GPU Web 2021-09-13

Chair: Jeff Gilbert
Scribe: Ken / Kai / others
Location: Google Meet

Tentative agenda

- CTS updates
- Issue triaging / milestone discussion (timebox 10m)
- TF.js status and feedback (timebox 10m?)
- timestamp-query is unimplementable on TBDR architectures #2046 (alternative proposal in Hao's comment)
- GPUExternalTexture: Deal with Chroma Reconstruction #2098
- Stretch
    - Should depth-stencil render attachments require views to have aspect = "all" for combined depth-stencil formats? #2062
    - Implicit GPUTextureSampleType for texture_2d<f32> #2064
    - (stretch) Feature request: ignore shader writes to color attachments #2060
- Agenda for next meeting

Attendance

- Apple
    - Myles C. Maxfield
- Google
    - Austin Eng
    - Brandon Jones
    - Kai Ninomiya
    - Ken Russell
    - Shrek Shao
- Intel
    - Hao Li
    - Jiajia Qin
    - Jiawei Shao
    - Xing Xu
    - Yang Gu
    - Yunchao He
    - Zhaoming Jiang
- Microsoft
    - Rafael Cintron

- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
- Michael Shannon

CTS updates

- KN: talking with DM about getting CTS running on wgpu via deno (Rust-based JS native engine)
- Exciting! will have test coverage on multiple impls.
- DM: already running on CI on every PR! Gradually enabling more tests.
- MM: as distinct from running in-browser?
- DM: right. It's not in browser, but via deno.
- MM: is it easier?
- KN: way of running it on wgpu's CI without building Firefox.
- JG: it's running on the "interior" project. Working on a harness for Firefox.
- KN: yes - also very exciting.
- KN: couple of good testing updates. DM was just running tests, weren't in his current checkout, because his checkout was 1 day old. Moving pretty fast. :)
- JG: that's great news.

Issue triaging / milestone discussion (timebox 10m)

- JG: MVP vs. V1.
- JG: last week's WGSL call - surprise from ppl not on previous WebGPU API call where there was rough consensus to change milestones from dealing with pre/post MVP, to pre/post V1.
- JG: had quorum that day. Some WGSL ppl were a little gun-shy about characterizing things as V1 vs. MVP. But already changed everything. Don't think worth changing everything back.
- JG: felt Chrome's OT constituted MVP officially, and I don't agree with that.
- KN: Yes. We didn't really have an MVP definition. Using it to tell whether a change was breaking or not. "Stable point" should be called 1.0. Would like to have something before 1.0, but because we haven't had such a label it's been hard to have an MVP label.
- KN: OT doesn't really have anything to do with it.
- KN: if we can come up for definition for MVP, would be great to have a milestone and move some things into it. API side, it was being used as a marker of a breaking change - needed before 1.0.
- JG: imp't aspect: not that we know what MVP is, but - things that are requirements for V1, vs. things we want in V1 that we don't necessarily need for it. MVP = only the things we need for V1. But the initial product's not just the MVP. May not be a super important distinction. If it is imp't and comes up in another WGSL issue, we can add more tags if we need to.

