GPU Web 2018-11-12

Chair: Corentin
Scribe: Ken!
Location: Google Meet

TL;DR

- MSAA (Issue #108)
    - The investigation is fantastic.
    - Consensus that resolving should be part of the render pass and not a standalone command.
    - Tentatively not require “hasResolveTarget” since we asked Vulkan to waive on resolve targets for render pass compatibility (for single subpasses)
    - Specify the sampleCount once for the whole render pipeline.
    - Rest of the proposal looks good.
- Push constant (issue #75)
    - Debate whether we should benchmark or rely on ISV/OHV experiences
    - We need at least one of push constants and dynamic buffer offsets
    - AI: cwallez will make a proposal for dynamic buffer offsets.
    - We should keep benchmarking push constants.
- PR Burndown
    - Accepted: Error Synchronicity Conventions/Guidelines #99
    - Accepted: s/window.webgpu/window.gpu #110
    - Rejected: Remove need for dynamic typing in WebGPUBinding #81

Tentative agenda

- MSAA (Issue #108)
- Push constant (issue #75)
- PR burndown
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
    - Justin Fan
    - Myles C. Maxfield
- Google
    - Austin Eng

    - Corentin Wallez
    - Dan Sinclair
    - James Darpinian
    - Ken Russell
- Microsoft
    - Chas Boyd
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
- Joshua Groves
- Tyler Larson

Admin

- CW: I still need to confirm with Google San Diego about hosting

MSAA (Issue #108)

- CW: Intel made a nice investigation. Most of you have read it (hopefully). What do people think? Any concerns?
- MM: The investigation is fantastic. Thanks to Intel for doing it. The general direction of attaching the resolve step to the render pass is pretty important, especially for tiling GPUs. If you do it as a separate step, then we have to load the samples back into memory.
- CW: That's the same feedback we got from the Vulkan people too. However, it seems that Intel's proposal does require a separate resolve.
- CW: In WebGPUAttachment, we'll need to specify the format?
- MM: Yes, I think it is a requirement for Vulkan, because the RenderPass needs to be provided for resolution.
- CW: My understanding is that we can only resolve to the same format, and since we know the format of the attachment..
- CW: multisample-ness is a different attribute, of texture creation. Don't think you can resolve from 4 samples to 2 samples, for example.
- RC: why do we need "hasResolveTarget"?
- CW: in current Vulkan spec, when creating RenderPass, have to create attachments for textures to be used as resolve targets. Never used by RP, but referenced by them. This texture 3 gets resolved into color attachment 2, for example. In WebGPU we only have single subpass so can simplify. Can just have bool "hasResolveTarget". That was true until today, because we raised this restriction to the Vk working group that this doesn't make sense for a single subpass. Nobody cares about that part of the spec and it can be lifted.
- MM: so should we try to lift this restriction if implementations will allow us to?

