GPU Web 2020-11-16

Chair: Corentin
Scribe: Ken
Location: Google Meet

Tentative agenda

- Make blending state explicit #1134
- Texture dimensions should be required to be >=1 (forbid zero-sized textures) #1200
- Are implementations allowed to crush DOMString labels down to ASCII? #1211
- There are too many combinations of texture bind point type and sampler bind point type #1164
- Swap chain formats #1185 (comment)
- Multi-queue things
    - Multi-queue synchronization #1169
    - Remove GPUFence #1217
- PR burndown
    - Add validation rule for QuerySet and Timestamp Query #1161
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
- Google
    - Austin Eng
    - Brandon Jones
    - Corentin Wallez
    - Dan Sinclair
    - Kai Ninomiya
    - Ken Russell
- Intel
    - Yunchao He
- Kings Distributed Systems
    - Hamada Gasmallah
- Microsoft
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert

- Domenic Cerisano
- Eduardo H.P. Souza
- Michael Shannon
- Mehmet Oguz Derin
- Timo de Kort
- François Daoust

Administrativia

- New W3C patent policy
- CW: should be mostly free for WebGPU to adopt the new policy
- FD: details are in the email. Should be free. Orgs already approved it. Don't see any negative points in switching to the new policy.
- CW: any process? Should we ask all member companies? Or can we approve the switch here in this meeting?
- FD: however you usually make decisions in this group.
- CW: usually come to consensus in this call. Everyone OK with using new W3C patent policy for WebGPU? Only hassle, all companies have to rejoin the working group, since it's technically a new charter.
- JG: have to just try to do it and see if anything goes wrong.
- RC: our lawyers have reviewed the patent policy extensively. We're OK with adopting it.
- DJ: and from Apple.
- CW: seems to be no concerns. Can do what Jeff suggested, do it and see if anything breaks.

Make blending state explicit #1134

- DM: originally thought could be spec'd as though blending is always enabled with these values. Semantically still the case, but have some state conditions under which blending's not valid to use at all. If-rule breaks here. Should be more specific. In native APIs it's also explicit: Metal, Vulkan. Any concerns about changing this?
- CW: to clarify: if you don't set the descriptor, then blending is disabled?
- DM: yes. And if you set it, we verify the descriptor as-if blending is enabled. Require the format to be blendable.
- JG: that sounds best. Fail-fast for people trying to blend something that's not blendable. Biggest sticking point for some of the WebGL blending issues. Not really clear until you try doing things, and then it silently doesn't work. Validating & failing ahead of time is better.
- CW: sounds like good usability change for the API.
- JG: takes us back to before-time? Thought we didn't need it, but now we do?
- DM: I don't recall. Prob doesn't affect users either. If they were specifying it, still can.

- CW: had discussion about whether to explicitly enable blending, or under certain conditions, automatically. This just tags it with the descriptor being present or not - very fluent in JS.
- JG: on validation - need to write down in spec - what if you write one down and not the other? e.g. define color blending but not alpha blending. Does alpha blending behave as-if it doesn't work? Validation error? etc.
- DM: right, needs to be written down.
- CW: sounds like we have consensus to do this. Add'l work to do on editing side.

Texture dimensions should be required to be >=1 (forbid zero-sized textures) #1200

- JG: nothing to discuss here, just need to spec it.
- CW: Myles had a PR that started doing this, but he's out for a while. Should someone take over and write the validation for createTexture?
- JG: I can do it.

Are implementations allowed to crush DOMString labels down to ASCII? #1211

- JG: consensus here is that it doesn't matter. Feels unsatisfactory. Maybe need to write the spec text that says you're not guaranteed to get these back?
- CW: a note on insertMarkerLabel and push/popDebugGroup? The string you put in you might not get back in the exact same form?
- JG: if that sounds good, can move to Needs-Specification. Add non-normative text to the spec. I guess I can do that too.