- KN: I think we have been fairly minimal about MVP; most things are in V1 milestone because they involve breaking changes. Some very core things like - compressed texture format stuff, etc. But conservative about putting things in V1 bucket, formerly MVP bucket. Lot of things in it, not sure how to divide it up. Nice if we could though.
- JG: main idea - there are shipping requirements, those are MVP.
- KN: i see.
- JG: and other things we may want to talk about for V1 - those are in post-MVP category. Deferring them, but still before V1, to keep things open. Thought is, we don't absolutely need those changes.
- KN: on API side, we should probably stop punting things. But maybe WGSL isn't at that point yet. That definition seems reasonable.
- YH: from spec standpoint - there are V1 targets. Will we add labels for MVP features?
- JG: idea - anything previously labeled MVP, we just label for V1. Just difference in labels. Room for things we may want in V1 but not critical. Still in V1 label for now.
- KN: makes sense. YH do you mean inside spec document?
- YH: yes, in incomplete areas.
- KN: probably won't have that distinction in the spec. Stuff that's in the spec is "done", unless there are missing details - adding those could be MVP or V1, depending on their impact.
- YH: so is spec targeted at MVP or V1?
- KN: spec should always be trying to move toward V1. Can go past V1 in some aspects if it wants to, for example adding details about optional features that wouldn't have been called V1. If they're done, they're done, and in the spec. Not parts of spec that are V1/MVP/Post-V1. Whatever wording's in the spec is written.
- YH: OK, so my understanding - spec's V1, and MVP is temp status for implementation.
- JG: that's right, that's a good way to think about it.
- MM: once V1 ships in 2+ browsers - marking things as V1/V2/V3 is meaningless. MVP, V1 only useful for the next few months. After that, there'll be no point to these milestones.
- KN: that's correct, except for things that are agreed-upon / tested behavior that we don't know/have yet.
- KR: Is that because there’s no versioning capability besides optional features? Is nothing new going in to core ever again? Does everything need feature detection?
- MM: way this works - features are just added to spec.
- KN: planning on living spec. Everything added after 1.0 should be feature detectable. If we need to add feature detection for each new feature we'll do that.
- JG: MVP is more project management. Now just talking about it in terms of 1.0.

TF.js status and feedback (timebox 10m?)

- JQ: Have been working on it for two years, lots of input and development. From the data we’ve collected, WebGPU behaves really well, and is 1.1-3x as fast as WebGL.
- JQ: issues:

    - Out-of-bounds clamping. Big perf regressions. 30%. Reported on WebGPU Github. Some discussions. Hasn't been solution. Currently using vec4 instead of floats to reduce impact of clamping. But still about 8% hit.
    - https://github.com/gpuweb/gpuweb/issues/1202
    - We investigated that shared memory array clamping is the heaviest one. So I am wondering whether you can provide a special optimization path for shared memory robustness access. Or give users a hint on how to use the shared memory index so that we can avoid index clamping under the hood.
- JQ: hardware limitations
    - Ex. max workgroup shared memory size.
    - Reduced usage of shmem from 32KB -> 8 KB. 7% regression on some modules on Windows.
    - One module: 2x slower than before on another platform. shmem size probably the culprit.
    - Can we expose those maximum values per device?
    - MM: you're asking for more workgroup shared memory?
    - JQ: yes, if hardware supports more.
    - KN: we have support for this in the spec: maxComputeWorkgroupStorageSize. There's a limit you can raise when device is created, but we don't implement it in Chrome right now.
- JQ: timestamp query APIs
    - Imp't for us to debug perf. E.g. perf impact tuning same operator's perf with different input shape/size.
    - So we need to know each dispatching call's accurate time for performance.
    - But, won't be convenient to get these measurements if we can only measure on the pass level as in the TBDR timestamp proposal.
- JQ: warmup time.
    - Currently WebGPU has long inference time compared with WebGL. Time mainly in shader compilation. Compute pipelines seem not cached in browser. Want browser to support caching compute pipelines. 2x-10x slower than WebGL for first run / first-time inference.
    - MM: spec allows caching these modules, not sure if a browser does so yet. Ideally before changing spec, browsers would implement some kind of caching.
    - JQ: testing on Windows, Chrome Canary.
    - KR: Dawn's DX12 backend?
    - JQ: yes.
    - KR: so maybe fxc is the main problem. Have had a lot of issues fxc just in WebGL fragment shaders, can imagine more problems using it for compute shaders.
    - Discussion about dxc, and SPIRV->DXIL.
- JQ: dxc support. What's the plan? We'd like to try SM 6.0 features.
    - Subgroup support, for example.

