GPU Web 2020-09-14

Chair: Corentin
Scribe: Ken
Location: Google Meet

Tentative agenda

- Viewport rect must lie entirely within the current attachment size? #373
- Separate linear sampler binding type? #1034
- Allow writeTexture into depth textures? #1043
- Method of ensuring GPUShaderModules can contain MTLLibraries #1064
- Query API: Disallow resolve the unused queries or resolve them to 0? #1072 (was spun off a PR that made it in the homework deadline)
- (Late but no homework needed) What OSes and hardware are we targeting? #1069
- PR burndown
    - ~~Add valid usage for resolveQuerySet #1042~~
    - createBindGroup: Require a superset of the layout's bindings #1061
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
    - Myles C. Maxfield
- Google
    - Austin Eng
    - Brandon Jones
    - Corentin Wallez
    - James Darpinian
    - Kai Ninomiya
    - Ken Russell
- Kings Distributed Systems
    - Dominic Cerisano
- Microsoft
    - Damyan Pepper
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
- Matijs Toonen
- Mehmet Oguz Derin

- Timo de Kort

Administrivia

- Instead of cancelling the WebGPU meeting, replace it with WGSL? How to move WGSL along faster?
    - DJ: Have a lot of issues to discuss in WGSL and it takes time. Probably need more contributions to the spec, there are resolved issues that aren't in the spec yet. +1 to having more meetings.
    - DJ: we usually make progress on issues within a couple of weeks. Not sure about David / Dan's time. Everyone else comes to these meetings too.
        - Greg and Tex too.
    - CW: David / Dan could join this meeting, but their agenda's free for it.
    - DJ: would need to check with Greg; he's very active on the calls.
    - CW: we can check for that, if they can reserve this slot. Turn WebGPU mtg into WGSL meeting. Do we need to increase length / frequency of WGSL meetings?
    - DM: a lot of groups meet for 1.5 hours - still acceptable range.
    - DJ: I'm usually burnt out 45 minutes into these meetings. :)
    - MM: think 1.5 hours at a time is too much.
    - DJ: I favor more, shorter meetings - then the difficulty is scheduling them with 10+ people. Might have to try.
    - CW: or, more aggressively turn WebGPU meetings into WGSL meetings.
    - DJ: could definitely take agenda items from WGSL, as long as the people invested in them know about the agenda in advance.
    - CW: let's check with Greg and Tex if they can join this meeting. Every other week, have WGSL items on the agenda.
    - DJ: let's discuss tomorrow if we don't hear back from Greg and Tex.
- TPAC.
    - CW: agreed a breakout session about WebGPU would be nice.
    - CW: W3C suggested putting up a small video a couple of weeks before the breakout session. Could make some small demo videos on maybe Chrome / Firefox? What else could we do?
    - KR: could WebKit run any of these demos?
    - MM: we don't have a WGSL compiler yet.
    - CW: we can sync up offline on what the content should be. Maybe shorter video rather than longer one. Will start a thread on internal-gpu@.
    - CW: Immersive Web would like to talk about WebGPU at TPAC.
    - RC: I don't have the details about this yet.
    - BJ: haven't decided on appropriate time for this yet, but yes, would like to talk.
    - CW: Machine Learning for the Web group will be having a workshop, will mention WebGPU for some things & have some questions, I'll be there. This is this Thursday and very early for California. Should be fairly small.

- VF2F?
    - CW: enough topics to make this profitable?
    - DM: would be nice to close multi-queue topic at a F2F, unless we can close it earlier. But, yes, WGSL still has a lot of stuff, and we could use more time.
    - CW: OK. Will send a poll for F2F times, some weeks before / after TPAC.