- CW: yes.
- JG / CW: (confirming exactly what the restriction is)
- CW: the restriction exists in the Vk spec today. Raised this with the Vk WG today, and they said yes, we can probably lift it. Conditions: there is a single subpass.
- JG: can't a single subpass have multiple resolve attachments? Like Depth/Stencil, and they're different attachments than the color buffer?
- MM: Intel's analysis says you can't resolve depth/stencil in Vulkan
- KR: what about multiple color attachments?
- CW: that would be lifted.
- MM: it's supported to have multiple, multisampled color attachments, and resolve them to independent textures.
- MM: are we crossing our fingers that every Android phone doesn't enforce this requirement?
- CW: not 100% sure. Seems nobody cares about this. No driver seems to use this information for renderpass compatibility.
- MM: if there's a Vk test suite and this is tested, and it says that the drivers have to do the thing we don't like, that's bad.
- CW: renderpass compatibility is generally something that's tested positively.
- JG: not a lot of negative VK tests because it crashes if you do something wrong.
- CW: we can basically just not wait for a Vulkan version.
- KR: can we do something behind the scenes to fix things up if driver behaves differently than we want?
- CW: yes. Would be expensive, have to recompile pipelines, but can be done.
- MM: so are we resolved on "hasResolveTarget"?
- JG: thought latest Intel work said, no depth/stencil resolving.
- CW: latest conventional wisdom: do this inside a render pass.
- JG: I suspect you can resolve depth/stencil in Vulkan, just not explicitly.
- CW: I reviewed Intel's investigation before they posted it, think I verified that, but worth double-checking.
- CW: so if we remove hasResolveTarget: when you create pipeline, in addition to texture formats, you give sample counts. If you want to resolve, you give the resolve target inside the WebGPUColorAttachmentDescriptor. Formats have to match, have to give MSAA render target in the right place, etc.
- MM: I think it doesn't make sense to have one attachment with sample count of 4 and another with 2.
- DM: the hardware technically can do this in multiple ways, but have to check what Vk expects.
- MM: in D3D and Metal, you tell the rasterizer, raster into X multisamples. All attachments have to match this. Think U32 samples has to be moved into WebGPURenderPipelineDescriptor. Think this has to be done for both D3D and Metal.
- CW: thought D3D had an extension or feature level where you could do mixed rasterization sampling? AMD/NVIDIA like to push this for VR.

- DM: think in D3D, sample count *and* quality level have to match. Think I agree with Myles.
- JG: there's a Vk extension allowing differing samples. VK_NV_framebuffer_mixed_samples.
- MM: NV. :)
- CW: ok, we move this outside of sample counts into RenderPipelineDescriptor.
- DM: or maybe attachment state?
- CW: is it an attachment or rasterizer state?
- DM: just a property that all attachments have to share and rasterizer has to obey. Could also remove whole WebGPUAttachment, and just use the format like we used to?
- MM: that's how D3D does it.
- CW: resolution: we keep WebGPUAttachment, rename to AttachmentDescriptor? Keep WebGPUAttachmentState.
- CW: any concern with proposed API for RenderPassDescriptor?
- RC: does API let you specify more than one WebGPURenderPassColorAttachmentDescriptor ?
- CW / MM: yes. You have one of these per attachment. Specify multisampling independently per attachment.
- MM: think this particular design works out pretty well.
- CW: devil will be in the writing of the spec to say what's allowed and disallowed, but we can look back at this investigation.
- JG: do we think we'll ever add multiple subpasses?
- CW: not from our side. You?
- JG: don't know. Seems like something that wasn't the best design, but they thought it would be useful. Other ISVs came in and said it was too hard. IHVs seem to like it. Maybe one day somebody will figure out how to use it and then everyone will want it.
- CW: nowadays people do tile-based deferred rendering which don't take advantage of multiple subpasses in Vk.
- RC: I owe DM a position on default positions. Will get back to you on that.

Push constant (issue #75)

- MM: couple new things. Gathered some metrics. 3 GPUs, 2 NVIDIA, one Intel. Don't have access to an AMD GPU running Windows. Wrote benchmark that tries to read a lot of root constants (this is a D3D example) as "A" test; "B" test was reading from a buffer. This is trying to figure out how much faster a root constant is than reading a buffer. I posted the fragment shader.
- MM: measured the GPU runtime, not CPU runtime. Got some results. The way I was doing this, I wrote the benchmark to read a bunch of root constants, then would modify a line to read from buffers. The time between those was so short the device was hotter, so the second run was slower. :(The numbers in the Github issue are wrong.

