GPU Web 2019-08-05

Chair: Dean (Corentin on vacation)
Scribe: Ken, Austin, Kai
Location: Google Meet

TL;DR

- Conformance test suite
    - CTS repo moved to github.com/gpuweb/cts
        - It is written in TypeScript and built to JS.
    - Ready to write tests but not with shaders.
    - Working on integrating with Chrome infra.
    - CTS will be developed in gpuweb/cts and builds will be exported to WPT.
        - To be automated.
- Multi-threading #354
    - MM identified 7 categories of parallelism used in Metal.
        - Texture/buffer uploads
        - Shader compilation
        - Pipeline state creation
        - Parallel render pass encoding
        - Each thread records its own command buffer
        - Multi-queue on same device
        - Multi-queue on different devices
    - Texture/buffer are nontrivial because they have mutable state.
    - Shader compilation and pipeline state should be easy because they're totally immutable once built (except device loss).
    - It's hard to design a JS API that works well for parallel command buffer or render pass encoding.
    - KN to send issues for the easier cases: shader/pipeline, buffer/texture

Tentative agenda

- Compressed texture copies #363
- Multi threading
- Coordinate systems and winding #379
- PR Burndown
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
    - Justin Fan
    - Myles C. Maxfield
- Google
    - Austin Eng
    - Idan Raiter
    - Kai Ninomiya
    - Ken Russell
    - Shrek Shao
- Intel
    - Yunchao He
- Microsoft
    - Chas Boyd
    - Rafael Cintron
- Mozilla
    - Jeff Gilbert
- Mehmet Oguz Derin
- Timo de Kort

Administration

- Plans for explainers into the specification
    - Corentin volunteers for mapping, fences, usage validation
    - Volunteers:
        - KN: I volunteer to do the ones I did. Error handling, image bitmap texture uploads.
            - DJ: timeframe?
            - KN: could do in the next week probably.
            - KN: I did “error conventions” but that doesn’t go into the spec.
        - JG: don’t remember which ones I may have done.
        - DJ: next time I’ll come up with a list of them all and we’ll assign them there.
        - DJ: assume everyone already emailed attendance at New Orleans meeting.
        - KR: reminder: folks who attend the Khronos meetings, please plan to join the IP Free Vulkan Portability meeting Tuesday afternoon of that week. WebGL is Wed, WebGPU is Thu/Fri.

- Reminder: ask your lawyers to look at the Draft Working Group Charter: https://htmlpreview.github.io/?https://github.com/grorg/admin/blob/wg-charter-draft/wg-charter.html

Conformance Test Suite

- KN: renamed it to “cts”, moved to gpuweb org. Status:
- KN: ready to write tests, but not tests with shaders yet.
- KN: however, will fix that pretty soon. Should be fine.
- KN: working on integrating in Chrome. Issues with wpt. Trying to figure out whether to put our build into wpt, or move sources into wpt. Doable, but we’d have to do cts devel in wpt.
- DJ: that’s what you’re supposed to do, but I’d prefer to not do that. wpt is huge. Annoying to have to check out everything to do your little bit. Prefer that our spec and cts are in one place, too. My pref: do dumps into wpt, either automated or every so often.
- KN: was thinking about how to automate. Now, with Francois’ help, set up CI so it pushes to gh-pages version of repo with build in it. Would also have to make it copy that into wpt and put up a PR for it.
- DJ: you have a sub-area in wpt where you have ownership. For WebGPU sub-area we could do it semi-automatically. Wouldn’t even need a PR; we’ve reviewed it once.
- KN: would be good way. Could also set up so it lives in wpt. For now will manually put a build into wpt. Just need comments on top “do not edit”.
- KN: it does work inside wpt. Works in wpt in Chrome except I’m waiting for a small test infrastructure change. The tests all run but don’t all pass in Chrome because of issue with buffer unmapping. Fail in Safari but because of things like “Transfer” being renamed to “Copy”. Otherwise, should pass them.
- MM: when you said “sources” what type of sources are there? Would expect HTML, JS.
- KN: it’s TypeScript and the build is the same files with the types stripped out.
- JG: interior types, so other things can expect certain results.
- KN: types for all the objects in the tests.
- MM: so the tests are written in TS?
- KN: yes. Pretty non-invasive.
- MM: not familiar with TS.
- KN: very much like TS. Babel pass literally knows how to parse TS and emit code without the type annotations.
- MM: we’re losing all the benefits?
- KN: it has to build the JS to run in the browser. Build has the types stripped out. We run type checks during CI / Build to make sure we don’t make mistakes.
- MM: so the types are there not so we can check we implemented the API correctly but so the tests are written correctly?
- KN: yes.
- MM: willing to go along with this and see how it works.

