GPU Web 2020-11-02

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

Tentative agenda

- There are too many combinations of texture bind point type and sampler bind point type #1164 (also #851 and #1076)
- Method of ensuring GPUShaderModules can contain MTLLibraries #1064
- Multi-queue synchronization #1169
- Documented guaranteed swap chain formats and validation #1185
- PR burndown
    - Add validation rule for QuerySet and Timestamp Query #1161
    - Make blending state explicit #1134
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
    - Myles C. Maxfield
- Google
    - Corentin Wallez
    - James Darpinian
    - Kai Ninomiya
    - Ken Russell
- Kings Distributed Systems
    - Daniel Desjardins
- Microsoft
    - Damyan Pepper
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
- Eduardo Souza
- Matijs Toonen
- Michael Shannon
- Mehmet Oguz Derin
- Timo de Kort

There are too many combinations of texture bind point type and sampler bind point type #1164 (also #851 and #1076)

- MM: I think the goal here is to give programmers a good set they can use, but they don't necessarily know which is the best.
- MM: few paths:
    - We could try to limit number of these things
    - We could declare bankruptcy, say we get rid of all of them
    - Say this is how it's going to be, there will be a lot
- CW: there was a concern at the F2F that papering over these issues will result in draw time validation. Maybe fine for some WebGPU users, but not OK to force on all users of WebGPU given that we're trying to get a performant API
    - MM: what are we buying here? What are we getting in return?
    - CW: we had values for minBufferSize. I don't remember off the top of my head. Just for minBufferSize, a single thing, it was measurable.
    - CW: there's a concern that we'll go more toward WebGL-style draw-call-time validation.
    - KR: like to echo this - WebGL has had to add many more draw-call-time validation rules over time. Got very expensive.
- MM: what would this look like? Fewer texture / sampler bind point types. What would our implementations have to do to support fewer of them? Rewrite shaders? Add more pipeline barriers?
    - CW: cost would be that for every draw call, we have to look at the pipeline, and for every statically used texture/sampler combination, check "something" about them - check something in the texture, something in the sampler. Need to check the texture's float type, but also used as storage, have to check view dimension, etc.
    - MM: whereas now these checks are done at pipeline creation?
    - CW: and bind group creation, yes.
    - CW: I'm not able to find the benchmark Idan made, but he found that the cost per draw call per buffer that didn't have minBufferSize set was in the order of 10s of nanoseconds / 100s of nanoseconds. Not much but if you add it per texture/sampler - we'd like to do 10s of thousands of draw calls per frame and this will add up.
    - MM: doesn't D3D have GENERIC_READ? Users can choose to specify things up front or not?
    - CW: that's the third option.
    - DM: are you talking about resource states? Not relevant here. It's about whether things are legal or not.
    - MM: talking from design POV. General bind point type, "if the bind point is this particular type it means that validation is deferred"?

- DM: similar to - leave it undefined if you want to defer it. Note it's not just about performance. One factor is readability. When I look at WebGPU code I want to reason about whether it's legal. When I see a draw call I can check whether there are enough bind groups bound earlier, vertex attributes, etc. If I have to statically analyze the shader to see texture/sampler combos? Not going to happen.
- CW: doing same thing we did with minBufferSize - leaving it undefined will result in draw call validation - seems ok as long as there's the possibility of specifying everything up front for validation. In the coding guidelines to always spec bind group layout in full.
- MM: makes sense to me. You pay for additional perf with complexity.
- KN: my intuition is that that'll work well for many options. It's something we can do later. We can make the API less restrictive later. Could start with more upfront info and see what the feedback is.
- CW: if we go the route of the optional data, I think it should be clear that impls will not be expected to heavily optimize that path. ANGLE goes to extreme lengths to not revalidate state, but it adds a lot of impl complexity. Much easier to check up front.
- KR: State validation in Chromium was extremely expensive, switching to ANGLE's implementation with dirty bits is a big perf boost, but an extremely complicated implementation that's hard to keep correct. There's a big performance pitfall and it would be good to have WebGPU be more strict here. It is one of the hottest loop in WebGPU.
- MM: That's would work if it is optional, developers can get zero cost validation.
- KR: What if you opt-in additional complexity but get it wrong then implementation should paper over?
- MM: Think the proposal was an explicit switch. Either you opt-in or you opt-out.
- CW: example: texture view dimension. Undefined -> at draw time will make sure the combination you specify is valid. If you do specify it - no check is performed. If BGL completely specified - no validation. But if BGL isn't fully specified, there will be a check - is view undefined? No. Then we validate. But that makes BGL validation / creation more expensive. For each piece of bind group creation, is the layout saying "undefined"? Then you can validate the texture view.
- MM: I'm not following.
- CW: if you say "undefined" - bind group creation should always succeed whatever the dimension of the texture view you give. In bind group creation - another API hot spot - we'll have to say "is the dimension undefined? no? validate the texture view dimension matches what i have in the layout."
- DM: I wouldn't worry about bind group creation here. It may be hot by some metric, but it's another order of magnitude compared to draw calls.
- CW: OK.
- CW: then I think for fully specified bind group layout we can get validation costs to be the same as they are today.

