

GPU Web 2019-03-25

Chair: Corentin

Scribe: Ken

Location: Google Meet

TL;DR

- WebGPU F2F tentatively at Mozilla Bay Area office May 15-16th
- Dean put up a first version of the bikeshed spec. Could have an automated process to publish it on a Web site. (webgpu.io?)
- Vertex input refactor to int-keyed objects [#186](#)
 - Consensus to do sparse arrays for the vertex input descriptor
 - Still not consensus on setVertexBuffers
- provide getSubData as sibling to setSubData [#213](#)
 - Discussion whether usability for beginner is a goal for WebGPU.
 - Concern that performance of setSubData() and getSubData() will be non-portable
 - Will need more discussion
- Agreed to merge:
 - Remove Anisotropic texture filtering [#202](#)
 - depth-stencil texture formats [#211](#)
 - move Window.GPU to Navigator.GPU [#217](#)
 - Remove clamp-to-border-color option from GPUSampler [#235](#)

Tentative agenda

- The Great PR Burndown
- Agenda for next meeting

Attendance

- Apple
 - Dean Jackson
 - Justin Fan
 - Myles C. Maxfield
- Google
 - Austin Eng
 - Corentin Wallez
 - Dan Sinclair
 - James Darpinian
 - Kai Ninomiya

- Ken Russell
 - Shrek Shao
- Microsoft
 - Chas Boyd
 - Rafael Cintron
- Mozilla
 - Dzmitry Malyshev
 - Jeff Gilbert
- Mehmet Oguz Derin

Administrative items

- CW: no concern with having a F2F between May 13 and 17.
 - Tue-Wed, Wed-Thu?
 - DM: like having the idea of having it for more than one day, but no preference on which day it is.
 - KN: slight preference for Wed-Thu.
 - JG: Tentatively Mozilla will host
- KR: How about a WebGL F2F on Thu-Tue depending on what day is free?
- DJ: WebGL should be first, Tuesday should be good
- KR: Will send an email to the ML.

The Great PR Burndown

- CW: Dean has a Bikeshed version of the spec. [#248](#)
- DJ: hope it's a 100% accurate copy of the Web IDL into the spec. Deleted the sketch IDL, referred people to spec. Checked in generated HTML. Assume you'll be able to view spec from Github. There's no actual text in the spec except for the IDL, so could start merging in the notes, descriptions, etc.
- CW: did you do this from the latest sketch IDL?
- DJ: yes. From yesterday.
- CW: suggest we merge this. Then every PR today we should rebase on top of it.
- DM: why would we not have the Web IDL?
- DJ: spec programmatically generates IDL in the appendix. Merges all the bits together. We'll have to submit some pull requests to Bikeshed to give template for our spec. Web IDL parser also barfs on some of our default values / parameters. (Doesn't handle does in default values, for example.)
 - KN: I don't think our IDL is valid. Should you be able to access members of other interfaces? Bitfields are not much of a concept in the HTML spec, so not surprising that it's not supported yet in Web IDL.
 - KR: The WebGL repo has a WebIDL extractor script since it is still in HTML. Could use it for this spec as well?
 - DJ: yes, good idea. Will look at that.

- DM: since HTML spec is product of the other files do we need it in the repo?
 - DJ: not really, but want an online version that's up to date. Need either somewhere to serve that or something that auto processes the Bikeshed for us. Could do some CI to do it?
 - KN: when you submit patch can add a CI step that pulls, builds, and pushes to Github pages.
 - DJ: good idea. Will make it a separate repo?
 - CW: we have webgpu.io.
 - DJ: will try that.
 - CW: do we merge the Bikeshed without the HTML, or wait 1 more week and take the merge conflicts?
 - DJ: would be pretty easy to keep things up to date. So let's hold up on mine till after today, and I'll submit a new one.
- CW: let's go older to newer then.
- CW: [#186](#)
 - Vertex input refactor to int-keyed objects
 - KN: generally speaking, happy with doing what DM suggested. Rest of them is more complicated.
 - KN: discussion's been just happening, most people haven't seen it yet. For vertex inputs with push model, don't need to be holes in vertex buffers array. All you have to do to represent vertex buffer indices that aren't used is to have an empty array of attributes. Each vertex input has array of attributes. If empty, nothing happens.
 - Don't need sparse array, map, object, etc.
 - DM: piece of code I provided with "fill" call - do you think it's valid JS that users can use?
 - KN: yes, it's valid.
 - JG: it's weird.
 - KN: nobody will likely have to do it. People will probably just do the other thing.
 - DM: people have options. All of these are edge cases.
 - JG: I still like the keyed-index thing.
 - KN: I'm neutral between them.
 - DM: think if we have the keyed-index thing then we should modify `setVertexBuffers` to match.
 - KN: we can't pass an object in there because of GC. Can't have any super hot paths take Objects.
 - JG: only if the Object changes, and you need a different object.
 - KN: then app has to hold on to it. Actually true for what we have now, where we have an array. An array is also garbage.
 - CW: most people don't care much either way?
 - KN: DM cares the most. :)
 - JG: it's a little weird to use a default object as the object type for this.

