GPU Web 2019-03-11

Chair: Dean Jackson
Scribe: Ken
Location: Google Meet

TL;DR

- Updates
    - Apple: Uploading images to textures, they work.
        - Made a patch to remove WebMetal.
    - Google: Busy connecting Blink to Dawn and implementing JS API.
    - Microsoft: Contributions to Dawn: Buffer destroy, etc.
    - Mozilla: wgpu 0.2 release! Blog post! https://gfx-rs.github.io/2019/03/06/wgpu.html
- Error handling #238
    - Scoped error capture.
    - RC: Just still concerned about debugging - async doesn't work for that.
        - KN: Not intending to solve debugging. Dev Tools should do this with "pause on WebGPU error". We need to ship with that on day one.
        - ai: KN: Clarify usage of the "createReadyRenderPipeline" example in the doc (you can't use it inside rAF)
    - Tests will be written fully async unless they use rAF.
    - Merged PR.
- GLSL on the Web #219
    - Mozilla/Google want SPIRV ingestion + GLSL-to-SPIRV + (W)HLSL-to-SPIRV.
        - As a data point, shaderc wasm was 500k (in 2017).
    - Apple/Microsoft want (W)HLSL ingestion + GLSL-to-(W)HLSL.
    - ai: JG: Investigate/document how to do GLSL with WebGPU. I.e. binding model changes and stuff.
- getSubData/setSubData #213
    - "Symmetry" between get/set isn't so important. Upload/download isn't symmetric anyway.
    - MM: More concern with ease of use of the API. Easy to upload but hard to download is bad. [KN: Esp for compute]
    - JG/DM: createWithData instead of createBufferMapped?
    - DM: If setSubData is for ease of use and not max perf, a shim should be ok.
        - Discussion: A well-made JS shim for setSubData actually should be able to be about as optimal as the browser could be.
        - JG: If we do add setSubData then we'll end up having to optimize it because apps will depend on it.

    - KN: Shims like this can be part of a tutorial and that doesn't hurt beginners at all. They're going to be starting with a tutorial anyway.

Tentative agenda

- Error handling #238
- Case for existing GLSL on the Web #219
- getSubData / setSubData #213
- PR burndown
- Snapshot tracker burndown
- Agenda for next meeting

Attendance

WIP, it is the list of all the people invited to the meeting. **In bold the people that have been seen in the meeting:**

- Apple
    - **Dean Jackson**
    - Jason Aftosmis
    - JF Bastien
    - **Justin Fan**
    - **Myles C. Maxfield**
    - Robin Morrisset
    - Theresa O'Connor
    - Thomas Denney
- Google
    - **Austin Eng**
    - **Corentin Wallez**
    - **Dan Sinclair**
    - David Neto
    - **James Darpinian**
    - John Kessenich
    - **Kai Ninomiya**
    - **Ken Russell**
    - **Shrek Shao**
    - Victor Miura
    - Zhenyao Mo
- Intel
    - Aleksandar Stojilijkovic
    - Bryan Bernhart
    - Yunchao He
- Microsoft
    - **Chas Boyd**

    - **Jungkee Song**
    - **Rafael Cintron**
- Mozilla
    - **Dzmitry Malyshau**
    - **Jeff Gilbert**
- Yandex
    - Kirill Dmitrenko
- Elviss Strazdiņš
- Joshua Groves
- Markus Siglreithmaier
- Mateusz Kielan
- Mehmet Oguz Derin
- Samuel Williams
- Timo de Kort
- Tyler Larson

What's new in WebGPU Update

- Apple
    - JF: have texture resources working. Uncompressed image -> 2D canvas, then use that to convert to ArrayBuffer for upload to WebGPU.
        - Next couple weeks: updating impl to match changes in API from last few weeks, like SwapChain model, new context model, etc.
- Google
    - AE: in the process of connecting Blink to Dawn and getting that through the command buffer. Hopefully will get a lot of this work done so that it'll be quick to fill in the rest of the API, soon.
- Microsoft
    - RC: contributing changes to Dawn. Contributed Buffer destroy. Reviewing issues and PRs and giving feedback.
- Mozilla
    - DM: released v0.2 of native impl. Made quite a bit of noise. People confused all over the place of what WebGPU is. Haven't done much Gecko integration.
    - https://gfx-rs.github.io/2019/03/06/wgpu.html
        - MM: we've made a patch to remove the old WebGPU (WebMetal) impl from WebKit. Don't want to land it until we have compute support in our new WebGPU
    - DJ: please link to blog post in the Minutes. Thanks.