- MM: changed to interleave them. Results on the NVIDIA GPUs: no statistically significant difference between root constants and buffers. On Intel GPU: statistically significant regression to use root constants.
- MM: if you want to check this: the Github repo is open.
- MM: this is surprising. I wrote this benchmark because I thought root constants were faster to read in a shader. Someone ("zeus") says they're faster to write on the CPU side. Still no results from that measurement.
- MM: in my measurement there were tons of wavefronts. Ran ~12,000 instances. That particular benchmark satisfies one of the two numbers zeus wanted to measure.
- DM: I did request CPU measurement. Zeus agreed, saying both are important. For your case: the values will be in the GPU constant cache. Don't think your benchmark gives us one of the 2 results we want to measure. If you use the buffer your numbers will be in the GPU constant cache so we won't see the difference. The loop in the shader doesn't do us any favors.
- MM: ok.
- CW: not sure how to write a benchmark to measure what we want. Hard to mimic bottlenecks in real applications.
- MM: if we can't craft a benchmark to show these speedups then we shouldn't add this functionality.
- KR: disagree, should work with ISVs to understand how they use that because they report speedups from using this feature.
- JG: there's just pushback as having it part of this working group to benchmark test every feature that we have.
- CB: Zeus is a rendering lead at Epic working on Unreal Engine.
- RC: he's at Roblox not Epic.
- CB: I stand corrected.
- MM: the point I'm making is that we shouldn't design an API because some smart person told us to do it this way.
- CB: some NVIDIA drivers inline these as immediates. Create copy of shader in the background, swap it out when you're not looking.
- JG: parts of the API represent optimization opportunities for various IHVs. May be hard to figure out what systems they're useful on. But shouldn't undersell the ones that some smart people got in a room and decided to do a thing.
- CW: for push constants, we know this is biased toward AMD. They can promote them to scalar registers. NVIDIA would either do the caching in programs, or create fake UBO aligned with 256 because that's what their hardware likes.
- JG: these are also optimization opportunities going forward. Maybe only AMD likes these today because their hardware does it, but maybe other vendors will take advantage of it.
- MM: don't think we should design our APIs on what somebody might do in the future.
- CW: our charter says we aren't designing hardware features.
- JG: choosing not to do something is making a choice. It's cutting off opportunities down the road. I understand that you don't want to add something that's a potential not current optimization.

- DM: it's a fair request. All we can do is work with Myles to modify the test case and find a situation where it's a speedup.
- JG: we shouldn't do this on our own - should partner with ISVs.
- CW: in our current state it's hard to use ISVs as an oracle.
- MM: if we can show that there's a speedup on a significant number of customer installs, we should add it. If not, we shouldn't. Exposing a new concept to a web API that we can't remove later is not something we should do.
- JG: we can remove things from the MVP.
- CW: yes, we can.
- MM: it's easier to add than remove.
- RC: there will be content out there that do it without root constants. People using the API will have two code paths anyway. So it's a path of least resistance to not add them for now.
- CW: we really need dynamic buffer offsets in that case.
- DM: and that comes down to benchmarking again.
- CW: I can make a strong case for either push constants or dynamic offsets. Creating bindgroups is the most expensive thing you'll do all the time. Allocation of JS object, object in backend, lot of stuff that needs to happen for creating bindgroups. If we can avoid bindgroup allocation over and over again that's a win. Tiny changes between their use.
- MM: that's fine. Adding an offset to the bindgroups is not a new concept in the API the way push constants would be, and could be implemented everywere.
- CW: offsets to constant buffers and UAVs. Not 100% trivial for D3D?
- RC: should be able to do that.
- CW: so for MVP we choose to go with dynamic buffer offsets and not push constants?
- JG: why do you need dynamic buffer offsets and not immediate command buffer upload?
- CW: if you keep uploading to same area of immediate buffer you'll have stalls. Application could upload all objects, staging buffer for uniforms, put things in at 256 byte offsets. Compute space needed, put into staging uniform buffer. Bindgroup with dynamic offset -> do draw. Most efficient way to use uniform data.
- JG: OK. In some cases you can get the same effect by creating a bindgroup with the offsets you want and uploading it to different region of buffer.
- CW: there are situations where you have to create a new BG and that's expensive-ish.
- MM: CW are you OK with writing up a proposal?
- CW: yes.
- DM: concern. Push constants and dynamic offsets will fight for limited root signature size in D3D12, since we'll have to emulate dynamic offsets with push constants. Maybe push constants are more universal and have more value?
- CW: in D3D12 I think we'd put these as root descriptors, but you're right, that does eat root table space. You're right it's fighting for the same space.
- JG: if it's more universal - can build one on top of the other and not vice versa - should just implement the lower level one.