- KN: we could also build in JS and throw out the TS sources, don't want to do that but the output would be human-editable. If agreement we want to get rid of TS we could do that.
- MM: we'll see what the experience of writing and debugging new tests is.
- KN: spec editors should formalize how the test plan will work. When changes land, also write a line in a file saying "we have to write tests for this". Would be good for us to agree how that'll work. Right now there's almost no text in the spec, so no test plan.
- JF: any deadline for impls to meet a deadline to keep up with changes?
- KN: probably up to each browser.
- MM: when I go to this site it says "potential security vulnerability"?
- KN: only during build step. When I tried to fix it there was no fix available yet.
- MM: when we do periodic dumps into wpt are we going to be dumping the ts?
- KN: no, just the built files. If you look at the gh-pages branch now, you can look at the out/wpt directory and that's what we export.
    - https://github.com/gpuweb/cts/tree/gh-pages/out-wpt
- KN: we talked about this a long time ago and agreed that TS was the best thing to start with.

Compressed texture copies #363

https://github.com/gpuweb/gpuweb/issues/363

- YH: don't know the details actually.
- DJ: since nobody's read this let's move it to next week's agenda in the hope we have some comments by then.
- KN: did read it, but didn't comment yet.
- JG: did we have another one? I thought I commented on a compressed texture proposal.
- DJ: if you find it please add to the agenda.

Multi threading #354

- https://github.com/gpuweb/gpuweb/issues/354
- MM: no update. My last comment just reiterates what I said in last week's meeting.
- MM: either have, or not difficult to have affordances for, first 3 and last 2 items. #6 is separate issue. #4 and #5 are most relevant.
- MM: for ParallelRenderEncoder: secondary command buffers in Vk, Render Bundles in D3D. Can we modify our bundles API so it can be used in similar fashion? Then farm out bunch of command buffers to thread pool sounds natural. Upshot: problem probably isn't as hard as initially seemed, where doing these 2 things was straightforward.
- KN: I agree. Makes sense to me. 1, 2, 3 are pretty simple. More or less solved that in long comment on this thread. Parallel rendering, probably best solved with gpu bundles. Don't think they're quite as critical. Important to make sure API can solve them, but think 1, 2, 3 are the most important. 4, 5 are less critical items.
- MM: agree.

- RC: what's difference between 4 and 5?
- MM: subtle but big performance difference. #4, ability to take single pass and record pieces of it in different threads. #5, different command buffers on different threads, each command buffer has series of passes. With #4, one giant pass, all threads record into it. #5, collection of multiple passes on threads. More passes, more flushes in HW. #4 is more performant than #5.
- RC: how do people doing #4 ensure correct ordering?
- MM: in Metal you create a ParallelCommandEncoder, and from that create RenderEncoders. You reserve a slot in the output pass. All commands from one piece are replaced by its reserved slot.
- RC: thought it was a free-for-all.
- MM: semantics are as-if a bunch of RenderEncoders and wrap them up in a "meta" one. No pass breaks between.
- KN: hard to think how this would work with gpu bundle. Cross-thread communication in JS is difficult. First thought, create render bundle that's not ready yet, get both the render bundle and its encoder, and when you finish it it becomes available for the command buffer to use. Not sure it's the right way though. We can't transfer items back to the rendering thread right now. You'll end up breaking out of rAF.
- RC: for sanity, think we should have separate threads building up separate parts of the scene and transferring them.
- MM: agree, think we should do that and it's compatible with #4 and #5.
- KN: yes, but not compatible with rAF. Need to have onMessage handlers called before end of rAF. Can't get them back before you release control to the browser.
- MM: our motivation for implicit present was that it's simpler?
- KN: I'm probably going to have another proposal on that topic next week.
- MM: even if had explicit present, wouldn't solve this problem: need a bunch of callbacks that execute before rAF so you can talk back and forth between threads, then at rAF submit all previously recorded stuff.
- KN: yes, but think that would be annoying for apps. Other idea: create render bundle encoder, but get both RB and RBE together. Can transfer RBE off somewhere else, record into it, but you can use the RB whenever. Submit it into command buffer, that's OK, but actual submission of cmdbuf won't happen until all RBs have been recorded OK.
- MM: how do you know you made your deadline?
- JG: you'd get rAF later than expected.
- KN: you're not the one missing deadline. Browser should decide whether to call it for you.
- JG: you should expect rAF a frame ahead of time.
- KN: browser expects frames take equal time. In Chrome, think we call rAF later if your frame is faster. Any case, your rAF callback would finish and you've submitted your cmdbuf. Guaranteed that happens before presentation. Inside browser, would synchronously block on other threads finishing their work. Implicit join on that work finishing in the browser.
- MM: would it actually block or just not present until threads are done?