Error handling #238

- KN: this is the one which does ErrorScope stack, push/pop. Suggested by CW on last pull request. Think it resolves weirdnesses / issues with the last one. Any input?
- JG: I like it. Posted on the PR. Firefox, for our WebGL impl, uses something similar to this; LocalErrorScope. It’s not async but we use it to ensure we don’t generate errors in unexpected places.
- KN: we have similar things in Chrome too. Scoped things that keep errors within their lifetime.
- RC: think it’s fine but the main issue we need to address is to let the developer throw an exception synchronously so you can find where the error is. You find out on a callback much later what happened.
- KN: the async error handling is not intended for debugging. Will not tell you where the error happened. Only giving you this message. Not intended for this use case. Hoping that a DevTools option for break-on-error will be enough to solve that problem. It’s not 100% clear to me. Not visible to code inside the app. Can’t print an error stack, put arbitrary handler on there. But you can get devtools to pause there. As long as you’re not super set on using printf-debugging, the DevTools thing should work. But not sure it’s necessarily enough. One idea would be to have browser flag that gives you synchronous PopErrorScope. Wouldn’t be able to use in apps, because we don’t want them to do it synchronously - it’ll kill their performance.
- KN: think that DevTools option is something we’ll want on day one.
- RC: if we do have a sync DevTools option then what’s the push/pop error scope for?
- KN: testing. Detecting out-of-memory cases and recovering from them. Waiting for a render pipeline to finish compiling. Still need some mechanism for these. Think this works reasonably well even independently of debugging and testing.
- RC: today, aren’t RenderPipelines already promise-based?
- KN: I added the createReadyRenderPipeline recently, while working on error handling stuff. But not necessary to do it in the IDL with this new error reporting mechanism, and I took it out.
- JG: I like that.
- RC: as long as it’s used for writing tests that’s OK. In the docs we should say something so we don’t await a ready render pipeline.
- KN: we can have docs / examples. Async pipeline creation would be useful if you have a fallback or are willing to skip drawing, so you can avoid rendering frames while waiting for pipeline to compile.
- RC: will tests be written with awaits?
- KN: my intent with the tests was to make everything async. At least things not involving canvas. But we’re probably not as tied to canvas as we are wtih WebGL. Have to make sure right submits get done in a frame, in a test with multiple submits.
- JG: think it’s useful for most tests to use framebuffers and not go all the way to the screen.

- KN: will have some canvas tests but most of our functionality tests won't require canvas.
- DJ: do you want to update anything or should we approve this now?
- KN: I don't think I have anything needing updating now. Can make updates later.
- RC: the only change I think we need is to update sample code.
- KN: OK, let me do that before we merge it.
- JG: seems like something that client code will sometimes do.
- KN: I didn't give an example of createReadyRenderPipeline. Could maybe use an example. But maybe it doesn't need to be in the PR.
- KN: happy to merge it now and send something to be updated to RC.
- RC: OK.
- DJ: great. Done.

Case for existing GLSL on the Web #219

- DM: tried to write some GLSL shaders targeting WebGPU converting to SPIR-V. Fairly useable with extension that lets us specify binding points and locations. If we can come up with a good story for using existing shader bases targeting WebGL without rewriting everything would be good. Had vague idea about taking Vk extension for GLSL and create similar extension for W3C, and a WASM library on the web for letting users compile GLSL to Web-SPIRV. My bug had a question for Apple, how do you see interop between WHLSL and GLSL, if it's possible, and what the roadmap is?
- JG: would be great if it just worked as a source language.
- DJ: when you say "just works" do you mean as an extension?
- JG: a JS polyfill library that makes this just work.
- DJ: we would even be happier if it just worked without a polyfill. We like languages that people can type in.
- MM: three different options.
    - 1. Do nothing. GLSL is dead.
    - 1b. Help web developers add compilation, then they can compile to whatever WebGPU target.
    - 2. Add GLSL support to the core WebGPU API.
    - 3. Add it as an extension.
- JG: the first is superfluous. People will make it work. You can't *not* make it work. The options are along the axis of how much we want to help. Should the browser ingest it in the core API? As an extension? Make it viable to have a library that converts it to whatever language we ingest, but outside the API?
- MM: one of the things we've learned from having many languages in browsers is that having multiple languages ingested by the core API that purport to achieve the same goal is tenuous to attain.
- JG: don't think putting it as an extension is viable. Why would an app use an extension for one? Is this accepted by the core of the spec? Or should it be a compile source, and WebGPU is the compile target?

