GPU Web 2020-06-01

Chair: Corentin
Scribe: Ken
Location: Google Meet

Tentative agenda

- VF2F dates
- Texture views format reinterpretation #744 #811
- Add minimumBufferSize to BGL entry #678
- Tweaks to the mapAsync function #796
- Development-only API features #814
- Agenda for next meeting

Attendance

- Apple
  - Dean Jackson
  - Justin Fan
  - Myles C. Maxfield
- Google
  - Austin Eng
  - Brandon Jones
  - Corentin Wallez
  - Idan Raiter
  - James Darpinian
  - Kai Ninomiya
  - Ken Russell
- Intel
  - Yunchao He
- Microsoft
  - Damyan Pepper
  - Rafael Cintron
- Mozilla
  - Dzmitry Malyshau
  - Jeff Gilbert
- Matijs Toonen
- Mehmet Oguz Derin
- Timo de Kort

VF2F dates

- **June 22, 23, 24 11AM to 2PM PST**
- Doodle
- CW: 6 hours of meetings that week to start, and extend to 9 if we need to?
- CW: Okay general agreement. I'll send invites.
- Please reserve Wednesday too just in case we go over
- MM: is there anything scheduled for WebGL yet?
- KR: not yet - could do so if there's interest
- CW: there's a 3 hour time slot on Thursday that Damyan can't make, but not sure if he's going to want to attend WebGL

Administrative

- W3C staff still eager to create WG - iterations on the PR
- DJ: I'll update the latest version. We expect a response from Apple Legal in the next day or two.

Texture views format reinterpretation #744 #811

- CW: with #744, the concern is that if we allow texture views to be reinterpreted, on various APIs, textures will be created with suboptimal parameters. Could prevent textures from being compressed and use more memory bandwidth. Should be avoided if not necessary.
- CW: how to avoid? In #811 a suggestion is made similar to a Vulkan extension; when you create texture, spec add'l list of compatible formats you want to potentially use. Diff impls on diff APIs can decide which special flags they might need to use to implement those. Metal backend could be more efficient if we don't need the usage. On Vulkan, too, there's such a flag as well, pessimistic on some hardware though. RGBA8 unorm -> uint doesn't disable compression. Explicit list allows choosing most optimal path always for the APIs.
- MM: does the Vk spec say you need to specify the flag?
- CW: yes.
- MM: so not specifying flag is wrong. What are you saving?
- CW: In Vulkan with that extension, you specify the list of formats you want to be able to convert to, and the driver will decide if it can do compression or not. Instead of a single flag with a baked list of formats it allows, you explicitly state all the ones you want.
- MM: how many devices have this extension?
- CW: assume many, but have to check.
- JG: If it's VK_KHR_image_format_list it was promoted to core in 1.2. Which doesn't answer the question really..
- MM: most devices don't support Vulkan 1.2 though.

- CW: 65% on Windows, and 23% on Android. This is an optimization for the Vulkan implementation. If we don't have this extension and the list of formats passed is too big to not have the flag, then we add the flag.
- MM: what are the rules for D3D?
- RC: D3D has tables where for a given image format you can determine what you can do: render, generate mipmaps, etc. Nothing I'm aware of where, for a certain use, will it be slow or not.
- JG: is that untyped vs. specifying-a-type?
- AE: typeless can prevent compression.
- RC: I'm not sure about typeless preventing compression - I can ask about that.
    - https://microsoft.github.io/DirectX-Specs/d3d/archive/D3D11_3_FunctionalSpec.htm
    - "Any Resource type may be created as "Prestructured+Typed", also known as creating the Resource with a fully-qualified type or format. In general, this may allow Resource optimizations, especially when the Resource is created with flags indicating that the Resource cannot be Mapped/ Locked by the application."
    - So it seems TYPELESS may not allow such Resource optimizations.
- MM: even if typeless prevents compression, is that actionable? I'd think D3D would ignore this list of formats.
- JG: if you have this list, and you say you're only going to use this list of formats...think it could fit in (?) (not needing to use typeless?)
- CW: we haven't found the rules for D3D12 usage in this area. Maybe need to wait until we see that and decide?
- RC: is the proposal to do what we do with image format list?
- MM: there is a proposal on the table - #811.
- MM: proposal I was going to make (didn't have time) was to follow Rafael's suggestion and just have tables in the spec. Matrix, rows are formats for texture, columns are formats for pixel view. "Yes" or "No" entries - can you do this or no.
- CW: would there be (for Metal) yes's for pixel format usage requiring decompression?
- MM: I'd say no; illegal to use any combination of any formats requiring this flag. That's first pass. Second pass could be opening this up and requiring a flag, so matrix becomes a tri-state.
- CW: worried that these will be API-specific. On Metal, don't flags depend on GPU generation?
- MM: no they don't. Think we'll need to know whether the flag matches on all 3 APIs. If they all agree, seems like the way to go. If they disagree, having something with more detail makes sense.
- CW: The concern is that for Vulkan, the VkKHRImageFormatList extension was made because the format list and conversions that require the flag was pessimistic for some hardware.
- MM: I'd like to know more than it might be different on some hardware.

