GPU Web 2020-10-21 VF2F Day 2

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

Doc for Day 1

Doc for Day 3

Tentative agenda

- Method of ensuring GPUShaderModules can contain MTLLibraries #1064 (Myles)
- Capability-querying APIs for GPUAdapter
    - Make GPUAdapter.extensions a setlike interface #1098 (Kai)
    - Add a limit-querying API for GPUAdapter #1100 (Kai)
- D3D12 does not support SRC_COLOR used in SrcBlendAlpha slot #65 (Dzmitry?)
- Reconsider the name `OUTPUT_ATTACHMENT` #1153 (Corentin)
- Multiqueue (Dzmitry, Corentin)
    - Multi-Queue Investigation #1065
    - Strawman Multi-Queue Proposal #1066
    - Multi-queue proposal with explicit transfers #1073
- Proposal for importing Web platform images in WebGPU #1154
- PR burndown
    - Add aspect back to GPUTextureCopyView #873 (Austin)
    - 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)
- WGSL
    - Define the interface of an entry point in a WGSL program (#774)
    - Restrictions on function parameters (#1139)
    - shader programming model: permitted memory orders on control barriers (#232)
    - Issue with type converting module scoped variables (#1104)
    - Is Input/Output access one-way? (#1113)
    - Invariant qualifier (#893)
    - Method of ensuring GPUShaderModules can contain MTLLibraries (#1064)
    - Reorder expression sections (#1136)
    - Introduce operator precedence table (#1111)
    - Placement of read_only attribute (#1159)
    - what is the initial value of a workgroup variable? (#1137)

    - Entry point taking in/out as parameters (#1155)
    - Buffer indices should be unsigned (#1135)
    - Remove return requirement (#1156)
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
    - Myles C. Maxfield
- Google
    - Austin Eng
    - Corentin Wallez
    - Dan Sinclair
    - David Neto
    - Kai Ninomiya
    - Ken Russell
    - Ryan Harrisson
- Kings Distributed Systems
    - Dominic Cerisano
    - Hamada Gasmallah
- Microsoft
    - Damyan Pepper
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
- W3C
    - Francois Daoust
- Jerran Schmidt
- Henrik Edstrom
- Matijs Toonen
- Michael Shannon
- Mehmet Oguz Derin
- Timo de Kort

D3D12 does not support SRC_COLOR used in SrcBlendAlpha slot #65

- Can’t use color blending factor on alpha component in D3D12. Worked around so far in the implementation. Detect it’s used, replace it with the alpha component. Think it can be closed.

    - CW: suggestion is, every blend operation is allowed for both color and alpha, and implementations do what they need to?
    - DM: yes. Matches Vulkan and Metal.
- JG: that's great.
- Closing issue.

Reconsider the name `OUTPUT_ATTACHMENT` #1153

- CW: This name is a Vulkan-ism. There are both output and input attachments. Suggestion to rename it to "RENDER_ATTACHMENT" - it's an attachment, but you use it in a renderpass.
- Silent agreement.

Proposal for importing Web platform images in WebGPU #1154

- CW: Lots of interest in using WebGPU to work with video and canvas streams. Presentation from Babylon.js and this is their one big request.
- CW: right now have CopyImageBitmapToTexture. In prototyping, Shaobo found a number of issues.
    - 1. ImageBitmap's supposed to be a free snapshot of e.g. HTMLImageElement decoded view, or canvas back buffer, but that's not always the case. Can be hard to make a free reference to video decoder output. (HW video decoder issues.)
    - 2. No guarantee in ImageBitmap spec of where it lives. Right now ImageBitmap might not be allocated on GPU.
    - 3. ImageBitmap can theoretically be read by multiple consumers at the same time. Difficult to do e.g. if read by DOM and CopyImageBitmapToTexture at the same time can be hard to implement in browsers.
    - 4. Has a copy. Copies for video, fullscreen canvases, etc. aren't cheap. Want 0-copy input for these cases.
- CW: Proposal is two-fold:
    - 1. Go back on our earlier decision to only use ImageBitmap. Like WebGL, allow a sum-type which can import HTMLCanvasElement, Video, ImageBitmap, ..
    - 2. In addition to copyImageBitmapToTexture, we have "importTexture", which gives you a GPU texture. It consumes the ImageBitmap (or maybe not), or from any of these other "things", and gives you a GPU texture that's a reference to that content. Maybe there's a copy internally, but as much as possible, browsers should return you a GPU texture that "wraps" e.g. the internal canvas back buffer.
- CW: Allow sources that are more than ImageBitmap, and more 0-copy or 1-copy paths.
- CW: Some details around letting the developer ask WebGPU, "what's the optimal format / usage for this texture?" - if they want the closest thing to 0-copy, they have to query these.