- MM: what's the general solution? If there's a concern that adding stuff to the root signature will take up space that it doesn't have to...what's the strategy?
- CW: biggest problem: not the D3D12 limit on the root table size. It's: does it fit within the constant space on this or that hardware? Can't force WebGPU to have small enough root tables to fit on every hardware out there. Too constraining.
- MM: agree.
- CW: there will be performance cliffs. Like: if you align vertex buffer to 4 bytes, it's better for this hardware. Not a big cliff.
- MM: presumably these are perf cliffs for the native APIs too.
- RC: how do you do dynamic offsets in the other APIs?
- MM: Metal: pass offset as argument. https://developer.apple.com/documentation/metal/mtlrendercommandencoder/1515829-setvertexbuffer?language=objc
- CW: in Vk: when you create DescriptorSetLayout you say this is a dynamic uniform buffer. Creating DescriptorSet: pass the offset.
- RC: D3D: root constant buffer. SetComputeRootConstantView. Buffer location, GPU virtual address. Can offset it however you like.
- CW: am sure there are alignment constraints for NVIDIA.
- CW: I'll write a proposal for this. Do we want to benchmark push constants further?
- MM: if we're arriving at consensus, I like not doing work.
- DM: I think we should continue benchmarking.
- MM: OK, happy to do it.
- KR: reach out to Roblox for more help?
- DM: we can get the info we need from what they've already given us. Instead of one giant shader, lots of invocations for lots of work that fetch the push constant and pass it through. Hide cost of writing pixels, maybe use compute rather than fragment jobs.
- CW: let's add that to next week's agenda.

Source code license

- CW: everyone agreed that 3-clause BSD + license + patents was no good
- DJ: last I heard, everyone had agreed on 3-clause BSD. That's what we would have gotten if we'd done nothing.
- JG: is there any more legwork to do before we say the repo is 3-clause BSD?
- DJ: hope not.
- JG: maybe the pull request is old.
    - https://github.com/gpuweb/gpuweb/pull/76
- CW: old pull request, I will close.

PR burndown

- https://github.com/gpuweb/gpuweb/pull/110
- CW: this is the only new proposal.

- CW: https://github.com/gpuweb/gpuweb/pull/99
    - RC: I'm OK merging this. I approved it. Can revise later if we find more things to help people with.
    - MM: no concerns.
    - CW: merged.
- CW: multi-queue one.
- JG: no proposal yet. What's the dynamic typing thing? https://github.com/gpuweb/gpuweb/pull/81
    - CW: I should get back to this. Using some types in JavaScript makes it harder for WebGPU binding, because we don't know the type of the resource inside the WebGPU binding. Makes things complicated like C/C++ from it. But I understand that we're not designing a C/C++ API.
    - JG: seems pretty easy to wrap if you want to do it at the C++ level. Instead of big union type you could do something else. My instinct is to reject this. Or defer to an issue.
    - CW: think we can close this one. Would make our life easier in Dawn, and make Dawn's API match WebGPU better, but can list this as one of the differences between Dawn and WebGPU.

Agenda for next meeting

- Push constants
- Dynamic buffer offsets
- JG: barrier proposal for multi-queue
- MM: I'd like to hear from people if they're starting implementation, what they're thinking about. Want to know story for other engines.
    - DM: is there anything in particular you're interested in, or general status?
    - MM: just general status. Last I heard, Moz was going to do their own thing.
    - DM: we'll share next time.