There are too many combinations of texture bind point type and sampler bind point type #1164

- CW: I forgot to link to it - Brandon has PR #1223 for this.
- BJ: there is a PR in flight. Not done - has tentacles all throughout the spec. Weeding them out. Logic's pretty straightforward. Re-plumbing various paths. Updated syntax, had some question about the verbage, using "type" instead of "kind", etc. I'll keep updating the PR linked from the issue with progress. If you want more input leave me feedback.
- CW: we can keep going on this PR this week. Put it on PR burndown for next week?
- BJ: I'll continue to go forward with the editors. Won't merge until group signs off on final iteration.
- CW: please do look; potential to be massive usability boost.

Swap chain formats #1185 (comment)

- KN: Added this to the agenda late, but I have two quick questions if we can possibly figure them out now (maybe JG/KR can answer).
- KN: 1. Does it make sense to use `-srgb` render target formats for rendering into normal canvases (that don't have CanvasColorEncodingEnum "unorm8-srgb"), and vice versa?
    - JG: depends on where you want to spec it. Disconnect between how we do it in WebGL, and how we're doing it with swap chain formats in WebGPU. If you mark a canvas as unorm8, and the swap chain unorm8-srgb, that's weird.
    - KN: but you could? One defines how it's written, and one how it's read?
    - JG: that's not how i'd implement it. Footgun you wouldn't want to do.
    - KN: in webgl, canvascolorencoding enum defines how it's written. encoding in srgb, store those encoded values.
    - KR: Need to build the compositing path on multiple platforms. We might need to tag (on macOS) the IOSurface with the color space tag, pretty sure it is necessary to have the correct tag to have the correct rendering. Think we need more experimentation. Should be strict for now and see if we extend later. In WebGL the spec says what happens with the canvas creation parameters. If you set alpha=false you get GL_RGB. otherwise GL_RGBA.
    - KN: sounds like we want to figure that out, ideally with WebGL.
    - JG: think in WebGL space we're heading for context creation parameter, you spec one of several sized formats. I want RGBA8, or sRGB A8 one. That's the actual format you use. Think that's what WebGPU should do - pick the buffer and how it's encoded. YOu wouldn't say, I have an RGB16 buffer, interpret that as RGB16F. Wouldn't let you mix-match RGB and sRGB.
    - KN: makes sense.
    - CW: I lack context, but recall JG you said on some HW fast path requires RGBA8 unorm, sRGB.
    - JG: right.
    - CW: rather, BGRA8 unorm, sRGB.
    - JG: don't remember the specifics. When you go fullscreen, or in XR headset, some have prefs that we want to surface.
    - KN: OK. Interesting. Think that's a separate bit than is SRGB or non-sRGB preferred. Some apps might want to only write code path for one or the other - not query the preferred format. Would add complexity.
    - JG: sort of just works. Not that bad if you have a non-sRGB pipeline and render into sRGB buffer. Just encoding. Like if you have RGBA8 pipeline and render into floating-point buffer. NIcer if you had fully FP pipeline, but still get benefits if you do final raster into more appropriate encoding.
    - KN: makes sense.
    - CW: what does it mean for the sRGB render target format?