- CW: The size of the texture itself is known at the DOM level - by HTMLVideoElement / img / canvas / etc.
- CW: Examples of usage in the issue. Goal is to have 0-copy import of various dynamic things.
- RC: The semantic in WebGL when you do texImage2D is also a copy. I understand that this one is not a copy, so what will happen when you import a canvas, and then subsequently write to the canvas. Does that mean the contents of the textures changes from under WebGPU? One nice thing about ImageBitmap is that we can guarantee that it's readonly. Further, if we do allow multiple write usages, what happens if you do WebGPU multithreaded? That'll be a burden for 2D canvas which is single threaded.
- CW: current proposal doesn't explicitly say, but the idea is the thing you get is read-only. If you ask for usage that's writable, a copy would happen.
- RC: what if you get the read one and then you write to it from 2D canvas?
- CW: if you import from canvas, TBD to be spec'd. Maybe when you import, it's torn off from the canvas.
- KR: Canvas doesn't have that mechanism - Offscreen canvas does.
- RC: If we have the tear-off thing, we need a way to put it back. You need to give the frame back to 2D canvas so it can do the next one.
- DJ: Right, a canvas only has one backing texture really that you constantly draw into. In RC's example, you want to keep drawing into 2D canvas and rendering and updating in WebGPU, it has to be a copy.
- KN: I believe 2D canvas is double-buffered or n-buffered. Offscreen canvas when you tear it off, we just make a new one.
- KR: With 2D canvas in particular, contents must be preserved, however.
- KR: In experience from WebGL, it's critically important to have a lock/unlock mechanism at which you release the hold on the texture. So the video decoder can reuse the texture internally. If you fail to unlock, the browser will forcibly unlock it for you, and you probably get a black texture. Adding that mechanism is really important.
- KR: The other thing is the query. This is problematic because in WebGL, we had queries where you ask if it's backed by RGBA, etc. if the API depends on some kind of round trip, it's going to break the pipelining. Think through what the absolute requirements are. If it's somewhat opaque and they can sample from the shader, we should do that.
- KN: probably should have that in a VideoFrame, shouldn't try to do that with WebCodecs.
- CW: Is the format of a video frame only known on the GPU process side for various browsers?
- JG: Only known in the place where it's currently decoded. So.. maybe.
- KR: It's possible the information could be communicated asynchronously so the media player knows. The problem is that the streams can change mid flight and switch formats. WebCodecs intends to cover these cases so at least what you get back is correct - but that's a tricky case.
- AE: So maybe the import defines a map of video format to WebGPU format and it determines it in the future..?

- RC: Something I've felt uncomfortable about, going back to 2d canvas, is this thing where we keep a cache of textures and have heuristics about when to clear the canvas, etc. We can be more explicit - have an explicit tearoff, use it, explicitly reattach it. No n-deep buffering etc. WebGPU usage can set the canvas format(?)
- CW: WebGPU has an advantage where you can call .destroy() which completely destroys the textures. In WebGL even if you delete a texture it could still be bound to a framebuffer or something. .destroy helps in the case of 2D canvas or in the video decoder case.
- CW: Going back to what RC said about tearing off from the canvas - that seems like Offscreen Canvas, so maybe devs that want to render UI and use it in WebGPU should go through offscreen canvas and tear it off with IB. Or maybe import(..) it. But the fast case would be offscreen since they're not presenting it directly.
- KN: One concern with offscreen is that not all the browsers fully implement it. Makes sense to use offscreen canvas and make that the API we optimize for drawing the canvas to WebGPU.
- RC: Are you saying we shouldn't have importTexture from non-offscreen ?
- KN: Probably should, but it doesn't have to be designed in a way it has to be 0-copy/
- RC: Going back to .destroy(), destroying and recreating comes at a cost. The more we can have APIs give each other textures explicitly, it's less overhead.
- CW: .destroy() is loseAccess - it doesn't have to destroy the resource. If we're talking about something that has a lock on a video frame, then .destroy() actually means unlock.
- RC: So if you've imported it and you say destroy(), what is the outside ownership going to be?
- KN: Back to the canvas.
- CW: ^maybe. up to specification.
- DJ: One of the problems is that a Canvas/Video element (not offscreen) don't have a concept of the backing store disappearing.
- KN: That's what I was saying about not necessarily supporting CanvasElement.
- CW: Imagine we add GPUTextureSource so you can do more than ImageBitmap. B.c. of the reasons you've explained, it would always be a copy if it's Canvas. OffscreenCanvas would tear off. Video - this one I'm not sure.
- DJ: I agree with what would happen with normal canvas. Not sure if I agree with OffscreenCanvas. That feels weird that another API is changing the OffscreenCanvas API. Same with video. Some impls might do 0-copy tear off, but some might not. I like the general idea, but I prefer it to be consistent rather than have different behavior based on what you send in.
- KN: I think it would make sense more on OffscreenCanvas because it has transferToImageBitmap. That said, I have a question: what happens when you transferToImageBitmap from 2D canvas? is it copy-on-write?
- KR: No. freshly cleared backbuffer.
- KN: Okay so it doesn't even have to be transferred back.
- KR: No but it could go in a recycle queue.
- KN: If OC already has these semantics, then we should restrict it to offscreen.