- CW: On Intel hardware, if you set the flag on Gen9 or 10 it has to disable all compression because the hardware doesn't know how to do sRGB compressed decode. Only sRBG is the problem though.
- DM: "When using VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT with Android hardware buffer images, applications **should** use VkImageFormatListCreateInfo to inform the implementation which view formats will be used with the image. For some common sets of format, this allows some implementations to provide significantly better performance when accessing the image via Vulkan." (from Vulkan spec)
- MM: I wonder if our rules for WebGPU were different from the rules for Vulkan.. you pull out the ones that would be Yes/Maybe in the table and add it to the Vulkan extension.
- CW: yes, that'd be how we implement it. Having the user explicitly give list of formats eliminates the choices of the tradeoffs to make and things to not support.
  For example, it would be unfortunate if Intel asks to remove the flag from X cell of the matrix because it's bad on our hardware (even when not used)?
- MM: if there is a right answer, given all the hardware that's out there today, makes sense for this group to find that answer and put it in the spec. If there's no right answer that matches the current hardware today, your proposal makes sense to me. But if all the devices agree on when to enable / disable compression, then think this is a bad design.
- CW: think someone will have to build the matrix for all the APIs, and we'll have to reach out to all the IHVs.
- MM: won't have time to write this matrix this week, so perhaps someone else could pick up this task?
- CW: can we start an issue, and Rafael can put in rules for D3D12, and maybe you can extract rules for Metal, we'll extract the rules for Vulkan and ask some IHVs?
- MM / RC: sounds good.
- AE: I added a link above to the D3D / DX functional spec. Statement about some optimizations and when they're allowed - i.e., not for typeless.
- CW: I'll create the issue and link to people.
- DM: can we also pay attention to where particular formats can be used? E.g., RGB8 view, create Uint32 view, is it still renderable? Are there restrictions on this in D3D12 for example?
- MM: makes sense. Also can we use #744 for this?
- CW: yes.

Add minimumBufferSize to BGL entry #678

- CW: almost had some numbers as of ~2 hours ago, but I think looking at the WIP CL, it's solving the wrong issue. Still need to see whether the code's correct.
- AE: will iterate on it.
- CW: David was concerned that if for unsized arrays, if we don't include the last element, it makes the code instrumentation much more expensive. DM can you take this one?
- DM: there are different sub-issues here.