Viewport rect must lie entirely within the current attachment size? #373

- KN: been a long time since we looked at this in depth.
- KN: Metal docs say, xy rect of viewport has to lie entirely in current attachment size. WebGPU doesn't currently require this.
- KN: if we clamp it, changes the semantics. Can't just take the value passed in to the API and clamp it, passing the result to Metal.
- KN: so do we add the restriction to the API?
- KN: follow-up, do we do the same thing for the scissor rectangle?
- DM: the users can always work around this, so makes sense for the API to have the restriction.
- KN: unfortunate, because it makes the API brittle, but makes sense that we have the restriction. Would like to set things 100x100, and if user shrinks the window, automatically handle.
- MM: what's the error?
- KN: validation error.
- CW: do we allow flipped (especially y-flipped) viewports?
- KN: not sure. If we want to, have to emulate it.
- CW: never mind, let's not have it.
- KN: probably not that hard to emulate it. If you had it, could even emulate clipped viewports.
- KR: that would have the cost of a blit at the end?
- KN: thought we had a better solution. You can do a y-flip in the shader and the other flips / clips.
- KR: It gets pretty tricky. In WebGL it even hit clipping of primitives. AMD vs other vendors did different things and weren't easily worked around, even with shader transformations. Here, despite the fact that it's unfortunate, surfacing the hardware restriction is probably the best thing to do.
- CW: Okay, so no y-flip, has to be contained in the attachment size.
- KN: we should look back at the last time we discussed the y-flip. I don't remember the context.
- CW: unless we find something that allows specification of y-flip…
- DM: I'm pretty sure we can support it, but looking back at the investigation would be nice.
- KN: thought it was an optional feature in Vulkan.

- CW: at least something we're sure of is that the viewport has to be inside the attachment size. TBD on the y-flip.
- KN: scissor also? think that clamping scissor rect to attachment size is semantics preserving, since it doesn't change viewport coordinates.
- DM: the specific issue on Metal is it doesn't like zero-sized scissor. Would have to ignore all calls coming into render pass; this is doable.
- CW: no, because of writable storage buffers and vertex shaders.
- DM: oh, wonderful. :)
- KN: OK to have validation rule that the scissor rect can't be 0-sized.
- DM: the clamped one, not the user-provided one.
- KN: ah, putting scissor outside the attachment entirely.
- CW: let's put the validation rule for the viewport in the spec, and do investigations for the rest.
- MM: hope this will turn into at least 2 tests.
- CW: will be much more. :) we already have dozens of tests for this in Dawn, and CTS will be even more complete.

Separate linear sampler binding type? #1034

- DM: Vulkan, at least, has capability of a format to be linearly sampled. Have this in WebGPU spec too. Have a table, one column indicates linear sampling / filtering.
- DM: Vk is strict: if you try to linearly sample something not supported, today it's undefined behavior. Trying to clarify this with the group.
- DM: for us to validate this early, like to know at BindGroup creation time, that something's to be linearly sampled. That's what the binding type would let us do.
- DM: for shader side, someone suggested we don't need the separate type. If we did have the type, converting from SPIR-V into WGSL would be hard, and we can still validate things at pipeline creation.
- DM: so we think we should have the separation on the API side, but the WGSL side is unchanged.
- CW: only suggestion - maybe sampler binding type would mean "filtering sampler" and there would be a "non-filtering sampler" for integer textures, to keep things simple for the developers?
- DM: not sure I understand.
- CW: just bikeshedding the binding types.
- DM: you're saying, the default sampler should be linearly sampled?
- CW: yes, because that's 90% of the use cases.
- DM: but it'll fail to create pipeline?
- CW: just saying "sampler" and "non-filtering sampler".
- MM: you have a sampler - can be linearly filtered or not - and texture, which can have floating-point component or not. You put these in the BindGroup separately. How does the new binding type tell you this sampler can be used with this texture, but not this one?

- DM: I'm mistaken, it's not at BindGroup creation time. You can only tell the things about the sampler and texture individually. Verified at pipeline creation time. Still unresolved: at pipeline creation time we still don't know the user's texture format. R32F texture, can't tell whether it's valid to sample linearly from it or not. Waiting for feedback from Vk WG.
- CW: and D3D12?
- DM: don't know whether it's undefined in D3D12.
- MM: if we accepted this proposal, then some behavior (which we don't know what it is - undefined, maybe not) would be disallowed. Would still like clarification on whether this is disallowed.
- DM: unfortunately no. The PR protects us from linearly sampling different component types like integer / signed-integer. But won't prevent us from sampling floating-point component, which is actually unsupported. Could block this and get back to it. Linearly sampling float probably won't explode your machine, so maybe we can spec it as something else than undefined.
- CW: but linearly sampling integer textures might explode your machine?
- DM: yes, that's what the PR guards against. We only know the texture component type.
- CW: have to disallow filtering of non-float component type textures. Also, on D3D12 / Vk, might be undefined behavior - but hopefully only undefined value - sampling Float32 textures (R, RG, RGBA).
- DM: can't make it clean for all formats - depth comparison formats, Vk spec says - if linear sampling isn't supported, you don't have a choice / can't even check if it's supported. (?)
- MM: if it was undefined behavior what would we do in response?
- DM: could have the texture format be part of the bind group for textures, not just storage textures.
- CW: could have Float32 component type.
- DM: it's not just Float32.
- MM: need a matrix, rows are textures, columns are samplers, and need to find out what's legal for the 3 APIs.
- DM: we have that, know what's legal, and have this in the WebGPU spec. Texture capabilities table at the end. Just don't know how to enforce it properly.
- CW: we add filtering / non-filtering sampler at binding type. Static analysis at pipeline creation, know what sampler you're using per texture. Then, if Float32 textures are OK to use (undefined value, not undefined behavior), then we say, if component type is float, have to use filtering, otherwise non-filtering. Not-float, not-depth-comparison: have to use non-filtering sampler. If undefined behavior to filter Float32 texture: then add new texture component type IEEE float, and that should be fine.
- DM: if we generalize it, would have "filterable float" and "non-filterable float" as 2 different texture component types. Not the worst.
- CW: that's only if it's undefined behavior and not undefined values. Can you write that down in the issue?
- DM: OK.
- MM: why is undefined value acceptable?