- CW: Agreed. What I heard is: having an interface on GPUDevice that has side effects on OffscreenCanvas is unfortunate. So it would make sense to have transferToImageBitmap that is optimized for this?
- KN: Right have the caller transfer it. and then WebGPU consumes it on import. The hard part is figuring out whether to optimize the image bitmap for WebGPU. In theory the IB is holding onto some pointer to the original image - not true in some cases right now, but it could be. Might affect how OffscreenCanvas is allocated though so maybe we need something on OC creation
- KR: Think it's possible to have the 0 or 1 copy path if there's clear lock/unlock.
- KR: If you have HTMLCanvasElement, import to WebGPU, draw stuff, unlock, and then continue in 2D canvas - it should be fine. If you want to avoid side effects on canvas element, any operations in 2D canvas probably need to grab a reader lock, copy it back into 2D canvas' store.
- KR: Probably need semantics to preserved so you don't get - illegal state error or something.
- KR: The thing missing in WebGL is that you don't have the unlock step.
- CW: Ok for Canvas2D you need copy-on-write to happen, unless you're able to somehow lock the canvas, acquire the texture in WebGPU, unlock, and canvas can resume. Is that correct?
- KR: even in that model, to avoid perturbing 2D canvas, you still need copy-on-write so that 2D canvas doesn't have to throw exceptions if WebGPU has it locked.
- JG: Canvas2D - spending a decent amount of time. Seems like we never bothered to solve for WebGL b.c. 1-copy works fine there.
- RC: Babylon.js have expressed complaints about the cost of copying fullscreen 2D canvas using WebGL. They've found this to be pretty poor.
- JG: Drawing the whole UI..?
- RC: Yes, and the semantics of the API make it a fullscreen copy.
- JG: Are they trying to do a fullscreen or would they do subrect if we let them?
- RC: Not sure - would need to ask them. They do have 3D apps that show UI that takes up a substantial amount of the screen. Text especially.
- JG: The other option traditionally is to recommend DOM overlays, but I understand wanting Canvas2D.
- CW: Another q for RC: agreed we need to figure out the use case. Could Babylon.js use OffscreenCanvas and re-render the 2D UI every frame?
- RC: I've discussed with them. One proposal is to do stuff in 2D canvas, transfer to image bitmap, import into WebGL/WebGPU, export that to IB, and give that back to 2D canvas.
- CW: Okay so the semantic is copy-on-write after transferring it. But if you so happen to never use it in 2D canvas while it is locked by WebGPU, then the impl can optimize it to never do copy on write.
- RC: As far as 2D canvas is concerned, it has no backbuffer for some period of time.
- MM: If we go with lock/unlock, and I think we were saying that WebGPU could mark the texture as writable, what about a program where you draw into a backing store with 2D

canvas, lock it from WebGPU, draw into it, and then unlock it, and then draw on top of that in 2D canvas. Do you see WebGPU with 2D canvas on top?

- KN: I don't think we should allow writable for WebGPU
- CW: Either that - or if you ask writable, instantly a copy happens. I don't think the goal is to mix and match writes from canvas/webgpu in the same frame on the same texture.
- MM: If you wanted to use the results of WebGPU in 2D canvas, how would you?
- KN: drawImage with Canvas2D and pass the WebGPU canvas.
- MM: Right two canvas b.c. getContext.
- KR: Couple of things surfaced here:
    - explicitly giving the option to copy if they want so they can do subrect copies so they don't need to copy the whole thing. If lock/unlock might be doing a copy and they can do better with subrects, we should let them do that.
        - CW: Okay so like copyIBtoTexture except let more stuff
    - And videos are very complicated. We ran into tons of issues. There's clip rects and transformations that you need to honor that's coming from the video frame. If we don't want to handle that, we should focus on WebCodecs, or mandate 1-copy.
- CW: I know Chromium has some WebCodec interest. What is the interest in other browsers?
- KR: Lot of interest from video conference customers there. I'm sure they've reached out to FF and others about this.
- KN: Chrome Platform Status says: Chrome/FF positive, Edge nothing, Safari negative, developers positive
- JG: Lot of interest from customers - not sure there's any clear proposal now.
- KR: Sure and it'll evolve based on customer needs. It solves a major problem with video today.
- RC: We've also gotten complaints from Babylon.js, Teams, Flipgrid, etc. people who want to do video effects on regular videos and video from camera, etc. They don't care so much if it's WebCodecs or not.
- RC: Does WebCodecs solve these clipping issues, rotation, color space, tone mapping?
- KR: WebCodecs is indeed being developed by media experts unlike WEBGL_video_texture where we were reverse engineering the video stack.
- JG: FWIW, I think video texture is implementable. I worry here that there are some features we punted on in WebGL because we expected they would happen in OffscreenCanvas, etc. but that never happened. FF still doesn't have OffscreenC. Little worried about tying our solution to other APIs that may or may not be implementable in the amount of time we want them to. vs a more direct solution that could solve user needs and is tractable in this group. Something like video texture
- KR: Subsume what WebCodecs would be providing?
- JG: not sure.
- MM: How does WebCodecs solve.. any of these problems?`
- JG: At some point I want to talk more about what this would look like on the WebGPU side… if the idea is that if when you ask for a video as WebGPU texture - how does that

work? Do we expose an RGBA sampler? or expose exactly what's under the hood to get 0-copy..? but we can talk about that after is stuff.