- KN: in the bindings it does mean we have to manually deconstruct the object. Right now, not associated with a perf cost, but in theory, could have a perf cost. Because you have to manually take JS objects and deconstruct them, rather than an autogenerated thing that deconstructs them for you.
- DM: don't understand what this refers to.
- KN: if we have Object in the IDL then we get V8Object in C++ and we have to manually deconstruct it.
- AE: we have to go fetch the integer keys, get what they point to, try to get VertexBufferDescriptor, etc.
- DM: you have to do this when calling the native APIs. Just a question of how you get there. No more work with this approach.
- KN: doesn't currently impact perf, AFAIK.
- RC: so will setVertexBuffer change with this?
- KN: no, it doesn't have to. Should consider whether it should change or not, separately.
- RC: next Q then: for this binding sequence, can have gaps here and it doesn't matter?
- KN: the index matters. For DM's proposal there's a default value you use to fill in the holes.
- RC: what's the default value?
- KN: "this vertex buffer is not used for any attributes". Doesn't matter whether you have a vert buffer there or not. In object proposal, you just don't have the key if you don't need it. Then you have 2 ways to represent the hole.
- JG: did you put up a PR with your proposal?
- DM: no, have not. I can, if you think it'll foster the discussion.
- CW: we should just solve this for the sketch IDL. Can always change it. Most code will stay the same.
- DM: can we go a little step back and get to the sequence? Sequence is a problem for GC?
- KN: if you create an array every time you call setVertexBuffers, yes, it generates garbage. App could store it off somewhere and reuse it. JS math libraries do this.
- RC: so setVertexBuffers makes a copy of the sequence behind your back?
- JG: no. if you specify a new JS array every time, it generates garbage. There's no extra copy if you hold on to it.
- KN: default value is only for initial setup. setVertexBuffers is at runtime.
- JG: we can agree that GC pressure isn't the main concern as long as you can cache your input.
- CW: nicest / most JS-y is to have an array. So either sparse array or DM's proposal seem nicest, not int-keyed object.
- JG: most JS-y thing would probably be sparse arrays.
- KN: technically, int-keyed object would let you pass an array.
- CW: so we can go with sparse array for now, and later we can go with int-keyed object if we need to without breaking code?

- DM: can we take one more week to think about the ways without sequences at all?
- CW: if you can provide by homework deadline, sure.
- DM: it's about consistency / soundness of the API.
- MM: are you talking about calling the function multiple times?
- JG: you don't want to call the function multiple times.
- CW: crossing the JS -> C++ boundary is expensive.
- DM: so why isn't our bind group binding receiving a sequence?
- KN: I think they should be the same. They're not that much different.
- CW: let's make everything sparse arrays.
- MM: setting BindGroup, it's common to want to set an arbitrary random BindGroup. But for vertex buffers that might not be true.
- KN: BindGroups also have to be built first.
- MM: I might want to set 3rd BG, then do some work, then set the first, etc.
- CW: sparse array for BindGroup creation.
- MM: ah, I'm more on board with that.
- JG: sparse arrays. Make a PR for other things by deadline next week.
- DM: how would they look in the IDL?
- JG: they look like array.
- KN: sequence of Thing? .
- DM: this is better than using default descriptors?
- JG: it's more JS-y. Easier to use.
- DM: ok.
- JG: slight consensus for sequence of nullable objects. Do we have a PR for this?
- KN: in summary for this PR there's a link to a diff that has this in it. It's already been written, just doesn't exist as PR. I can do this.
- [#202](#)
 - Anisotropic texture filtering
 - CW: until recently, covered by patents. Should not have this in the sampler, but should be in extension.
 - JG: ideally, there aren't a lot of devices that *don't* support anisotropy.
 - KN: the reports on gpuinfo.org aren't representative.
 - JG: it's more useful for finding out which vendors support a given OpenGL / ES extension.
 - MM: can always add this back.
 - CW: so we remove it, can always add back.
- [#211](#)
 - depth-stencil texture formats
 - CW: bit sad that we can only have 32-bit float. Only cross-platform option though.
 - JG: does iOS not support 24-bit depth buffers?
 - KN: only 32-bit.
 - CW: possible to emulate 24 on top of 32? Legal in terms of precision?