    - JG: main thing: if you can set up pipeline so its back buffer is in whatever format you give it - should just work. 2 simultaneous questions. Guaranteed formats, and preferred formats.
- KN: 2. Can we simply describe the preferred formats as GPUAdapter having one preferred_sdr_format, one preferred_sdr_srgb_format, and one preferred_hdr_format? AFAIK the only reason the API wants to know the canvas to return the preferred format is to know what its color encoding is. (We'll omit srgb and hdr from the spec for the moment but want to know the answer generally.)
    - KN: q here is: do we want to have a bucket of formats of sRGB / non-sRGB out of which browser gives you the preferred format?
    - JG: think there should be a bucket, there's your choice.
    - KN: so one bucket with all formats.
    - JG: don't think particular reason to split that up.
    - KN: OK.
    - KN: floating-point formats should be different?
    - JG: maybe. If we can pick FP format we can support everywhere, think we should just offer it. Like 16-bit FP render targets; offer them.
    - KN: can browser tell you the preferred format is F16? Or do you have to set the canvascolorencoding enum to F16 to get that?
    - JG: as long as we support all the guaranteed formats, and they all work, and have preferred formats, people will say I'll always render into 16-bit floats. Maybe not as fast as possible everywhere.
    - KN: i mean, should browser be able to choose F16 without the app saying they want F16?
    - CW: if app chooses getPreferredFormat path, they ought to be able to handle the formats that come back. They get texture, they create render target / pipeline with it. Only use the format to create render pipeline.
    - JG: might be safest to just have guaranteed formats. Risk is: apps that blindly choose a guaranteed format, but don't use it if it's FP.
    - BJ: what's the fallout for developers? Testing on bunch of devices only which don't have a FP preferred format?
    - JG: they have switch statement in their app - switch on preferred format, doesn't handle all the cases.
    - BJ: if I as an app accept that, but have no contingencies for FP swap chain. Do I fail at some point? Is there a chance it works and I'm just not using the whole color space?
    - KN: think it will work all the time.
    - BJ: if you're doing any direct texture copies into swapchain texture, might fail. But if doing rendering all the time, might be ok.
    - MS: as long as you're just rendering, shouldn't be an issue. Might not look right.
    - JG: if you're doing multisampling...we made that easier, never mind.

- CW: if you use getPreferredFormat, but then in your code you forgot to update from BGRA8Unorm. Don't propagate preferred format in the right places. Validation error then.
- KN: makes sense - don't think huge risk - except maybe for copying. Don't think it provides value. Browser saying it prefers F16, every app / platform supports RGBA8 as efficiently. Making the API surface a little more unpredictable doesn't give us benefit here.
- CW: won't HDR canvases be F16?
- JG: you won't notice it. either app handles HDR, or it doesn't - then you can render into unorm8 buffer, compositor will be into F16 buffer.
- CW: think we need something to paper over BGRA8 unorm everywhere except Android (RGBA8 unorm).
- KN: proposal: 3 separate buckets. Preferred SDR format, preferred SDR sRGB format, preferred HDR format. Would make things simpler for applications. Only 2 possible values from getPreferredFormat. Don't have to worry about whether copies will go into wrong gamma, etc. Simpler's better for developers.
- BJ: 3 diff buckets - does this mean 3 different attributes? Or 3 different buckets from which you pick based on attribute from canvas?
- KN: the latter. they come back from the adapter, depends on how you set up your canvas.
- JG: interesting idea. One issue: don't want getPreferredFormat to return something people don't understand. What if it were "getPreferredFormat of this list"? You specify the list of formats, we choose one for you. Like, RGBA, sRGBA - I give you sRGBA back. RGBA, RGBA16F - preferred format is sRGB - next best one is RGBA. Of those that the dev supports, here's the one we recommend.
- KN: for sRGB that makes sense. Not sure for F16 - either way canvas has to be already set up this way. Unless we don't use canvascolorencoding enum. Only care about the format you requested for swap chain.
    - https://github.com/WICG/canvas-color-space/blob/master/CanvasColorSpaceProposal.md#the-colorencoding-canvas-creation-attribute
- CW: the list idea is interesting. Might be overkill. Browser just cares about one format vs. the other.
- KN: can add formats later. Problem is, don't want apps to rely on set of things that could come back from getPreferredFormat. Later, add RGBA10_A2, don't want to break anybody.
- CW: don't you verify app passes only valid formats, and at least one?
- JG: if later getPreferredFormat can return F32, apps might break.
- CW: gotcha. App today, start passing F32, would be JS error? In future, browser starts supporting F32, app code is updated, and starts generating errors on old browsers.
- KN: feature detection. Wouldn't have to throw an error if you put it in your list of possible formats.