- KR: Limited experience here. You get Planes from the video stream..
- MM: is that the encoded or decoded data? I thought it's encoded and that doesn't help us.
- CW: Planes you can ask to get as an ArrayBuffer the decoded plane. The assumption here is that for Plane, you would get something that you can import into WebGPU which is the decoded Plane.
- MM: If the form of that is an AB, that won't help us.
- KR: agree..
- FD: I note the current plan to get an "opaque" decoded frame for "GPU backed" frames in WebCodecs: https://github.com/WICG/web-codecs/issues/92
- CW: Clarify: Right now AB is an async thing. For WebGPU, since the data is on the GPU process, there's no ArrayBuffer and you "instantly" import into WebGPU.
- MM: Sounds like we would have to augment the WebCodecs API.
- ?: Correct.
- KN: Yes it would influence the design for sure.
- MM: We have some concerns about WebCodecs, but don't think we've stated a public position on it. Concerns about fingerprinting and design of the API, but I don't think it's fair to say "negative".
- KR: Let me suggest we focus still on copyImageBitmapToTexture. In Chromium, people were complaining about video->IB->webgl paths being slow. Shaobo went and fixed that. Maybe addressing impl details in that code path could get the CG a long way to satisfactory performance. And I think it should be prototyped before it's set in stone. That's how we did it for WebGL.
- <break>
- CW: important topic to solve b.c. customers want it and right now it's a WebGL regressions. Want to have some takeaways..
- DJ: Dumb Q: would be nice if we had OffscreenVideoElement. If you're drawing video to WebGPU/WebGL it's because it's not going to the page. This way ownership can be explicit. Doesn't address everything though.
- JG: It's a formalization of one of the impls we talked about. You might make a video DOM element but it might not be in the tree. I expect the first time you try to copy it into WebGL, if the video was not already being decoded in a place where it's compatible with WebGL, we would migrate it to the WebGL context, and then if you then inject it into the page later we have to do some work to make it usable in two places. Some heuristic - you have a video but video might be decoded in a special way. Also don't know what people do with videos in workers right now.
- KN: Yea that's one major issue with video right now - it's an el and you can't use it in a worker. Even with various media streams, there's still a video element somewhere. WebCodecs - probably, presumably doesn't require this. I don't want to use it if not appropriate, but it sounds like it would really solve problems for us.
- Action items:

- Try engage WebCodecs to make sure the spec works well for WebGPU if we go that way
    - MM: Tricky b.c. WebCodecs is fairly low level and WebGPU is more high level. WebGPU users will just want to get the frame.
    - KN: Fundamentally not possible to do 0-copy from a video into a texture that's accessed from a shader unless we have magic samplers. It can probably be done but difficult.
    - JG: I do think it's our cost and not the user.
    - KN: I think rewriting shaders portably and reliably is worse than users binding YUV frames and matrices.
    - JG: Problem space is complicated. FF has at least 4 different pixel formats and videos can be presented in 1 2 of 3 planes, etc..
    - MM: Let's say users had to do shaders themselves and had access to the matrices and crop rects, etc.. the platform decoding APIs reserve the right to add more transformations in the future. Adding to the web platform would be a mistake
    - RC: I agree with that, and I agree we shouldn't rewrite shaders. Part of the 0 copy needs to perform the tone-mapping and color space transforms, etc.
    - CW: That's the 1 copy case, right?
    - RC: Yes.
    - KN: Yes we definitely should support that.
      My argument is that the 0-copy case could be more complicated because it's more niche. 1-copy is much much simpler for everyone.
    - CW: Okay we all agree on 1-copy where you get video element to RGBA or something.
    - MM: And it should take a bounding rect.
    - KN: CW's copyFromTextureSource.
    - CW: Okay great. but people still want 0-copy.
    - KR: Let's engage the WebCodecs team and work with them to implement that on some platform, and get it working and see what it looks like. Make sure it supports GPU and have a better understanding of what it would look like. Not clear if the latch/unlatch is needed for other texture sources, or just video.
    - KR: At the same time, let's get the 1-copy path working.
    - MM: prototyping is good. IIUC FF has no public signal on this and Webkit also doesn't, it would be a mistake to require the API.
    - KR: I'm only suggesting we cooperate and collaborate on this.
    - JG: FF finds this worth prototyping but not worth tying other APIs to. very early stage.
    - FD: WebCodecs is in the scope of Media working group. Already has a home in their charter - provided it passes the incubation period. We will

be having a session on Monday 10/26 to bring people together to discuss this. It's a highly complex topic.