- DM: speaking of view dimension - isn't it required for argument buffers? I don't think we can make it optional.
- MM: when I opened this issue I was thinking specifically about binding & sampler point type enums. There's a big matrix about what can and can't be inferred. Maybe not as simple as leaving it undefined. If we wanted this change what would the spec say, what are the rules, what is inferred, what's delayed until when?
- DM: before we go this route, can we get a clear idea of use cases where users will have a hard time specifying these parameters?
- MM: I have a hard time, there are too many of them, not sure what values to use.
- DM: you'd rather lean on browser errors coming your way rather than reading the spec? The errors are coming at you either way.
- MM: this is fairly subtle. I imagine situations where author will write correct code, not doing anything crazy, but they don't know which of these enums match the code they wrote.
- CW: rephrasing: I guess random WebGPU developer comes in and sees this monster of a BGL entry, flips the table over. Want to avoid that. Most people will use RGBA8 unorm textures. They're 2D textures, defaults to 2D today. They'll be single-sampled textures, filterable format. Samplers will be filterable because the formats are. A lot of WebGL stuff works this way. WebGL 1.0 doesn't have integer textures, comparison samplers, etc. A lot of pages won't touch the additional stuff.
- KN: we were talking about reducing number of things in binding types. Reduce binding types and have nice defaults for the other things? Rather than do the validation at draw time. There is one common case - can have everything default to it. Then people won't run into it and they'll still get the optimized validation.
- MM: I think that's a pretty good compromise. Good place to start, then see if we need more when developers start using it.
- DM: it's about how you shape the learning curve. Less steep for developers just starting to use the texture. Why are we optimizing for this? They'll see this in the first texture tutorial. They don't need a default. Why do we need it?
- CW: and then we end up with the Vulkan spec.
- MM: whenever I follow a tutorial I use it as a branching-off point. I use it to start with my particular goals. The defaults help there. An example in a tutorial isn't sufficient.
- KN: error messages will be pretty clear I think.
- KR: needing here because we're adding a single texture component type. Don't think it should break the camel's back, we should still require things to be specified upfront.
- MM: None of the native APIs have the number of combinations that WebGPU has. That matrix is still growing.

    - KN: native APIs hae a lot of undefined behavior. We want BGL behavior anyway. Move more information out of binding type and into more explicit fields. THink we should do that anyway, and doing that will solve this problem, with better defaults.
    - CW: how about the following. There's enough agreement to reduce the learning curve of this thing. Have identified two paths forward: either allow undefined + draw-call validation, or really good defaults that hide the details until you need them. Maybe worth writing code of what this would look like?
    - KN: I think so. It's difficult to discuss at this point.
    - MM: I'm particularly interested in Kai's suggestion of moving info out of this field and into new fields.
    - DM: it's already described in the issue. CW commented 11 days ago.
    - CW: it's different because this is thinking about undefined / draw call case.
    - KN: really taking a part from that. For filterable-texture, non-filterable-texture, multisampled-texture, give them one bind type and a different field textureKind with 3 different options. Then you say type="sampled-texture" and you get a default textureKind="filterable".
    - CW: think it would work, still good to see what the code for the simple case would look like. Think we have good agreement and alternatives, so let's move forward to next topic.