    - JG: I'll write up the PR for that. Getting back to Kai's earlier point about what canvascolorencoding is for: great point, think we should just not have the canvascolorencoding thing. Imp't thing, the canvas is labeled with its color space. Encoding is whatever. Too much detail for what's underneath. Different for WebGL / WebGPU. From browser's compositing stack - it gets a texture , here's what it samples. Not sure it makes sense to have commonality between the various canvas APIs.
    - KR: Not 100% sure I agree. The <canvas> element is the place where 2D and 3D come together. You can allocate canvas with alpha:false and say it would have different definitions. The defaults might differ. Not ready to divorce WebGPU from other canvas APIs yet. We should have a way to link the canvas definition and the WebGPU swapchain.
    - BJ: random note: in VR headsets, display itself has built-in format it prefers. It might be HDR in the future. Dev won't have that canvas entry point to twiddle those bits. More reliant on hardware to say these are the preferred formats. As an array, format 0 is the most preferred one. Worried about scenario that people always choose format 0 though. Like variant where you give me a list and I pick the most preferred one. IF that's the pattern you land on, would be nice to reflect that into XR too.
    - CW: some XR apps are going to know how to handle any format. They pipe it correctly to render pipelines. Maybe getPreferredFormat can take an optional list.
    - BJ: don't know it's necessary. Some new headset comes out, wants F32 everywhere, nobody's prepared for it. Nice to communicate that, but mostly you'll have apps saying I'm not prepared to handle it, but can handle these very well.
    - KN: I like the list. JG you'll write something up?
    - JG: yes.
    - KN: BJ does have a PR up about swap chain formats (?).
    - JG: PR is nominally about swap chain formats. This can be about guaranteed swap chain formats, and worry about preferred ones later.
    - RC: no strong preference on list/no-list, but +1 to what BJ said, and don't forget about multi-mon scenarios. Some can be SDR or HDR. Or people can plug into monitor that's worse / better than built-in display. We can use the primary monitor for now, but maybe want a callback that the preferred format changed. Preferred one shouldn't be on an adapter, but more on swap chain / headset.
    - KN: right, good point. Didn't think about event. Leave the possibility open.

Multi-queue things

- Multi-queue synchronization #1169
    - JG: I requested changes to have the fence part split off, which was done, so I approve this now.

- CW: lot of discussion about making sync explicit via queue ops, rather than having transfers implicitly synchronize queues (signal / wait ops on the queue). RC you said you're mostly OK?
- RC: I think transferring objects between queues - think we should have. Not sure to also require web developers to do waits/signals to be explicit. API will check all your work. Numbers all correct, etc. First pass - API does signal/wait for you, and you just transfer the objects yourself. We'll run on a large diversity of HW. Some won't have async compute. Some will be so async it won't matter what you run async. Might make your program worse on those computers. Don't think we should be excessively flexible like the native APIs. Think signal/wait part should be implicit.
- CW: 2 main arguments for explicit signal/wait. 1) makes sync be entirely done via queue submit ops. sync aspect is ~2 levels deep, never see the wait, only the signal. never see the end of the synchronization arrow. explicit wait gives the graph explicitly to the developer. 2) if you transfer a resource, doesn't mean you want to sync this point of the queue. might want to synchronize before, or wait later, to control which parts of the work happen in parallel. imp't, in Doom slides, you have specific workloads you want to run in parallel to get the most out of the GPU. If only implicit sync, only part of the arrow, might be "too loose". Might end up doing compute-heavy workloads on both queues, where you wanted to mix one compute-heavy and one fixed-function-heavy workload.
- JG: what if explicit wait and signal were present, but optional? Could people who care use them?
- KN: you mean move wait earlier than what happens automatically?
- RC: if you move the wait earlier, which number do you signal?
- KN: still have to signal.
- RC: if you signal, you have to wait, and vice versa?
- CW: suggestion is: what if we take the PR, but instead of validation error on incorrect wait/signal, the wait's just done? could work. worry is like JS strict mode. believe if you're a dev wanting multi-queue, want thing to explode in your face at dev time if assumptions are wrong. Assumptions are what allow you to reason about perf in your app. One suggestion - there's a strict mode, can enable it in DevTools. That's a possibility. Or, strict mode when you create the device.
- RC: strict mode means having to use signal/wait?
- CW: yes. In JS this turns on a bunch of validation you wouldn't have otherwise.
- RC: strict mode is more about validation than having to do a bunch more things.
- JG: you can always do explicit waits. strict mode would say, i'll tell you fi you do it wrong.
- CW: if you do strict mode you wouldn't need the signals/waits in the first place.
- JG: like how subsequent submits to the same queue are synchronized. submissions to different queues are synchronized based on the resource transfers that are submitted.