- JG: think it might be OK, but copies are a problem - between R32 and Depth32 e.g.
- KN: copy would be compute invocation.
- DM: do we support blits of depth/stencil buffers?
- CW: Metal only supports from/to same format textures.
- MM: this is something we can add later.
- [#217](#)
 - move Window.GPU to Navigator.GPU
 - KN: we already agreed to this. Think we should just merge it and figure out mechanics later.
 - JG: it has merge conflicts now.
 - CW: Kai will update this. Will figure out the details later.
- [#213](#)
 - provide getSubData as sibling to setSubData
 - MM: didn't we discuss this 2 weeks ago while CW was absent?
 - JG: I think we were moving toward having getSubData if and only if we have setSubData, and consensus direction was toward removing both of them. In particular, in favor of createBufferMapped to handle common case of pushing data at the GPU.
 - MM: impl doesn't have to create new buffer when implementing this, so JS polyfill for createBufferMapped would probably be worse than this call.
 - CW: if you use ring-buffer-of-buffers, probably better than creating mapped buffers over and over.
 - MM: impl depending on what state the buffer's in can do get/set directly and might not need staging buffer at all.
 - DM: that's an interesting point. In Metal you'd have it in the same process as the GPU? Need to queue the update though.
 - MM: if the stars align it can turn into the memcpy.
 - JG: I'd like to not rely on the stars aligning.
 - MM: isn't that what perf is? I found extra work, want to remove it.
 - JG: way these APIs get perf is by making every step explicit. You don't have to trust us to make it fast. Rather see our API be able to support any optimization the browser can do directly.
 - MM: this call is the explicit API. It's a call where the programmer can upload data via the fast path.
 - JG: it's not explicit. It's not saying, upload this data, and try to do it fast. It leads to perf differences between browsers and devices - those with UMA and those without. Want to make all of these service the author.
 - MM: we shouldn't do this by artificially slowing down code.
 - JG: if you see an optimization here then let's enable it in the API.
 - CW: could be done with map-write-async and buffer subranges.
 - JG: it's more limited but more powerful.
 - CW: setSubData helps beginners.

- JG: I think this is a non-goal.
- CW: getSubData doesn't help as much. They'll copy to buffer, mapReadAsync. Use the buffer only once.
- JG: you'd like to reject this pull request?
- CW: sort of. In terms of usability: makes sense for the API to cater a little to usability.
- JG: I don't think that should be the goal. Don't think it makes it easier to use the API. Don't think it enables anyone who would previously not be able to use the API, to make them able to use it. Maybe we'll ship an update that improves perf in one benchmark, and then an ISV comes to us and say "you slowed it down", and we have to figure out how to make both cases fast. Or on phone it's slow, because of recent changes. Goal of this API is to not be easy; to make it easy to make all the right choices yourself and the most reliable for perf. Stated goal to make it easier to use than WebGL is just wrong.
- RC: so you don't want setSubData?
- JG: correct. I'd like to remove it.
- RC: without it, can people do all the opts that the browser would do themselves?
- JG: maybe. If we don't allow subrange-offset-mapping, then no.
- CW: we will allow this. It's just a question of when.
- DM: you mentioned you can emulate this with mapWriteAsync. I have doubts. If you update this every frame you can call setSubData. If you map, you have to pull, can't do anything until the map completes. Different workflow. Do see value in allowing UMAs to reduce copies. I take MM's point that if this can be an optimization of UMAs, ...
- JG: I don't think we should take advantage of this optimization. We should make that possible for people to do with the rest of the API. Should be able to map this UMA data directly from JS and let them write to it.
- CW: while I agree with you I don't think setSubData is a penalty toward optimization. Can use ring buffer.
- JG: I disagree. It's hard to write these optimizations because we don't know what the user wants, what their access patterns are. Write your own ring buffer allocator.
- CW: use ML. :)
- JG: maybe, actually.
- DM: how would the user take advantage of UMA?
- CW: if user knows it's UMA they can always create with mapped bit and there's no perf penalty.
- CW / DM: discussion.
- CW: if you use createBufferMapped in single-process on UMA, it's zero copy. Later, one-copy. createBufferMapped / async provides the optimal number of copies.. With regard to setSubData providing better optimization opportunities, I don't think there's a reason to keep this inside setSubData.
- JG: rewinding: if we make a straw poll on this, think we want to reject this PR.

- MM: straw poll is too simple. We think if setSubData, should have getSubData. They should come together.
- KN: I don't necessarily agree with that. There's a fundamental asymmetry there.
- JG: I'm willing to do this in 2 parts, 1) not add getSubData, and later 2) remove setSubData. Understand you want all-or-nothing.
- MM: that was the point of this PR - to make them match.
- JG: I'll submit a PR to remove setSubData, and we can talk about that there. It sounds like people are in favor of not having getSubData if we don't have setSubData. We should talk about whether we should have setSubData.
- CW: please do make the PR and we discuss whether to remove setSubData.
- DM: is there room for setSubData not receiving ArrayBuffer?
- CW: what would it take?
DM: would return Promise but not have same constraints as mapping.
- MM: think that's a good idea if we get rid of setSubData.
-
-
- [#235](#)
 - Remove clamp-to-border-color option from GPUSampler
 - Not implementable on iOS, not supported on macOS 10.12.
 - RC: no objections.
 - JG: I'm sad but OK.
 - CW: it can be an extension.

Agenda for next meeting

- [KN] put up sparse arrays for vertex attribs
- CW: shading language?
 - JG: if dneto@ can attend.
 - CW: and Robin.
 - DS: I can tell David that it's on shading language.
 - MM: Robin can almost certainly join too.
- CW: OK, WebGPU programming model then.