    a. how do we handle validation and shader compilation? Here, we have mostly converged as a group. We can make this flag optional, and it doesn't affect shader compilation. If you don't specify the flag, you get per-draw-call validation.
    b. Whether or not we require at least 1 element of unbound array to be included in min buffer size. Given latest findings by David, seems we should definitely do that. I thought we could mask it out, but on Metal accessing anything out of bounds can hang the driver, so it has to be a real branch and it will have real cost. So I think we should include at least 1 element in the unbound array.
    c. how do we work out semantics of array length, if we add an element ourselves? user will get array length they don't expect.
- MM: you want to put length of the array in the array?
- DM: no, the OpArrayLength is a shader instruction where you can ask for number of elements.
- MM: in Metal we handle this by passing another argument to the shader.
- JG: that may very well be how it works.
- MM: is there another way?
- CW: that's how we do it in Dawn as well.
- MM: Vulkan it might be hard because you're only required to have 4 bind groups, so passing additional data may be tricky.
- CW: we could use push constants for that, but OpArrayLength is a thing in SPIR-V.
- MM: Vk will use SPIR-V, Metal will add an add'l member...
- CW: D3D in most cases it’s fine because you put sized descriptors in descriptor heaps. We found recently if you promote dynamic storage buffers to root descriptor, then these are unsized, so we need to do something specific there like pass the size as a root constant.
- MM: and if no room for root constants, promote it to a buffer?
- CW: yes. The other possibility RC suggested is, make the root descriptor into root descriptor table. Encode dynamic descriptor in descriptor heap, point to that.
- MM: a buffer in our impl is two pieces of memory. The buffer itself, and another resource that gets bound to the shader.
- DM: I see.
- MM: you know the shape of this at compilation time. You're passed pipeline descriptor, so you know how many of these extra fields need to be baked into source code of shader.
- CW: SPIRV-Cross already has option to generate extra buffer where you pass in list of sizes.
- DM: Okay, is OpArrayLength not a problem then?
- MM: I think it's not a problem.
- CW: Okay so as part of draw call validation, for the buffer bindings that don’t have minimumBufferSize set in the bind group, we validate that they respect the minimumBufferSize for the pipeline which is size of the static interface plus one element of the unsized part. If that fails validation, there’s an error and your draw call doesn’t

happen. When you set minimumBufferSize on the bind group, it is checked against the interface.
- MM: I'd like to not resolve until we have the actual numbers.
- AE: I think we'll have them next week.
- CW: hopefully next week we'll have an impl that's optimized enough that we can say, these are good numbers. If it feels fast enough then...
- MM: I don't know what "feels fast enough" means.
- CW: we'll look at result of investigation and see if there are red flags.
- MM: we'll look at the results again next week then.

Tweaks to the mapAsync function #796

- CW: When reviewing the PR, DM had a concern that it wasn't clear that mapAsync was for read or writing. For readability, it would be better if there was some way to differentiate the reading and writing cases. Two possibilities: two function calls, or have flag arguments. For now just read/write flags. In the future there could be a Discard flag for example.
- DM: can you clarify what you mean by "can't import shmem on GPU" case? Or, why size / offset are needed for writeAsync?
- CW: yes. When you map for writing and have to copy from shmem to staging, app is saying map a subrange, makes impl know that range is dirty. Knows it needs to flush this from shmem to GPU-visible memory on unmap.
- DM: we know the dirty regions by the getMappedRange calls. Don't need that for mapWriteAsync.
- CW: Right. nevermind.
- DM: seems to me the flags make sense on mapWrite only. mapRead only needs offset / size.
- JG: there are use cases where you want a mapRead with offset/size. That's the common one generally pointed out.
- DM: I'm saying only read needs offset / size.
- JG: doesn't hurt to have them on the write.
- KN: If we don't have offset/size on mapWrite, doesn't that mean we might need to preallocate more memory than necessary?
- CW: I don't think so because the way the spec is written, it's not mapWriteDiscard - you probably need to keep the memory for the lifetime of the buffer, so JS can see the data you wrote before.
- JG: Sounds like an optimization opportunity to have the offset and size.
- KR: If you subsequently copied to that buffer, then it would invalidate JS's cache.
- JG: For now that's forbidden, but maybe in the future we'll allow it.
- CW: for now a mapWriteBuffer can only be copy-src, so can't mutate it on the GPU.
- JG: I want to be careful about relying on constraints that we have in our MVP spec in order to simplify too many things based on simplifications we've chosen for our MVP.