- DJ: part 2 is possible. Could put extra params on it. Possible, with 1 MB download of WASM binary? Or building on other parts of the browser, where in Chrome you can expose an API with a wasm fallback?
- RM: do we want to encourage the developers to compile on their machine and ship the compiled code, or ship a library to do the compilation on the client?
- JG: I think it's developer choice. Do you see that affecting the choice we make here? Developers will always have the choice to pre-bake their language or not. ShaderToy will need to ship a compiler. Fruit Ninja can precompile.
- RM: if we expect more developers to compile on their machine then the size of the compiler is less of a constraint.
- JG: compiler size is one of the unexplored areas of shader compilation to my knowledge. We don't know how small we can make it. Don't know if we even know the bounding size on it.
- DJ: Daniel's here - were you aiming for a 100 KB binary?
- DS: we can cut out a lot of what a compiler might need. Since we compile to SPIR-V we can just not do any validation. Our compiler's a lot smaller because we parse and generate SPIR-V without validating anything. Will make things simpler if SPIR-V is the ingestion language. Hard to make things small. JS seems the good option. It's hard for me to learn Rust. In C++ you have to be careful to not pull in ostream for example. Kai looked at wasm'ing GLSLang. 500 KB?
- KN: yes. Compiled with size optimizing, got 500 KB for shaderc.
- JG: exciting. Was expecting it to be 5 MB, which would suck. But would be possible for some uses.
- KN: think we can do better. If we took glslang rather than shaderc think we could get it down a little bit.
- JG: exciting. Makes my day brighter. :) DM, sounds to me like what we should do with binding locations for WHLSL. Legacy version of GLSL doesn't include them. Here you're showing that it's pretty easy to port them to the new versions of GLSL. Talked about WHLSL a little bit, how it could be polyfilled. Do you think it'd be possible to take a legacy GLSL shader without these binding annotations and polyfill them?
- DM: sure. Could have dictionary mapping sampler, texture, etc. names to bind groups + offsets, if you don't want to modify the shader source. WHLSL already has the register assignments, just not the ones we need, and here we don't have any.
- JG: could the compiler do this, automatically assign the binding groups, then have app introspect on where they ended up?
- DM: it doesn't work like that. The app's supposed to use the PipelineLayout between shaders. Automatic assignment of bindgroups won't work because it doesn't know the frequency of updates of various variables. If we go this way for WHLSL you have a good point that we should go the same way for GLSL compat.
- MM: there is a discussion: if we want to change the flavor of GLSL, etc. More broad question: this group seems to be saying, no native ingestion, but include a compiler in their web app?
- DM: I don't think ingesting multiple formats natively is good.

- MM: Why not just ingest a single format of GLSL?
- JG: the same way SPIR-V offers a better ingestion format than WHLSL, it also offers a better ingestion format than GLSL. I think we should offer only one ingestion language and that it should be the compile target. Think it's useful to talk about how hard it is to do that: for example, dealing with legacy GLSL and HLSL, bind group locations, etc. It's been my opinion for a while that we should ingest SPIR-V and that if you don't have SPIR-V on hand you should download and cache the compiler. This is not a new position.
- DM: it's just another conformation that there are multiple languages people want to write in. Finding lowest common denominator, want something not opinionated on syntax, etc., and that's SPIR-V.
- KN: I agree with Dzmitry.
- RC: I agree the API should ingest one shading language. My pref would be HLSL or WHLSL. Agree with JG, people will make it work. If they run into calamity then we could have multiple ones, but should have one for MVP and see where it takes us.
- CB: thanks RC.
- MM: nobody thinks the one true language should be GLSL, and so it sounds like we don't want an extension.
- DJ: wasm module that compiles flavor of GLSL to the ingestion format, SPIR-V or WHLSL?
- MM: if we agree on that then should we do any work to define a flavor of GLSL that will work well with WebGPU, and/or the compiler Dean just described?
- KN: would be in the best interest of current WebGL developers, and therefore our interest, to provide this or find someone to do so ASAP. Think flavor will be similar to what Vulkan uses. Have one that's good for SPIR-V, one pretty good to HLSL. These compilers exist. Think we need to take the responsibility to make sure these work well for the web environment.
- DJ: it would be nice if someone could document WebGPU-GLSL, a separate node in our spec (or a separate spec).
- JG: how about an investigation written down for it.
- KN: we won't need to make it formal; unless we write lots of it, it can be an impl detail. If it behaves like shaderc/glslang, nobody will have problems.
- DJ: the reason I think it's worth writing down is that if another compiler has to convert it to WHLSL, don't want it to be a different variant. Should have the tutorial of what you can do.
- JG: regardless of what we ingest, will be good to have docs of the deviations from spec'ed GLSL. It's something we want from an ecosystem standpoint.
- DJ: thank you for volunteering.
- MM: think it's probably not this group's responsibility to write a compiler. Writing an investigation sounds like a good idea since we are the experts on WebGPU. Probably shouldn't be trying to spec the flavor of GLSL. Maybe another WG?
- JG: having a compiler might not be our responsibility but it would be greatly mission-aligned with this WG.