- FD: TPAC zero-copy breakout session: https://www.w3.org/2020/10/TPAC/breakout-schedule.html#zerocopy
- CW: Everyone wants good 1-copy copyFromTextureSource
- CW: Chromium is volunteering to work with WebCodecs to see if a good 0-copy path is there. Doesn't mean it will be the way we go.
- CW: Open possibilities on other 0-copy paths for canvas / other sources.
- JG: Post MVP thing?
- KN: Should design to support it but shouldn't expect impls to actually be 0-copy
- MS: As long as 1-copy can be done all GPU side without trip to CPU, it's at least usable that way.
- JG: One of the measures I have in mind is that no case where WebGPU is worse than WebGL.
- MM: Even if we did use WebCodecs, wouldn't they need to do a conversion inside the API?
- JG: Depends on how our solution works on our end.
- CW: Not sure I understood. If WebCodecs gives Y texture and UV texture.. that's 0-copy.
- MM: Isn't that the same as asking the author to rewrite shaders to do YUV conversion?
- JG: That is one of the solutions in the solution space. A different solution would be a new type of sampler that has everything built in. And you sample like it's a normal texture and you get RGBA out.
- MM: Right. that makes sense. That doesn't require WebCodecs, right? Also if we wanted YUV to RGB ourselves, that also wouldn't require WebCodecs. Failing to see why WebCodecs help.
- KR: Provides 0-copy path that's not possible in other APIs.
- CW: WebCodec path is getting the planes separately. The other path is like what WebGL does and you have a special opaque sampler.
- CW: From our WebGL experience, the fat sampler approach is a lot to get right. There have been efforts to do this for a long time and still not completed.
- MM: Both of those could work with WebCodecs, but WebCodecs is not required.
- CW: WebCodecs needed for Y-UV
- KN: WebCodecs also provides a VideoFrame object as well which does the RGBA conversion too.
- JG: Disagree that you need WebCodecs for YUV
- MM: +1
- JG: Worried about the time to implement. Fat sampler feels like it could happen in less than a month. If it's attached to WebCodecs, it's blocked on WebCodecs.

Capability-querying APIs for GPUAdapter

- ~~Make GPUAdapter.extensions a setlike interface #1098 (Kai)~~

- Add a limit-querying API for GPUAdapter #1100 (Kai)
- KN: discussed #1098 before. Strict improvement. Landed. Still some desire to remove the possibility to enumerate the feature list.
- CW: the limits discussion was more open than that topic.
- KN: #1100 adds an entry point “supportsLimit”, gives you yes/no answer. Also adds getLimit. Useful for different use cases.
    - maxTextureSize: app can respond to any value. Reduce the number of tiles needed to render, etc. Useful to get the limit value.
    - Other cases where getting the limit is less useful. maxBindGroups; either they require 8, or have a code path that requires 4 or 8 and they choose between them; etc. Wouldn’t expect them to use 19 if they got 19.
- KN: Would it make sense to classify limits between these two? Limits where you can do supports and/or gets?
    - Revealing less information about user machine than necessary.
    - MM: realistically probably 2-3 real values for the max texture size limit, specifically. Can you come up with a different example?
    - KN: don’t have all the limits written down yet. But: resource sizes (buffer sizes), texture sizes.
    - DM: there is a buffer size limit in Metal.
    - MS: number of bindings.
    - KN: not sure about app responding to that.
    - JG: number of samplers. I have a list of the WebGL ones.
    - KN: currently: maxBindGroups, maxUniformBufferBindingSize, probably other max binding sizes, etc..
    - CW: think we don’t need to go through all the limits.
    - KN: point is, some of these clearly can be responded to dynamically and some can’t be.
    - MM: which one do you think would be responded to dynamically?
    - KN: Resources sizes and resource binding sizes, probably. We can always decided on a small set and add as developers find need for them. Or devs will do multiple supportsLimit requests to extract the value.
    - DJ: Still have a feeling that I’d like to see information on what workloads would benefit on being able to use higher values. For that I feel we need to MVP to exist. understand there’s a need for some workloads to query and use more samplers. But I don’t know why we need to add it right now.
    - KN: Many people have already run into and asked about the maximum storage buffers per shader stage. Difficult for them to write their app with as few as we require. They wouldn’t dynamically respond. They might have a fallback path.
    - DJ: So if they don’t write the fallback path, it won’t work on a lot of hardware. I think that’s pretty bad.
    - CW: It’s their choice, and the API is opting into having higher limits. If you want something that runs on everything you don’t opt into higher limits.

- KN: We shouldn't force every app to run on every piece of hardware, that just means people won't use WebGPU.
- MS: Typical for every desktop card to support 16 buffers, and we're limited to 4 for some phones. There are quite a few more complex 3D apps where you don't need to support running on a phone but you need more than 4.
- MM: that's a strong use case for asking "do you support 16" or the higher rather than the lower threshold. If a lot of cards would return 4, 5, 6, … and a lot of apps that would do diff things if they returned those values, that would be a pretty strong use case. I don't see a strong use case right now.
- KN: I'm proposing right now that we limit getLimit to only a few of the limits.
- MM: That is good. I'm a little unclear if a few means 0 or not.
- KN: I think maxTextureSize is needed.
- KR: I think maxResourceSize is needed too.
- JG: I think we should expose all the limits. If we use it and rely on it in our impls, we should pass that along. Should be possible to stack these things. Not envision ourselves as the top driver in the stack. Three.js and other engines will want to query and dynamically choose how many binding groups i have - dictates how many lights, etc.
- DJ: good argument, but just because we can use it internally - we're not the user. Part of the argument is exposing identifying information to the user.
- KN: We should clarify that the browser can respond with any answer it wants. It doesn't not have to expose the actual number and fingerprinting information.
- JG: ..not working in Safari would be a big reason for…. something
- CW: which limits to expose and granularity is a UA decision. Either return a lower value, or trust the app. Maybe drive-by ads would get different answers. Installed PWA might be trusted and have access to finer grained limits.
- KR: The APIs should be there to make the queries. Querying graphics features has been there for decades.
- MM: We're trying to limit the amount of entropy exposed.
- CW: Property of the UA.
- KN: internally the UA might come up with set of buckets, each one would have a set of limits, and return with that set. This API is not prescribing a behavior to the browser at all. Improvement of this API is that it allows the app to request only the information it wants. That way the browser can more dynamically make decisions about what it returns. Still puts all the control in the hands of the UA about what it returns.
- DJ: True. Jeff's ecosystem point. The API can say something and the UA can say what to do, but the ecosystem puts pressure on what the browser does. The decision we make in the API does have implications for more than that.
- KN: I disagree because lower limits are always going to be present on some devices. The browser will always have to run a lower value for the bottom 25% of devices. This doesn't create more pressure on UAs to expose more information.

