GPU Web 2018-01-17

Chair: Corentin
Scribe: Ken
Location: Google Hangout

Minutes from last meeting

TL;DR

- Next F2F will be in Montreal around the end of April as the WebGL WG will meet there.
- Discussions on the shape of the JS API
    - There's a standard shape of JS API for the Web but no written down guideline. Anne van Kesteren and Cameron McCormack would know about it and WebIDL
    - Explicit graphics APIs tend to have "descriptors" that gather argument for object creation. These descriptors are where a bunch of the extensibility happens.
    - Potential API shapes discussed:
        - Builder pattern for descriptors
        - Property bags for descriptors
        - Pre-created and closed javascript object for descriptors
    - There's a desire to make things type safe, especially in the with extensions.
    - We want WASM to be first class but there is not precedent. The WASM CG mentioned a solutions for property bags but doesn't have something concrete yet
    - The JS API could later be transformed into a shim over the WASM bindings
    - Consensus that we don't want to innovate on the command buffer building and keep function calls for each operation (vs. a spec for the wire format)
    - Agreement that the best way to make the JS API (if there is no constraint from WASM) is property bags, ignore unknown keys and have devtools + debug contexts for unknown keys.

Tentative agenda

- Shape of the API
- Synchronous APIs and upload / downloads
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson

        - Myles C. Maxfield
        - Theresa O'Connor
- Google
    - Corentin Wallez
    - Kai Ninomiya
    - Ken Russell
- Microsoft
    - Ben Constable
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
    - Markus Siglreithmaier
- Yandex
    - Kirill Dmitrenko
- Elviss Strazdiņš

Admin items

- CW: Asked about the CLA: lawyers thought it was agreed upon.
- CW: would be nice to do another F2F
    - OK to do it in Montreal next to Khronos F2F at end of April?
    - Dean would prefer if it were before the Khronos F2F (i.e., a day before rather than a day after)
    - Myles & Dean are busy the week of April 9
    - We can schedule the WebGL F2F so that we can get the WebGPU F2F co-located well.
- CW: Would be great to get a strawman draft of the API to iterate on it
    - Getting the shape of the API in place would be good
    - Also synchronous APIs

Shape of the API

- CW: what are the constraints of the shape of JS APIs? (Asking Dean, Jeff etc.)
- JG: no constraint. Pretty wide open
- MM: should we have a completely different shape for WebAssembly and JS?
- DJ: from the JS API side: while there's no formal def'n of JS APIs, there are pretty well adopted conventions between the WHATWG and W3C specs.
    - If we deviate too much from them we'd get pushback from those communities, though of course we can push back on the push back.
    - There's a general shape of the APIs though it's probably not written down anywhere.

- CW: is the shape of this API something easy to transcribe to WASM?
- DJ: more concrete examples:
    - Web APIs nowadays expose things with constructor calls that take dictionaries of options
    - Not the type of API that NXT exposes (builder pattern)
        - Not saying anything's right or wrong, just not the same
    - Dean can dig this up
    - new WebGPURenderQueue({ key1: foo1, key2: foo2 })
- CW: in NXT, using builder pattern because it was easier to autogenerate code for. Not necessarily pushing it.
- DJ: there are autogenerators for taking dictionaries in constructors
- JG: the release of Vulkan / D3D12: tended toward structures that you constructed and passed to function calls. Maps pretty well to passing dictionaries into functions. That's my initial preference because it lets you reuse them without creating extra garbage.
- CW: Metal also has this esp. for texture creation. One thing: the only API that handles extensibility is Vulkan. Others have new versions of constructors (e.g. CreateDevice1). We'll probably have extensibility along multiple dimensions. Would vote against the Vulkan way (chain structures together on the stack). Easy to do in JS. What do you do at the C++ level? extensions make things messy.
- JG: in WebIDL you can have partial dictionaries. Availability at compile time of WebIDL. Seems hard to do at runtime.
- RC: one problem with dictionaries: since it's string/value pairs, hard to know whether adding new things to the dictionary actually took effect.
- KR: When we use dictionnaries in JS it is easy to make a typo. This problem has happened in many places for WebGL. It is the style of JS engine but Flooh (developing a WASM engine etc.) is using the builder pattern and it looks quite nice in C++. Gets stronger typing.
- CW: the builder pattern could give extensibility. Extension exposes either new enum value that can be passed, or a new function that applies to the dictionary. If you use the new function when the extension isn't present you'd get an error. Doesn't look much like other web APIs though.
- DM: can't make a typo with the builder pattern b/c you'd have a runtime error, but if you have a typo in a field you'd have the same error?
- KR/CW: no, because of WebIDL dictionary conversion rules. If you have a typo in a key in a dictionary then it's ignored
- MM: there are two schools of thought, unknown keys being no error, and unknown keys being an error.
- CW: is there any precedent for this in web APIs?
- KN: don't know of one. It's a bindings-level decision. Or special-case it after the bindings level.
- KR: Basically we would be bound by the WebIDL rules or extend the dictionnary WebIDL rules. Pretty sure the conversion process from JS object to WebIDL dictionnary drops unknown keys.