    - MM: float16 support too? fxc only supports min-something types, but dxc supports real float16?
- MM: should we open issues for all of these? Try to discuss one-by-one?
- JG: they're more or less in bullet points already. 4 issues. Perhaps have discussion on how to mitigate. JQ could you file them?
    - JQ: yes. We can do it.
- JQ: would like to have short discussion about out-of-bounds clamping. One proposal: is it portable to have special optimization pass for shmem robust access? shmem is fixed size. Usually use local invocation ID as index. Max one depends on workgroup size, which is constant value. If WG size is smaller than shmem size, easy to avoid the shmem index clamping. Could solve our problem. Does that sound good?
    - https://github.com/gpuweb/gpuweb/issues/1202
    - MM: think the compiler knows the workgroup size and shmem size. Can just detect, no API change needed.
    - JQ: Tint inserts clamping of all indices. Driver doesn't do this optimization. If we don't insert the clamping, perf is ~2x faster.
    - MM: do you have potential solutions? What would you like this group to do?
    - JQ: for Tint project, can you analyze the shader and find that the memory index will never exceed the max value? Maybe don't need to insert the clamping.
    - JG: sounds viable. Sounds more like a Tint enhancement request than a WebGPU spec bug. Maybe better to talk about this in the WGSL meeting.
    - KN: agree, this would be a Tint / Naga enhancement. Compiler optimization. Involves some analysis. Think purely implementation issue. If reason to believe not viable with current state of the language - can discuss. But today, think it can be done.
- MM: you mention you use workgroup memory extensively. Are these shaders created at runtime? Do you know before the app starts running what the full text of the shader is?
    - JQ: we construct the shaders at runtime. Only impacts the first-time inference. Shader cached afterward.
- MM: when you generate these shaders, is each one distinct? Or are most of the shaders similar, with a few tweaks?
    - JQ: if same operator, almost only one shader. Different ops, shader will be different. Some shaders - based on input shapes, workgroup size will be different, maybe have different shaders.
    - MM: I see.
- JQ: another issue: hardware limitation. Opinions on this?
    - KN: the API already defines a way to raise the limit when you create a device. Our impl doesn't let you do that yet. Only exposes base limits; doesn't let you see the adapter's real limits. The spec supports this.
    - JG: Chrome will eventually support this?
    - KN: yes.
- JQ: I'll go through my issues and post them into Github.

- JG: thanks so much for coming and glad we had this meeting at this time so you could attend!

timestamp-query is unimplementable on TBDR architectures #2046 (alternative proposal in Hao's comment)

- HL: another proposal. Myles' is great - thanks for it. But eliminates the ability to measure time between draw calls/dispatches on immediate mode devices. Need it for TF.js profiling.
- HL: from perf data on issue #2046, don't think we'll split dispatches into passes.
- HL: suggest new timestamp API, something for TBDR, and immediate-mode devices. Like Metal, supports two kinds of timestamps.
- HL: we've discussed, timestamp query's a developer feature. Detect which kinds of queries are supported on their platforms.
- MM: are you proposing two separate, distinct extensions?
- HL: yes. Current extension is, we can call writeTimestamp between draw calls/dispatches. And the other one is to record timestamps in the pass descriptor.
- HL: We batch multiple dispatches into compute passes thanks to your performance investigation that compute passes have overhead cost.
- MM: TF.js runs on iPhones, and M1 Mac (I believe). On iPhones / M1 Macs, what does TF.js do there?
- JQ: why do you think it will change with other platforms? Currently it's the same.
- MM: on immediate-mode GPUs (like Intel's), you ask for current time, then execute dispatch, ask for current time again. If you can't ask for current time around individual dispatches then you lose information. Need info about individual dispatches. Correct?
- JQ: yes.
- MM: on iPhones / M1 macs, can't get info about single dispatch. Can't write that. You have to get current time around an entire pass. Believe TF.js (or TensorFlow) runs on iPhones / M1 Macs, so what does TF do on those platforms where it can't use that kind of code?
- JQ: I think we can support both. Can get whole pass's time, and each dispatch's time. Each dispatch's time is useful for profiling. And on Mac can get whole pass time.
- MM: what benefit do you get get from profiling draw calls/ dispatches rather than passes?
- JQ: need to know operators' accurate time. One operator may be executed multiple times with different input shapes. Different input shapes, different performance.
- MM: do you do that at runtime? Or does the app developer have to be involved in tuning process?
- JQ: we do the profiling in profiling mode. Not in release mode.
- MM: sounds like it would be better for developer tools to expose. Our developer tools on iPhones will give you much finer detail than an individual draw call. iPhones will give you info about assembly statements in shaders.