- DM: it's a good question. Samplers are inherently fuzzy. You don't know what anisotropic sampling will give you, for example. Hardware-specific.
- CW: pretty sure there's hardware that uses fixed-point internally and loses precision.
- MM: in programming languages, boundary between undefined value and behavior is nonexistent.
- CW: you can set your processor to modes that are non-strict for IEEE. And where floating-point precision can change per environment. That's true for CPUs. GPUs are even fuzzier. Viewport computation might be fixed-point, so number of digits after the decimal place might be 0. No way to paper over this. So many small things we can't paper over.
- MM: if it's truly undefined value then the answer could be 0 for every texture; not helping anybody. Think for first WebGPU version we should restrict it.
- CW: at least for Float32 would like to get answer from Vk and D3D12 groups. If the value has to be between the two samples then it's not that bad. Filtering float / non-filtering float would be confusing.
- MM: think that's OK as long as all 3 APIs agree.
- CW: that's the plan. Metal has this defined, have to consult on the other 2 APIs. That's the AI.
- DM: would like to see Myles/Damian/Rafael come back with information on this.
- DP: we (Microsoft) can take this as an action item. Trying to figure out where this component type shows up in D3D12.
    - CW: it doesn't. This is about filtering DXGI formats R32_FLOAT, RG32_FLOAT, etc., and whether that's defined. Let's investigate offline, and write down in the issue the potential API we can do.

Allow writeTexture into depth textures? #1043

- DM: seems we could easily do that. Other alternative, fairly complicated. Create buffer as storage buffer, put data in, bind it, create pipeline reading from buffer and writing to fullscreen quad with gl_FragDepth writing. Not trivial, but possible. For us, fairly trivial to support in writeTexture.
- CW: currently, WebGPU assumes depthTextures' values between 0..1. If we allow this then we should allow unclamped depth textures in the future.
- DM: it's up to us. Could make it non-clamping.
- MM: with this PR, I can upload data from CPU safely, but can't copy data from GPU into a depth texture?
- DM: other than rendering into it.
- CW: hunch is, it's technically doable, but because we can doesn't mean we should for V1 / MVP. Maybe can wait for people to ask for ti.
- DM: we've been asked for it, that's why I brought it up.
- MM: if we can do it, and it's free, why not do it?
- DM: it's one more feature, and there's work behind it.
- KN: it's also not future-proof.

- CW: if we allow outside 0..1 in depth textures, we have this entry point that does implicit clamping. writeTexture for all other texture types just takes the data and puts it into the texture. For depth texture uploads, we do this clamping and maybe even conversion - implicit operations. How do we have a writeTexture that just copies the data in the future?
- CW: also not free - for each feature we have to write the spec and exhaustive conformance tests (and implementation).
- CW: just saying it's a -1 from me, but not a "-5" - I'm not blocking it.
- DM: we don't have to spend a lot of time on features not needed for MVP.
- CW: would be good to get a list of all the flexibility we wanted to add into WebGPU but didn't have time for the first version.
- MM: think that list would be contentious.
- KN: more like an agenda for future discussion.
- CW: not saying these won't ever be in the spec, just things to potentially add in the future.