- DJ: it sort of does if the app only works on the dominant browser, because they're exposing higher limits. Puts pressure on other browsers to lift the limits.
- KN: it means it works on Chrome *on high end devices*. Apps shouldn't run on only some devices unless they need to.
- DJ: Example we got is that most desktop devices support 16 and mobile is 4, so it's not just super high end.
- KN: sure. Also think it's OK to expose to a web page whether it's running on desktop or mobile. We already have high / low precision limits; that's not going to change.
- KR: Can we put the API in place and have continued discussion about bucketing to allow high end apps to work while still maintaining portability?
- DJ: Don't think there's a difference between giving everything and one-by-one. The vast usage of these APIs will be trackers. That's the case for ALL Web APIs.
- KR: All Three.js demos are using these limits to work properly, and make their content portable between devices.
- DJ: I think there's 2 orders of magnitude more trackers that are using this for fingerprinting.
- JG: Insist on data here.
- MM: I appreciate more entropy on some limits and less on others. I wasn't particularly satisfied with the discussion about which go in which bucket. I think the answer is 0 but I'm willing to admit it may not be. How about a proposal where we defer this until after MVP so we get more data about what authors are asking for, and then we can move limits into the additional entropy bucket as necessary.
- JG: I disagree.
- CW: there's already an ecosystem of web apps that interact with limits. Babylon.js I assume takes advantage of higher limits in WebgL today.
- MM: clarifying: not saying apps will only be able to use lowest limits. Set of limits where we return more entropy ("7") is the empty set. Apps that ask "are you high or lower" we give 1 bit of information.
- CW: what's the advantage? Anyone can do binary search
- MM: saying there's no integer as argument to function. It's "do you support the high or the low limit"?
- JG: this'd be a new proposal.
- MM: yes.
- JG: i don't think that's different in practice than exposing the limits object & watching what people fetch from it. Can tell when people check what the max texture size is, and you know what bucket it is and what they're in.
- <break>
- Discussion about what data would make it compelling to change the limits - i.e., the claim that WebGL's used for fingerprinting 2 orders of magnitude more than for actual rendering

- JG: think it would add information to the discussion but don't want to see it be a hard-line approach.
- CW: Because WebGL exposes limits that are used for fingerprinting - UAs have mitigated it by giving low limits unless the app is really using it. But there's also value in exposing the system limits for installed PWAs. The user trusts the app. We should not prevent fingerprinting by construction in the API, but should definitely let user agents prevent fingerprinting.
- MM: agree there's a tradeoff between fingerprinting and fidelity. What would have to happen - what would be compelling for your side to fall on one end of this spectrum vs. the other? Being more concerned about fingerprinting than fidelity?
- KN: regardless of where we fall on this, we're saying it shouldn't affect the design of the API.
- MM: Right if everyone were on the same part of that spectrum, such an API will be silly.
- JG: do we enforce that nobody will ever be able to fingerprint in the API? Or we don't do it politically? Do we bake it into the API or do we not do it? But jumping to try to shape the API to forbid it rather than deciding whether we should allow it is premature.
- KN: There will always be specific use cases where you fall on one end or the other. Ex. if it's literally an ad. Or if it's a Node.js application. But they need an API surface.
- MM: node can do whatever they want. they can add their own API surface. They're not a web browser.
- MS: as user of graphics APIs, I can say that whenever you limit the ability to query it reduces the things we can do. We have one app where we have to send a different shader through based on whether it's an NVIDIA or AMD card because we have to work around bugs. Any time we lose the ability to know underlying details we might have to just send the workaround shader for all the cards.
- DJ: To be clear, we're not arguing against the fact that getting the information is useful to graphics applications. It's about the tradeoff between exposing something on the web that is not used for graphics apps. If we do expose an API, we should just expose all of them - I know MM disagrees. I don't want people to do 50 calls.
- DM: internally in Mozilla our standards folks are not sure about the privacy budget, and were concerned that we shouldn't build support into the API until we're sure about it. So querying all the limits at once is something they like more.
- JG: One thing to point out: today you get the limits object. One thing you can do - and we have to do in some parts of the API, is tell when you access particular members. That's pretty useful for collecting data. We can add restrictions later w.r.t. privacy budget. And we can also add doYouSupportTexture512 after the fact so privacy-minded apps don't need to touch the specific texture limit.