- TOC: WebIDL has concept of partial typed dictionaries (..? didn't get all)
- JG: key/value bags that are non-extensible? No new keys?
- CW: how do you handle extensions then? Have to enable them at device creation
- JG: extensions that aren't compile-time: would still have those on the key-value bags, but they'd be ignored. Fixes the typo issue.
- CW: so would have "new WebGPUTextureDesc", set a bunch of stuff on it, then pass it in to create a texture?
- KN: don't know of any precedent for it, but don't know of situations were you aren't allowed to use expando properties on JS objects in WebIDL
- CW: seems we have to do a bunch of research and figuring out precedents
    - Somebody we can talk to who knows about all of this?
- KR: suggest annevk@ as one person.
- DJ: also Cameron McCormack, WebIDL author.
- MM: still trying to understand goal: are we trying to make an API where unknown keys are errors but aren't errors for extensibility / extensions?
- CW: constraint #1: create descriptions / pass whole bunch of arguments and have other arguments based on extension availability. #2: safer: get an error if you set an unknown key or attribute.
- RC: personal vote is to make things as type safe as possible. Linters, code completion, etc. To play devil's advocate for the builder pattern: if you have a PipelineDescriptor and a builder that has type safe methods, it's obvious which one you have to make first (desc vs. Pipeline). As for extensions: WebGL gives you back an object that has methods and fields only valid when that extension's in use, which is good. But we should use regular JS feature detection for everything that's not HW specific. Asking for the string of the adapter shouldn't be an extension for example.
- BC: versioning in D3D: the function Foo becomes Foo1 with a Desc1 instead of a Desc (or 2, or 5…). The dictionary works well here because they can have the extra thing for the "1" version of the API. Don't know whether we want code to be forward and backward compatible. Agree on the type safety points. If you pass something in, should barf really early to avoid lots of error checking later. Maybe if you pass in a version token when you create the context, tells you what's valid in the context and the property bags.
- KN: think that only applies to API revisions, and not things like HW extensions.
- BC: in D3D these are sometimes the same thing.
- CW: D3D's somewhat of a closed system. MSFT dictates what goes where. Hard for vendors to make their own extensions. Sure that everyone is going to want their own extensions here. (Similar to how OpenGL has)
- BC: so, if you have exts like WebGL, is enabling an extension making dictionary value pairs valid that would be invalid if you didn't turn it on?
    - Descriptor of texture, and one property is something only provided by an extension, so becomes an error condition if ext is not turned on
- JG: hard to enforce; can't remove things from dictionary. Main concern is one that Ken voiced: can easily typo dictionary keys. JS will treat it as undefined. If we fix the typo-ing

issue and have only valid things we support, think we're good. Don't think we have to have an error if you set anisotropic filtering and the system doesn't support it.

- KN: if we have checking saying "this is the set of valid tokens", we can have checking for all browsers. But say you have an ext that adds a new key. You'll have to try with the key and then remove it?
- JG: you'd just do feature detection somewhere else.
- KN: ok, agree
- JG: at browser level: "CreateTextureDescriptor". Browser might support devices that have anisotropic filtering, but current one doesn't. In this case we could either accept and ignore it, or produce an error if it's the non-default value...doesn't work that well with trying to eliminate typos.
- JD: if you have two browsers, one supports an ext and one doesn't, it's always going to be an error in the browser that doesn't support it. Think it's better to have browser that supports it throw an error if extension isn't enabled.
- KN: think we can't use IDL to describe this.
- JG: why?
- KN: makes sense that you throw the error when you receive the dictionary rather than when you set an unknown attribute.
- JG: proposal: pass in a normal dictionary, or ask the API to give you a proper, non-extensible dictionary. Will prevent typos. Or do it yourself in a linter library.
- KN: agree with this; will the object have a different set of allowed keys depending on which extensions are available?
- JG: not if it's device-independent. But if you say "create it for this device", it could be. Keeping people from setting keys they shouldn't use with this device. Better than supporting plain dictionaries with random keys.
    - Creating a non-extensible key/value dictionary for these different types lets us sidestep this.
- KR: think we're already heavily relying on JS language features even for this concept.
- KN: one idea: start with a C++ API and then have a shim out to JS. A JS API that looks like a C++ API, then nicer JS bindings on top. Means that WebAssembly has something to bind into. Can do anything at the JS level on top of it.
- CW: had one discussion with the WASM group.
- JG: no precedent for an API that has a different WASM vs. JS API.
- KN: some web APIs are already difficult to expose to WASM because they use concepts that don't translate well to C++.
- JG: we shouldn't define our API so that it's easy to consume for WASM. Instead design it so it works, then figure out what primitives are needed to bind well to WASM.
- DM: extensions can have a combinatorial explosion of structures.
- DM: what about having pNext pointer? Pass it along?
- JG: that structure is a C++ way of looking at it, not necessarily preserved in the WASM bindings.
- DM: if there are a finite number of things you can put in the pNext field, there would be a finite number of things you put it there

