortability, we've agreed on 8, non-combined. Trickiness - there are a lot of things that can end up being a buffer. Arg buffers, aux buffers for passing down unbound array sizes, etc. Combined limit => difficult to specify.
- CW: any concerns with 8 max vertex buffers to start, and figure out more later?
- (no feedback)
- CW: great. Good that we can solve old limits like this.

OOM handling for GPUBuffer mapping #872 (Austin)

- AE: discussed in F2F. Decided: when mappedAtCreation=true, have to allocate staging buffer in some impls. Desire to indicate failure to allocate that if really large, and get OOM.
- AE: agreed: we'll throw exception in Device.createBuffer.
- AE: however - when discussed again, undesirable for 2 reasons:
    - Introduces another error path. Buffer creation can fail in OOM + throw exception, or async with OOM.
    - Makes Emscripten layer complex. If exception thrown, you don't have a buffer - it's undefined. Hard thing to do except by returning something that's null.
- AE: talked about in the issue - unify the two error paths. If staging allocation fails, also asynchronously make a GPU OOM. getMappedRange will fail with Operation error because buffer isn't in mapped state
- AE: makes it simpler to deal with error handling, but don't know what kind of OOM happened.
- MM: sync exception - does WebProcess (distinct from GpuProcess) have to know the state of things?
- AE: this is on buffer creation where there's an allocation failure, where you're allocating staging memory in WebProcess. No round-trip to GpuProcess.
- DM: don't understand Emscripten concern. Resource creation fails, you get a "wrong" ID - seems consistent with error model today.
- AE: diff is: if use same error path, in native it'll be an Error buffer, and in JS we'll have no buffer. Incompatible a little bit.
- RC: can you explain more about dummy buffer in the issue? Something you make from shmem, or something else?
- KN: "dummy" buffer was - if we fail to create buffer (fail to allocate shmem, e.g.) then dummy buffer is purely CPU side allocation which only exists to make mapping work, even though buffer's invalid. Data doesn't go anywhere, just a black hole.
- RC: so would only be made if shmem allocation failed?
- KN: think so.
- AE: one goal - getMappedRange nearly never fails. Also technicalities where create buffer, device is asynchronously lost, you call getMappedRange, still have to give you a result where data goes nowhere.

- RC: is getMappedRange a place where we have to round-trip to the GPU process?
- CW: no, it's handled renderer-process side.
- AE: I think we should make them use the same OOM path because it's simpler.
- KN: definitely useful to collapse as many error paths down as possible. Only concerned about collapsing CPU OOMs with GPU OOMs. Generally fallible allocation failure response would be, free up GPU resources and retry. In this case if it's a CPU side allocation failure that won't necessarily help, and you'll have unnecessarily evicted GPU resources. Should we have an indication to users? Flag on OOM error object so they can choose to use that extra info if they want? Could be a shmem exhaustion instead of regular CPU memory exhaustion. Think there are limits on number of pages sometimes. So freeing up CPU resources might not help.
- WG: we have a similar problem allocating worker threads in browsers where vendors haven't told us what happens when we allocate large numbers. OOM in large cases -> Chromium hits C++ exception and crashes the tab. We could conceivably recover from this. One of the things we have to do is gather metrics - like to know why stuff fails, so we could not retry the same operations on the same machines or similar ones. If you can bubble up info about this to the users it's helpful down the road.
- DC: we ran into this with multiple GPU setups. K-anonymity issues in browsers. At least WebGPU in native, would like to be able to query this sort of information.
- KN: there are a lot more obstacles with giving back memory information. Could probably tell you how many bytes of free memory there are, but it wouldn't be useful - fragmentation, limits on numbers of pages that can be mapped. Ran into this a lot before in WebGL. We can't expose that in a way that's useful.
- WG: even just telling us "there was a RAM allocation problem" is more useful than "there was a problem", which is more useful than a tab crash.
- MM: that makes sense. In short term, look at ServiceWorkers, might be able to figure out more information with that.
- WG: we have, will look more. We're trying to do something a little different, hard because some browsers throw exceptions while others crash the tab.
- CW: being able to differentiate different error modes is imp't. Kai's suggestion sounds good, have one error path but can differentiate between diff scenarios.

PR burndown

- Allow BufferSource in writeBuffer/writeTexture #832
    - KN: we said we'd discuss this on Github but never did. Issue's hard because nobody has strong opinions. From mtg notes, seemed Jeff wanted to discuss this, but now it's been so long I don't know if we have anything to say.
    - KN: maybe should push back to editors' meeting.
    - CW: ok.
- Don't use EnforceRange types for flags consts on interfaces #928
    - CW: defer to editors.

Agenda for next meeting

- DM: Multisampling
- DC: multi-GPU
    - CW: can we have an investigation for multi-GPU?
    - DC: sure
    - MM: more than just having 2 devices exposed?
    - DC: yes, in compute API, want to be able to use all the GPUs available in the system. Esp server-side. Looks like WebGPU is set up for this, but Dawn hasn't implemented it I believe. WebGPU is so browser-focused that we're running into issues finding out that there's more than 1 GPU. Kai's added some suggestions on Github. That's the basis for the investigations.
    - KN: there's a lot to figure out. Right now no adapter enumeration, and only powerPreference.
    - DC: we can provide hinting
- CW: won't have multi-queue - sorry.
- CW: please make more suggestions. Floating / open agenda.