- KN: If the api I've proposed doesn't satisfy Apple's concerns then I agree we should do that. We can do what getLimit does with the current API. The only question is supportsLimit() but we could add that later.
- CW: Concerned that we're trying to make a decision in this group when we're GPU experts and not fingerprinting or privacy exports. We know about it, but really the privacy teams should figure something out, not us. Not this group's place to force a specific design on everyone. However, it is this group's job to make sure the user agent can do what it wants.
- MM: could this community agree on a shared set of buckets?
- CW: had a brief discussion about this internally; agree this group won't be able to agree on a set of buckets. The reason is that people in this group are both making browsers and devices. We have an incentive to make sure our devices are in the "best" bucket available. If the bucket can be made slightly worse so our device can fit into it, that would be amazing. Not saying there are evil intents, but the incentive scheme is wrong, and it'll be hard to have these discussions and get consensus.
- MM: So you said at least until we ship the first version. Do you think this would change after?
- CW: I don't have visibility into what happens after that. I'm sure we cannot get to it before the first version though.
- MM: Ok.
- JG: Had a related issue in WebGL where we had some hardware that could only support 7900 sized textures but not full 8k. It was hard to reach consensus in the group whether 7900 should be reduced to 4k or exposed directly. That was a problem we didn't know when we made the API, and only later when working around driver bugs.
- MM: Sounds like your'e agreeing with CW's reasoning. Are you also agreeing with his conclusion?
- JG: what was the conclusion?
- CW: Before v1, this group will be unable to agree on profiles other than the base profile.
- JG: 2 things wrapped into 1 there. Not sure how to go on it. Would be surprised if we didn't have cases where we discovered hardware in the field that was almost but not quite in a bucket, and we decided to reduce the bucket to give good fingerprinting resistance.
- MM: D3D has buckets. How have they dealt with the problem?
- KN: D3D directly impacts the design of hardware.
- MM: I think that's a good answer. There is a set of buckets.
- KN: Only on D3D devices.
- JG: 2 things when we talk about buckets. Do we have 4 or 5 different limits? Or does each limit have 2...4 options? D3D has the latter. Each limit has 2-4 tiers.
- CW: And the same hardware on Vulkan has even more limits.

    - JG: think we can slightly restrict ourselves to buy back privacy. Don't think we'd want to only have one configuration.
    - MS: reality is that texture size is probably the easiest. 4, 8, 16, maybe 32K. All the other limits become more of a gray area where the buckets should be. Textures traditionally are powers of 2. Doesn't apply to the majority of the limits.
    - MM: Underlying question here: if the group can agree on a shared set of buckets - we can expose it. If we cannot, then there is nothing that any vendor in this group can do that can affect any other vendor. A vendor will pick buckets and ship with it. Either a shared set or browsers do their own buckets.
    - CW: I'd say that's correct. A bit more nuanced b.c. the number of buckets depends on types of pages or content. If you're an untrusted iframe vs installed PWA. Overall very much agree.
    - MM: Everyone can pick their own buckets. We're discussing so we can see if we can coordinate. If we cannot achieve coordination, then each must do it themselves.
    - JG: Is it super valuable to have shared buckets?
    - KN: it only improves the overall story if the page can't tell what browser it's running on, and that's a pipe dream I don't think we'll reach.
    - MM: The reason we participate in standards is to influence the spec to provide fingerprinting direction.
    - JG: there is agreement that we should be responsive to fingerprinting concerns, but not agreement on how that's done.
    - CW: As a participant in this group, I don't feel comfortable choosing Chromium's privacy strategy in this group. This is not my role, this is not the group.
    - RC: I don't have the Telemetry numbers for new CHromium based Edge, but there was a large delta between sites that used WEBGL_debug_renderer_info and those that actually linked programs. That's probably the difference between trackers and non-trackers. Other reason to have buckets: how many fallback paths do you have. D3D9 just had a huge pile of caps bits you had to check. Was nice when we added feature levels; if in FL 10, max texture size is A, and also have these other things. Limits amount of fallback code you have to write. "Pretend you're FL10", then can test whole classes of devices on your machines. I don't know what those buckets should be for e.g. Android devices. But it's a plus for developers to have well-defined buckets.
    - JG: At one point I put together artificial buckets for WebGL - didn't widely publicize, but that sort of thing is possible already.
- PR burndown
- Add aspect back to GPUTextureCopyView #873 (Austin)
    - AE: Bunch of linked issues there. In all APis you can copy D/S separately. Result is packed. Can't copy to depth, b/c have to make range 0..1. For Texture-to-Texture copies, have to copy all aspects. Putting the aspect into GpuTextureCopyView, make sure it selects all for T-2-T copies.
    - CW: without this, can't' copy D/S texture into buffer?