- JG: one idea is that WASM would produce structs, then calling out would produce property bags. Don't like the pNext field from the API design perspective.
- DM: not a problem for usability.
- MM: there's still a long time before WASM is going to be able to make any API, so how about making a really great JS API, and then figure out later whether we should figure out to bind it down to WASM, or create a new WASM API.
- DJ: if it turns out that we have to make a new API for WASM, we could do what Kai suggested, and go to a lower level later.
- CW: would still need to figure out a way to handle extensibility and the typo problem. Solve the WASM problem later? Do trivial binding to Emscripten later.
- TOC: why is the typo problem uniquely bad to this API?
- CW: think because we expect more extensions than APIs
- KN: also this API is going to be a lot bigger than most APIs
- JG: solving a problem that we've repeatedly run into and that isn't solved by the surrounding environment.
- KN: extensions do make the linter's job more difficult.
- JG: not concerned about that
- KD: there aren't linters that handle extensions
- MM: The proposal is you use dict, put whatever you want in it, pass it to a function that can decide to ignore unknown keys, or we set a policy to throw
- KN: could also just throw these errors during development and not production
- MM: pretty good point. If we want to throw when you have the WebInspector open, couldn't define that with WebIDL, but each browser would have to figure out how to do this.
- RC: has to behave the same whether the tools are open or not. If we throw an exception, what does it contain? A dictionary containing all the unsupported keys? Don't want to have to parse strings
- JG: instinct: creating a debug version of the device? Throw warnings, etc. for things it doesn't understand or extensions not enabled.
- RC: what if your HW doesn't support instancing. Do you run the debug WebGPU device on all machines?
- JG: conflating feature detection and validation. Should be possible to feature detect things in non-debug (Release) devices. Should figure out what to do for feature detection. Don't think we should do feature detection by having a function reject keys it doesn't understand.
- RC: so how to do feature detection?
- JG: the WebGL way? or the way Vulkan does it for built-in features. Giant dictionary of booleans of what's supported.
- CW: have to pass that struct back in, in Vulkan. Say what you're enabling.
- JD: so feature detection happens when you create the device, and later, do the right thing only for features that are enabled?
- CW: yes, get validation errors if you try to do things you shouldn't.
- MM: sounds great

- MM: little misleading to say it's the builder pattern. Should we make a declarative form for every rendering command? No.
- CW: don't want to make a JS array and put draw calls in it, then pass it to the command buffer.
- JG: interesting idea: why not just provide a formal spec for the command buffer wire format?
- MM: some situations where we might want to innovate: don't think this is one of them.
- CW: agree
- KN: what it gets you: in WebAssembly, you don't have to exit the VM to encode a command buffer.
- CW: but you already get that if you have a native WASM API. WASM compiler can inline the functions, etc.
- JG: no, we have to call into the driver.
- KN: can call into the driver later.
- JG: but if all we're doing is writing this format, then why not just expose the raw byte format?
- CW: because we don't want to spec it and promise it's stable.
- CW: agree: not going to innovate on way of creating command buffers. Builder pattern. Agree for now that we make a great JS API now, then a great WASM API later. Need to push WASM group.
- DM: don't agree on that -- we have to think about it now. Property bags can be inherently slow. Is there are evidence that they're not slow?
- JG: could be OK for preallocated property bags where you know all the keys
- KN: sure, you can reuse them, but if you've already diverged from the way Web APIs work then why continue to use property bags as the concept?
- JG: for near-real-time APIs can suggest that people use certain patterns
- CW: don't have consensus on making the JS API first. But agree that the best way to do the JS API (if no constraint from WASM) is dictionaries, property bags, ignore unknown keys, some developer tools that warn about unknown stuff.
- JD: debug context that warns about things.
- CW: think this could be a library, but doesn't have to be.

Agenda for next meeting

- MM: in the last call we talked about how an HLSL variant will be the language we want people to write. This isn't spec'ed yet. But at some point this group needs to make a variant of HLSL.
- CW: Ben is looking at a small prototype of the HLSL spec and tests?
    - BC: yes, it's on Github now
- Synchronous API and upload / download
    - Can JG or KR have a recommendation from the WebGL WG's recent discussions?
    - JG: can do.

- And more topics if you want to add.