Method of ensuring GPUShaderModules can contain MTLLibraries #1064

- MM: gave 3 diff possibilities for how this could work.
    - 1) Inject additional data as a Map. Each entry point has exactly one of these pipeline layouts associated with it. Impossible for one entry point to use 2 different pipeline layouts. Not terrible, this is how Metal does it, encoded in the type system.
    - 2) Have intermediate object between ShaderModule and ProgrammableStageDescriptor. Specify shader module and entry point and pipeline layout, and compile that into an opaque object. That's when the Metal compilation would happen. First create ShaderModule, then from that additional information, "GpuEntryPoint", and that's what you'd pass into createPipeline.
    - 3) Do it similar to Metal, encode pipeline layout inside the shading language. Take various parts of BindGroup, wrap with braces, struct, new keyword, etc..
    - All of these 3 options would work fine. All have pros / cons. No preference for any of them.
- CW: all are interesting. Encoding in shading language is elegant if we can get away with it, but I'd argue strongly that we can't. GpuEntryPoint idea is interesting, one additional level of indirection. In your comment you also discussed the list of things in ShaderModule creation we need like sample mask state (?). Concern: if we take one of

these things to make compilation fast, it might be great for the first 6 months of WebGPU. Then later we have to add more shader instrumentation. Or even before that. For example: in Chromium we're looking to use vertex pulling to reduce CPU-side validation for compute shaders, index buffer validation, etc. In a world where we only give the layout, impl is not the option, or completely lose the benefit of being able to compile Metal libraries. I'd argue that the gains that any of these proposals give will disappear very quickly, because we'll have to add more workarounds / shader instrumentation / etc. Best approach is to give the whole render / compute pipeline state. That's what createRenderPipelines is.

- MM: an impl is still allowed to defer compilation until pipeline creation. You'd still do that. This is giving an impl the option to do it earlier. As for the dev cost, we were saying to make it optional. No cost to the developer, but they can opt into it for better compile perf. Seems like all around win to me.
- CW: it's a win insofar as it's one thing that'll help one impl. We could have pipeline barrier hints - would help impls on Vk and D3D12. In addition to being a hint to one impl for who knows how long, there will be very differing optimization guidelines between different impls. Maybe that's OK. A developer that specifies the defaults and gets great gains on one browser will try on another and finds that it doesn't work well. Maybe that's fine.
- MM: the other option is that perf will always be terrible in WebKit.
- DJ: it's not on WebKit, it's on Metal.
- CW: same thing on D3D12 and Vk as soon as we do shader code instrumentation - will happen any time soon. Driver bugs are going to come. createReadyPipelines solves >90% of the problem. The problem doesn't exist unless you're in a hot load. On cold load createReadyPipelines only doesn't solve the problem for dynamically created pipelines.
- MM: Metal team says dynamically created pipelines are the common case for large engines.
- DM: can you dig more information for that, that most users are doing dynamically created pipelines? Do they mix/match entry points, or change render pipeline descriptor fields? If the latter then we can't do anything. Many of these fields change how we generate the shader, so we can't up front generate the shaders. But if they just change entry points, we can have a hint about the number of entry points for this descriptor for this pipeline layout, and we can do some work up front.
- MM: are you saying - if for dynamically created pipeline, some pieces of information are changed, then ?
- DM: then we will likely end up having to dynamically generate MtlLibrary as well.
- MM: because those pieces of state affect the Metal code you generate?
- DM: yes.
- CW: also surprised Metal team gave that feedback. The last 3-4 years I've looked at GDC / SIGGRAPH presentations for big engines explaining how they do pipelines, almost all of them say that QA plays the same, a lot of dynamic pipeline generation

happens, then precompile all those pipelines and then they ship the game. They know that these are the ones used during gameplay.