    - AE: correct.
    - CW: doesn't seem to be any disagreement. We really think it's needed. Think we should just add it.
    - RC: how does this handle the "plus" formats? Does developer need to know how many bits there are?
    - KN: depth channels of those aren't copyable.
    - AE: I might need to update the PR, don't know if we have a table of copyable depth formats.
    - RC: OK, if we can do it easily in all APIs, SGTM.
- ~~createBindGroup: Require a superset of the layout's bindings #1061~~ (Corentin)
    - CW: I'm going to retract. Idea was to provide more bindings than the layout required. Has bad side-effects; if you pass BGL that's too small, it'll silently work, but trying to use the BindGroup later, won't work.
    - (No disagreement)
- Add filtered texture and sampler binding types #1076 (Dzmitry)
    - DM: turned out that some FP formats are not linearly filterable. Not just that integer formats aren't filterable. Have this in the texture caps table now. This PR allows us to provide this info at the BGL level. Can validate it at BG creation & pipeline creation, not draw time.
    - DM: adds 1 more texture binding type, 1 more sampler binding type. Instead of just "sampleTexture", has "filteredTexture" and "sampledTexture". filteredTexture is the one you can linearly sample.
    - DM: similar for samplers. Regular & comparison samplers. Non-filterable samplers. Doesn't require texture filtered capability to be present.
    - DM: this is just enforcing something we know we need to. Just whether we're enforcing at BGL time or draw time.
    - MM: so now we have 5 types of textures?
    - DM: possibly? Texture bindings.
    - MM: sampled, filtered, multisampled, read-only-storage, write-only-storage?
    - CW: yes.
    - JG: minification / magnification filters are hints, not requirements, in drivers. Just keep that in mind. If you ask for nearest, you won't necessarily get nearest.
    - KR: Was that related by anisotropic filtering?
    - JG: Was made worse by anisotropic. If you asked for anisotropic, you got linear anyway. I wrote tests; didn't work well.
    - KR: Surprised. WebGL has tests that you get the right mip-level. Drivers had tons of bugs, but they've been fixed.
    - JG: Mip-level selection works well, but not filtering. That's what I found.
    - CW: If that's the case, we can't do anything except deny-list the hardware..
    - JG: It's IHVs not hardware. i.e. could be all of AMD/NVIDIA, etc.
    - CW: Also the problem here that trying to filter integers would crash GPUs - especially AMD?
    - DM: Yes.

- MM: So a nonfiltering-sampler, and I try to make it filter at bind group creation time, it'll fail?
- DM/CW: Yes.
- MM: Without this PR, I'd make a normal sampler, and..
- DM: The PR makes you specify if you are filtering or not.
- CW: If we didn't have this, you would be able to sample an integer texture with a sampler that filters, and it would crash - or per draw call validation.
- KR: Sounds like the right design decision to require this at bind group creation.
- MM: recently became aware that WGSL will have a generic function over which texture it reads from. So don't know which texture is associated with which sampler? So do we need a type in the shader that matches?
- CW: if that's the case, yes.
- KN: static analysis isn't impossible but it might be hard.
- DM: We define static use in WGSL, and if it could be one of 10 textures, we consider all of them used with the sampler.
- MM: because we don't have arrays of textures yet.
- KN: analysis would have to look at the place where texture() is called, then trace up through the call graph to see what pairs of (sampler,texture) is passed in. Really conservative rules probably wouldn't work for most apps.
- MM: So at the point that we add texture arrays, and you pass in a dynamic offset into the array, we'd have to say it could be anything so every texture in the array must be nonfilterable.
- CW: think arrays of textures will need to all have the same type. If it's an array of storage textures, have to all have the same storage texture format. Sampled textures, same texture component type. Still have texture2D of float. Array of floating-point 2D textures, they'd all be the same.
- MM: for this program, those textures have to be bound using which of these texture binding points? In the example where I have an integer texture, want to sample it non-filtered, which of texture bind point types would I use?
- DM: sampled texture.
- MM: sampling from it but non-filtering way?
- DM: yes.
- JG: sampling texture, non-filtering sampler?
- CW: name's open to bikeshedding.
- DM: we discussed mirroring those types to WGSL when discussing comparison samplers / textures. Conclusion: don't want WGSL to have those types b/c they're not used to them, and complicates SPIR-V->WGSL translation, and makes it more limited.
- CW: discussion of how that's reflected in WGSL is a tiny bit orthogonal. Up to naming, we have to take this PR. It prevents undefined behavior that crashes the GPU.
- KN: ...moving it out of draw-time validation.

    - JG: only concern I see is porting content that doesn't have this notion. Shimming Emscripten-style content that doesn't have a notion of this. Can add that later. Seems reasonable aside from not liking the names.
    - MM: wonder whether like minBufferSize we could say "I don't want to be detailed, do the slow thing in the implementation" vs. passing it ahead of time and program will run faster. 5 texture types & 3 sampler types is pretty daunting.
    - JG: I do like the idea of incremental optimization.
    - MM: not saying I don't like this change, but it's the straw that's broken my back.
    - DM: we could defer this and some other things to draw things. Artificially present in the BGL. Texture component type is one. Doesn't have to be there. When we discussed minBufferSize, argument was: unlike mBS, people know these things ahead of time, so they should be able to put the concrete values in. A matter of asking, when you have the texture binding, will you filter it or not.
    - MM: yes, that argument made sense before we had a 5x3 matrix. Now I look at this and am overwhelmed.
    - KR: I think we should do this ahead of time. Validating things at draw call time is expensive.
    - Discussion about WebGL's draw call time validation & costs between browsers. Firefox fast, ANGLE fast, Chromium's old validation was slow.
- CW: please review the remainder of these PRs offline to make progress on them.
- GPUColor: remove sequence overload from the union #1079 (Kai)
    - (not discussed)

Agenda for next meeting

- Pure WGSL tomorrow. Lots of API discussion today.