- JQ: thanks for that. But we need it on other platforms. If only can measure at pass level, how to evaluate on Windows?
- MM: think browsers should expose developer tools.
- JQ: so depend on developer tools?
- KR: objection to not giving Intel the opportunity to spec these extensions.
- DM: would be nice to have detailed timestamps in the APi. But not in core. My priority is to get the core finalized. Would like us to not spend too much time on timestamps specifically. for Intel team, native GPU tools give you superior info to anything we can give in the near future. Would like to not rush this in.
- MS: we discussed it at our company. For us, it's more of a developer feature. Would like to keep what we have, but we'd turn it off before distributing an app. So we don't care if it's officially in, but would like to keep the functionality we currently have. (writeTimestamp between calls/dispatches.) But we don't care if it's ever published.
    - MM: when you say developer feature, you mean for the employees in our company?
    - MS: yes. We wouldn't publish to the web with it init.
    - MM: if you have to start browser with a flag, that's fine with you./
    - MS: yes. And it's OK if the interface changes.
- SY: think TF.js is using timestamp query because we have regression prevention mechanism. We run TF.js benchmarks every day. We'd use more powerful tools for analysis.
- MM: every proposal offers regression prevention.
- SY: if you can only get the pass timestamps you can't get the individual operator times.
- JQ: if we support timestamp queries for individual dispatches behind a flag it's fine.

GPUExternalTexture: Deal with Chroma Reconstruction #2098

- SY: hoping for more comments on this. Internal discussion about correctness of GPUExternalTexture. Pre/Post filters are questions.
- SY: how to handle chroma reconstruction step in implementations?
- SY: Does spec need to add words for GPUExternalTexture correctness? E.g. we don't commit postfilter?
- JG: sounds like you're asking - are we committing to make sure we're decoding these with proper chroma reconstruction? Or want to specify chroma reconstruction for video?
- SY: first, how can we support it? We could choose one as impl detail. Other, more wide support - e.g. implicit reconstruction, or don't support postfilters in the spec, or just ignore it?
- JG: wish we'd timeboxed earlier to have time to discuss this.

Stretch

- Should depth-stencil render attachments require views to have aspect = "all" for combined depth-stencil formats? #2062

- Implicit GPUTextureSampleType for texture_2d<f32> #2064
- (stretch) Feature request: ignore shader writes to color attachments #2060

Agenda for next meeting

- JG: are we having a WebGPU APAC office hour this week?
    - KN: EMEA/APAC office hour's still on the calendar for the usual time.
- MM: bunch of issues that require Intel. We could say, next week's WebGPU call is at this time also.
    - JG: maybe...trying to balance it. Not sure whether it's better to go for continuity between meetings, or reduce latency with other people.
    - MM: what does Intel team think?
    - SY: for my issue I'm not in that much of a hurry. Need input from other team members in the interim.
    - JQ: for TF.js, I'll file all the issues I have on Github. Want to collect more discussions there, see feedback. See if we need separate meeting to discuss.
    - JG: OK. GIven that feedback, I think meeting at EMEA time next week, but should consider another APAC-friendly meeting the week after.
    - KN: thikn that would be good.
    - JG: OK, let's do that - next week, 12-1 PM Pacific time. Tentatively, the week after, this meeting again at this time slot. (5-6PM pacific)
    - JG: thank you Jiajia and Shaobo for presenting things
- TPAC meeting(s)?