- KN: how is that different?
- MM: one holds the main thread, one doesn't.
- KN: way event loop is spec'ed, think that's the same thing. Can't run any more main thread code until present is called.
- MM: we could say WebGPU "present" is different than browser's.
- KN: we already sort of have that. One issue with swap chains: if you do WebGL work in a frame, it will show up in the same frame *unless* you're rendering into offscreen canvas, in which case it becomes decoupled from rest of page rendering. We could do that already with OffscreenCanvas.
- MM: not sure that saying we have OffscreenCanvas is the same as rendering into a cmdbuf in parallel.
- KN: I'm saying that you'll block the main thread if using a regular Canvas, but if using OffscreenCanvas it's decoupled.
- MM: if programmer doesn't use OffscreenCanvas but still want parallelism and they get it wrong (slow workers, etc.) should that hang the rest of the page?
- KN: is there any other way to do it?
- KR: depending on how you structure the JS API that may violate the HTML rendering rules.
- KN: they could just not do the submit until they get everything back from the threads.
- JG: are you talking about rendering things that haven't been resolved yet?
- KN: yes.
- JG: how about not doing that?
- MM: alternative is that all the threads communicate back and forth multiple times before doing rendering work.
- JG: sound impl of rAF is to call it N milliseconds before end of frame.
- KN: my concern isn't rAF timing, it's whether apps can even use that. You have async stuff which builds bunch of bundles. Waits for all of those *and* for rAF to fire before rendering. Lots of rendering code not tied to rAF. Probably works, not great.
- JG: simplest version would be SharedArrayBuffer views, cross-worker signaling via SABs.
- KN: each thread has its encoder, it finishes the encoder, and sends a signal back to main thread via SAB, so main thread waits until it gets all those signals before submit.
- MM: that's contrary to Rafael's desire because encoders are shared between threads.
- JG: sort of. Can structure it so composing cmdbuf in one thread and submit in another. That's the only point of cross-thread contention. Pretty simple. Two sides of that don't have contention. Would like to see better docs about Metal's parallel render encoder. Sounds like, give me N encoders, and they'll be issued in order when building this command, but can write to them in diff threads.
- MM: yes. Each thread has to declare when it's done.
- JG: that's a way not contrary to Rafael's concerns I believe.
- RC: think it's a variation of what Kai said earlier, farm out, then submit on main thread. Can submit before other threads are done. Then main threads should wait.
- JG: don't think we should do that because it violates HTML rendering rules.

- KR: Don’t see how returning from raf violates rendering guarantees today. Should be fine if wait is implicit, if you submit a cmdbuf that’s waiting for a bunch of bundles, but actually internally wait on the bundles.
- JG: But incurs presentation jank from workers.
- MM: sucks if that can happen.
- KN: Kind of same as while(true){}.
- JG: Not really because it’s too deeply hidden.
- KR: could fail validation if submitting cmdbuf with incomplete renderbundles.
- AE: similar to blocking the main thread.
- KN: you can’t Atomics.wait() on main thread.
- MM: maybe having everything done via postMessage is better.
- KN: very interesting to turn the whole rendering paradigm into async thing, but (1) people aren’t doing this now and (2) I don’t think browser perf is good enough to do rendering in async (too much overhead in switching tasks)
- MM: would love to see some measurements.
- KN: sort of fundamentally true because there’s lots the browser can do in between tasks. Also probably extra overhead.
- MM: think thesis is correct. If we’d base system on postMessage and postMessage takes half a frame that’s a no-go. Would like to see measurements.
- RC: doesn’t have to be all or nothing. We could solve the top 3 items.
- JG: I agree with that.
- KN: Yes.
- RC: the top 3 don’t need these parallel render encoding.
- KN: the top 3 are easily solved, can put up a PR.
- AE: parallel encoding can be separate from bundles. If we want bundles backed by native objects they have different representations.
- MM: if we do go with this model of using bundles to represent parallel encoding, we probably wouldn’t use Metal ParallelRenderEncoder internally.
- AE: in #5 you mentoined add’l cmdbufs are bad because you flush between things. Is that cmdbufs or also indirect cmdbufs in Metal?
- MM: it’s the pass boundary that’s the expensive part.
- JG: so you could compose render passes concurrently but not smaller granularity.
- KN: I suspect parallel render encoder is a convenient API but if we did things with bundles and internally we wrote out the bundle in a single thread then we wouldn’t see a perf hit. More of a convenience. With MetalParallelCommandEncoder you don’t have to write your own serialization code.
- MM: do we agree that we’re not going to do #4 and #5?
- KN: we should do them eventually.
- JG: what about #5.
- KN: #5 comes back to exactly the same problem.
- JG: frame timing problems become less severe if using explicit present.
- MM: OTOH you would miss frames.

- JG: you'd miss them differently. Implicit, you'd have stuttering, and explicitly, you'd not see those frames.
- DJ: should we make a decision about whether to punt #5 now?
- JG: I'd like #1, 2, 3 most. Suggest doing #5 next, then #4 maybe.
- MM: if we think a solution to #4 or #5 would dramatically change the API then we should do it earlier. But if just annotating objects as sharable / transferable then we could do it later.
- KN: depends on whether we end up posting objects back to the main thread or have some external synchronization.
- JG: think the fastest way forward is to do #1, 2, 3 now. I could put together a #5. Then if #4 is useful we can do it later.
- KN: I think if we do it the async way then #4 and #5 are pretty easy. We should do #1, 2, 3. Doing #4, 5 with Transferable is easy. Can see if that'll work at all - if async rendering is a good idea at all.
- JG: thanks.
- DJ: Kai you'll send another issue for the swapchain?
- KN: yes.
- MM: think this was pretty productive. Thanks everyone.
- KN: agree.

Agenda for next meeting

- DJ: let's leave it at that and discuss next week. Coordinate systems, bindless resources, mipmap gen, PR burndown, compressed textures.