- MM: think we're agreeing with each other.
- DJ: KN could you link to your GLSLang investigation?
- KN: will do.
    - https://lists.w3.org/Archives/Public/public-gpu/2017Oct/0013.html
    - https://lists.w3.org/Archives/Public/public-gpu/2017Oct/0022.html Live demo
- DS: glslang and shaderc are the same thing. Different API to the same thing. Different backends.
- DM: you mentioned there's a GLSL to HLSL path?
- KN: would go through SPIR-V today, currently. There are other compilers for GLSL to HLSL but don't think they'll be that robust.
- JG: ANGLE has one.
- DS: we're currently putting the glue into shaderc to make it talk to spirv-cross. Using spirv-tools binary parser and using it to parse the binaries. Replacing their parser with our parser and then calling into spirv-cross.
- JG: does anyone want to do the investigation
- DM: I'll do it.

getSubData / setSubData #213

- MM: we've got setSubData as a one-shot upload that doesn't have to worry about buffer timeline management. Cool for symmetry to have getSubData that's the same idea. Reason for this PR: we should have both or neither.
- JG: I agree with that.
- KN: I don't see it as that important for them to match. Nice symmetry in having both, but for one thing: the two dimensions are not symmetric. If you're trying to write a WebGPU app from scratch, setSubData is generally more useful. Less true for compute, where you need data back as well. Don't think setSubData is that useful. Not convinced that having that extra API is worth preventing developers from having to write 5 lines of extra code.
- MM: guess it's important that WebGPU be easily writable by new developers. If you need to write lots of code to download data but not much to upload, isn't friendly.
- JG: crazy idea. Agree that uploading data to GPU is something you immediately want. Does it make more sense to expose this asymmetry better if we allow providing with buffer creation an initial set of values, and then remove setSubData, to capture 80% of the use case? Updating buffer in place would require mapping, etc.
- DM: does that deprecate createBufferMapped?
- JG: maybe.
- DM: then I like it. :)
- KN: you're suggesting something sort of like createBufferMapped but instead of once, it's N times, and then you disable it?
- JG: I'm proposing things like texture creation APIs from D3D11 and OpenGL where you can create the texture with formats, etc. and undefined data, or you can provide data for it. Don't have to map/write/unmap. Lots of places where you have the data and it's not

going to change. Exposes better the asymmetry too. Uploading, you often just put data on the GPU and never touch it again. Reading it back will always have to deal with the realities of how async the GPU is.
- KN: when you do the readback your data will never be available in the way you can pretend it is with setSubData. What's the functional difference between this and createBufferMapped?
- AE: doesn't this let you reduce a copy where it can come back already mapped?
- KN: yes. Could be zero copy in Safari. Could be shared memory in Chrome.
- DM: setSubData was supposed to be for easy use cases, not reaching max perf. Also, doesn't Chrome have the ability to batch setSubData, not polyfilling over createBufferMapped?
- KN: lifetime of data in setSubData is trivial and can be done in ring buffer rather than some more complex data structure. Depending on how quick-and-dirty we want it, it might not be necessary to have it or the API, but not sold on it.
- MM: for a new developer it's important.
- JG: it's not unfair for new developers to have to deal with this and pipeline creation.
- KN: I think this is useful for new developers. Could provide the 5-line functions for get/setSubData. Isn't possible to pick up the new API without tutorials, and easy to provide them.
- JG: if we add setSubData we're going to be continuously optimizing it and making sure it goes fast.
- MM: that's good.
- JG: that's not good. They should just do the right thing in their app instead of the quick-and-dirty thing.
- MM: we're turning into a philosophical discussion. Having the ability to improve the perf of every WebGPU app would be great.
- JG: they should do the right thing from the beginning. How easy should we make it for developers? Think setSubData and getSubData don't meaningfully improve how easy it is to use the API and so we shouldn't have them.
- KN: would like to wait for CW to come back in order to make a decision.
- DJ: let's table this until next week.

PR burndown

- https://github.com/gpuweb/gpuweb/pulls
-
-
-

Snapshot tracker burndown

- https://docs.google.com/spreadsheets/d/1cw0R4fjJddCQZvmgIOxDMK2YieMSaK0yGYof01K1OXM/edit#gid=0

-
-
-

Agenda for next meeting

- Vertex shader only? #240
- Default values #241
- KN: have people had a chance to look at the tentative F2F dates in May? (13-17)
- DJ: May's fine, but the particular times in September might be difficult.
- MM: September will almost surely be impossible for me.
- DJ: we should decide who's going to host. Google/Apple/Mozilla.
- MM: we've been having a lot of F2F's in the US. Want to give non-US people a chance to say whether that's OK.
- DJ: a lot of issues from David Neto about programming model.
  - DS: he probably won't be on the call next week.
  - MM raised a question about running the vertex shader only.
  - DJ: I see about 10 PRs approved with small changes. Should land them to get them out of the queue.
- <after the meeting>
- JG: https://github.com/gpuweb/gpuweb/issues/171
-
-