- MM: the big question is how many cache misses there are. You're right, many of these are created ahead of time to prewarm the cache. As user plays the game there are plenty of cache misses. Because of the exponential problem you can't pregenerate all of them. The observation is there are enough cache misses that this is a big deal.
- KR: Surprising, when exporting a game from Unreal Engine it spends a lot of time at export time to generate all the pipelines it's going to use. They try to enumerate all the possibilities of the pipelines ahead of time so they don't need to generate them at runtime. Even more, they run the game through QA to be sure to catch more.
- MM: don't know whether that's true.
- CW: without passing judgments based on our intuition - because we're receiving very differing guidelines - like you said, adding just a hint with just the layout is fine. It's not the end of the world. Just ~20 lines of validation.
- MM: we're willing to agree to a hint. At a risk of reopening this argument - if we go down the hint route, if the author does provide a hint, then later when they create the pipeline for real, the hint should match what they specified at pipeline creation. Want to avoid compiling things twice.
- CW: yes. Adding a hint like that is fine. Chromium won't be able to use the hint. wgpu probably won't be able to use the hint.
- DM: we're mostly in agreement with Corentin about what parts of the pipeline descriptor we'll need. We'll use programmable pulling, so will need vertex attributes. Agree it's weird to add a hint for one impl for one backend API. If we want a hint, it'll need to take most of pipeline descriptor, and most of info for the entry points - most of what I suggested in the precompile pseudocode. But again, this is a complex task, and feels like we should think about it after we release MVP.
- MM: of the three options I listed, #2 and #3 would be breaking changes.
- CW: they are; but maybe GpuEntryPoint could be added as an overload in various places to not make it a breaking change. Probably little value in this hint outside of WebKit on Metal. When WebKit runs on other APIs it won't add value. I'd rather not shape the API around that. Option 1 seems the most palatable.
- CW: [discussion about potential plan in Chromium]
- MM: that plan's acceptable.
- CW: assuming createReadyPipelines.
- MM: createReadyPipelines is a good idea, we should have that anyway.
- DM: if Unity comes to us and asks why we should provide it, we'll tell them that it might help Safari on Metal specifically?
- MM: or other browsers.
- CW: super unlikely.
- DM: we'll need more of the pipeline state.
- MM: if we're discussing a new feature that will need to modify the Metal source code generated from the WGSL shader.

- CW: today, both Firefox and Chromium will need more than just the layout. The vertex state, at least.
- MM: that's these browser's choice.
- CW: we're not going to go back to CPU side validation of index buffers, as far as we can avoid it. Yes, technically, we can do without it, but then put a whole optional RenderPipelineState and you can put anything. If you set it it's binding, if not it's not binding.
- MM: that defeats the purpose of this. Users know pipeline layout ahead of time, but not the entire state. We need the rest of the layout but not the rest of the state. Lumping it into one big object graph because developers won't be able to specify it.
- KN: question of whether Metal apps work that way naturally, or because of the way the Metal API works.
- DM: I don't understand even in your impl how you'll do this. Let's say they specify the layout, but use a multisampled pipeline. You'll have to inject your sample mask with theirs.
- CW: that's the function constant. For all of the cases, sample mask is functionally constant.
- DM: constant doesn't change anything. If you write to the sample mask you ruin your early depth, esp. if you don't … at all.
- CW: can see if it's not statically used with constant propagation / dead code elimination, etc.
- MM: metal compiler does optimize after applying function constants. That's why they exist.
- CW: getting back to the discussion: if we add the layout today, it's fine. We can live with that. The layout today, for every functionality we add in the future, have to wonder wheter we have to to add more hints. Or we eventually end up with the whole render pipeline descriptor.
- MM: or you put the contents of the RPD, each element is optional.
- CW: that's what I meant. Sorry if that wasn't clear.

Multi-queue synchronization #1169

Documented guaranteed swap chain formats and validation #1185

PR burndown

- Add validation rule for QuerySet and Timestamp Query #1161
- Make blending state explicit #1134

Agenda for next meeting

- CW: please familiarize yourself with multiqueue sync PR Dzmitry put up. Swap chain formats. What’s left on the agenda for today.