Method of ensuring GPUShaderModules can contain MTLLibraries #1064

- MM: WebGPU has 2-stage shader compilation process - create shader module, then pipeline from it. Every backend API has 2-stage shader compilation pipelines. Reason they're valuable - expect first compilation stage to be heavier than second stage. Also first stage should be less frequent.
- MM: usage - someone puts lots of source code in 1 big file, compiles it once, has intermediate representation object, then create whole lot of pipelines from it.
- MM: since all APIs match this 2-stage compilation model, seems a waste that they can't be implemented in terms of each other. In Metal, we have to do the text generation at pipeline creation time.
- MM: not particular about the method, but would like to solve this problem. I came up with two possible ways to solve it. 2) adding info that Metal would need inside shader module creation, so it would have the info needed to make Metal library; 1) put the same information that's missing inside the shader text itself.
- KN: the order's actually reversed in the issue.
- CW: not sure about Metal, but in D3D12 and Vk, pipeline creation takes longer than shader module creation. The driver will do specialization based on a lot of info, including optimizations across vertex / fragment shaders.
- MM: then why separate compilation?
- CW: validation. So you can extract multiple entry points from same source code, so you can share source code, bindings, etc.
- MM: I mean why do the 3 APIs have 2-stage compilation?
- CW: because they can do some amount of work there.

- MM: then we should do some work there too. We should do the same kinds of work that the 3 API backends do.
- CW: in a lot of Vk drivers, you take SPIR-V, put it in LLVM IR. Then at pipeline creation, it takes only the necessary functions, puts the right environment in place, etc.
- MM: lowering to LLVM IR is something we should be able to do. Running Clang is a big lowering step.
- CW: Metal shading langage team pointed out in an online video https://www.youtube.com/watch?v=VFHYaH5Vr4I that register allocation, scheduling, etc. is more than 50% of the compilation time. Not all hardware has fixed function stages for certain operations, like vertex transformation. These are added at pipeline creation time. Driver, since it has access to LLVM IR. We don't have this deep access in the driver.
- MM: if we utilized shader module creation then we'd have that level of optimization in the driver.
- CW: then you're forcing users to provide all the information at shader module creation time.
- MM: that's what developers provide in Metal today.
- CW: no, there are things like vertex format that are provided at pipeline creation time.
- MM: I'm proposing we add that information.
- CW: Developers don't have access to all this information at shader module creation time. If we're doing this, we're going to be asking everyone to create an entire pipeline at SM creation time.
- MM: re: first concern: the set of additional information is exactly that which is required to create a Metal library. Authors already supply that.
- CW: In WebGPU we have additional constraints. There's the constraint that on Metal, if you want to transform to do programmable vertex pulling for robustness, this depends on the vertex state. We also have the sample mask on the render pipeline which is injected into shader code. That would have to also be given at SM creation time. If we find bugs or need emulation, everything that might require a workaround will need to be exposed at shader module creation time. My suggestion is to use a cache where you key it on anything in the impl that might change the resulting MSL. Either you have a single metal library there, or your cache gets very filled, which is good for the developer and saves them from setting up many different variants.
- KN: you save the app work too - they don't have to do as many createShaderModules.
- MM: I think there's a lot more to discuss here - running low on time.
- CW: let's discuss more on the issue then.

(Late but no homework needed) What OSes and hardware are we targeting? #1069

- KN: I compiled what I could remember, but does anyone remember any additional decisions we made to drop devices not listed here?

- KN: if anyone has particular opinions, please comment on this issue.

Query API: Disallow resolve the unused queries or resolve them to 0? #1072 (was spun off a PR that made it in the homework deadline)

- CW: everyone, if you could voice your opinion on this it would be appreciated. What happens with unused queries? This is blocking implementation.

PR burndown

- Add valid usage for resolveQuerySet #1042
- createBindGroup: Require a superset of the layout's bindings #1061

Agenda for next meeting

- CW: we have two different multi-queue proposals. Plus, multi-queue investigation. Think we should spend time on it next meeting. Please, familiarize yourself with this topic.
    - Multi-Queue Investigation #1065
    - Strawman Multi-Queue Proposal #1066
    - Multi-queue proposal with explicit transfers #1073
- CW: Continue discussion on today's topics if there's new things.
- DC: understand someone (Hao at Intel?) is working on a timestamp implementation?
- KN: not really related but they work together.
- DC: he's using them for performance metrics in TF.js WebGPU backend.
- KN: they let you measure time on a single queue.
- DC: I'll make some notes on Github about that. We'd want to do performance metrics on WebGPU.