    - KN: if everything happens implicitly, what are you actually getting? what have you done that you couldn't do with one queue? maybe can do async compute, but maybe not guaranteeing it. is there really value? do you want automatic sync instead of single queue?
    - JG: you can do async compute. This workload happens on this queue. Have talked about transfer queues in the past, can work there too.
    - CW: for async compute, wouldn't help maximize bottlenecks, but minimize bubbles.
    - KN: still seems pretty good.
    - CW: worried about strict mode you opt into is like us saying, we don't know, so we'll give you both.
    - KN: can you just have a flag, when you transfer, i don't want to have to wait explicitly for this?
    - DM: would like to follow up with either that, or the implicit waits. can always add that flag later.
    - CW: follow up with the app saying " i want easy mode"?
    - DM: yes. I think the PR right now is what we want. Not convinced about easy mode for MQ. MQ is not easy.
    - JG: requiring people to signal/wait is just busy work. Core thing we needed was eager resource transfers on sending queue. Otherwise need to patch in resource calls later. If expect people to enqueue submits - submit, signal, wait - don't know how helpful it is to have that add'l signal/wait, as opposed to making it optional. Have the validation in submit, all these things have to have been waited on.
    - CW: what you lose by doing this is the explicit shape of the execution DAG. Hard to reconstruct it by reading code. Can still get any shape you want. You do submit, has command buffers that contain bind groups that contain resources, implicitly syncs with queue. Then you do something different, and your sync happens much earlier / later. MQ is feature for expert developers, you don't want that happening behind your back.
    - RC: for me, wait/signal seems like the expert thing. Wouldn't say strict mode - opt me into doing it myself. Don't think we should have a halfway point - signal(1000) and browser implements that as signal(3). Either don't have signal/wait, or opt into it, and the signals/waits have to be correct, or you get an error.
    - CW: not sure i got all that, can you put it on the issue?
    - RC: yes.
- Remove GPUFence #1217
    - JG: I was one of the people behind that conversation. After convos with DM, think that whether or not we remove GPUFence, doesn't change core design of MQ proposal, and think core design there is good. Can talk about this too if you want.
    - CW: last week think you wanted to gate the discussion more on this. But can talk more about MQ synchronization as a follow-up rather than preliminary.

PR burndown

- Add validation rule for QuerySet and Timestamp Query #1161

Agenda for next meeting

- BGLEntry rework
- More multiqueue?
- DC: with MQ, have use case where you want to separate out render load from async compute load, and don't want sharing between queues. Multi-device - lot of laptops have 2 GPUs, internal and external. Actually are 2 queues.
    - CW: there are 2 queues, but different adapters. It's not multi-queue.
    - JG: could you file an issue on Github for this?
    - DC: just seems that there are multiple queues, and it's arbitrary that they're on different adapters. Seems you're getting bogged down in the sharing.
    - CW: please file issue on this, can put on agenda.
- CW: for any other topics - please put Needs-Discussion on it.
- JG: request to do PR burndown.
- CW: next week is US Thanksgiving, still OK to have meeting on Monday?
- KN: I'll probably be out.
- JG: we have quorum though.
- CW: OK, we'll meet next Monday.
-
- CW: Joint WebGL / WebGPU meetups