- KN: don't think that adding size/offset to mapWrite later would be a problem. They could be optional, and if found to be necessary for new features...
- JG: That's possible, but the best end situation is.. I expect that even if we start with different mapRead/Write signatures, I think they will end up the same. Might as well start that way.
- KN: I'm perfectly happy to have offset/size on mapWrite. Also promote possibility of sub-range tracking. Different regions of buffer with different usages, which that enables. Adding API for that would be fine. It's the optimization opportunities I'm worried about now.
- DM: OK. To me it sounds like of whether we agree that in the future we'll have read/write mapping, or mappings combining GPU reads/writes. And we don't have subrange tracking. So for current PR it's not necessary. But for now it seems best to have single function with the flags, and offset/size.
- CW: So the second version essentially, and always requires the flag even though it's redundant right now. Because a buffer can only be MapWrite or MapRead.
- JG: it'll be obvious to the author which one to do. You're either going to read or write.
- CW: I think it'll confuse people because they'll say, I can pass flags, so I want to pass READ | WRITE, but that's forbidden.
- JG: it's constrained already by creation parameters, so if you keep that in mind that's obvious.
- CW: so we go with option 2?
- MM: if we go with option 2, offset/size fields won't be ignored for the writing flag? We'll implement them for both?
- CW: yes. In spec right now, it's like option 2 without the flags. Offset/size are stored as internal fields on the buffer. getMappedRange has to be a subset of the range defined by offset/size.
- MM: can I defer to Justin?
- CW: sure. Any other comments?
- (none)

Development-only API features #814

PR Burndown

- KN: Add GPULimits.maxUniformBufferBindingSize. #803
    - maxUniformBindingSize
    - Does having this sound good? No objections?
    - CW: it's required by hardware?
    - KN: yes. Trying to find other one...
    - MM: do we know this is the minimum of what all 3 APIs require?
    - CW: yes.

    - KN: yes. Metal's confusing because feature table has something saying "maximum number of function constants" - can't find docs about this. It's 65536, so at least 64K
    - MM: it's like a specialization constant in SPIR-V. Different than max uniforms. Think there is actually a limit in Metal.
    - KN: it's a limit on the number of args you can pass to a function.
    - MM: we should just accept this, and if it's not compatible with Metal, we'll find that when writing a test.
    - JG: sounds good, I'll just merge it.
- Move pipeline statistics query to render encoder and compute encoder #797
    - CW: Removes from render bundles and removes from pipeline statics query because there can only be one at a time. Sounds like no concern
    - MM: what's the concern? Oh, not supported on render bundles?
    - CW: correct.
- Disallow using image cube or cube map array texture views as storage textures #798
    - CW: HLSL doesn't support Read-Write texture cube or cube array. Only 1D/3D.
    - JG: And you could just make a texture view that's not a cube map if you wanted to store into it, right?
    - CW: Yes.
    - JG: Agreed
- Add validation for GPUDevice.createTexture() #799
    - MM: I think the only issues were (except what Kai said) are all stylistic.
    - KN: My comments are minor.
    - JG: we have 8K textures by default? Sweet.
    - CW: I had a concern about 3D textures but we can iterate offline.
- GPUSamplerDescriptor and GPUSampler creation spec #805
    - not ready; discuss next week
- Add BC formats. #812
    - CW: There was an investigation from Jiawei about this a while ago. The controversial part is that it adds stuff for an extension to the main spec. We should define how extensions/features work, and whether they are separate documents or in the main spec.
    - JG: don't think it matters a lot.
    - CW: think it'd be nice to have things we expect all browsers to implement in the same spec. But if someone writes a raytracing extension, or mesh shaders, etc, then maybe we keep it as a separate document until there are 2 browsers implementing it.
    - KN: I agree with that.
    - JG: concern: How behavior is different when a UA knows/doesn't know about an extension. If you add a new PVRTC texture format, for example, if the browser doesn't recognize it it'll throw if you try to use it. If it does recognize it but doesn't support it, you'll get the error monad.

    - KN: the behavior described in errorConventions.md - if you use an enum from a non-enabled extension, it'll throw an exception. E.g., using an enum in createTexture when extension isn't enabled. Have to spec that out for all the entry points.
    - JG: sounds good.
    - MM: putting common extensions in the main spec increases readability. One stylistic request: it should be more clearly enabled that these things are not expected to work everywhere. More than just a comment.
    - KN: it's normative. There's a list of extensions that are enabled; if we see any of these enums, and the extension isn't enabled, throw an exception.
    - MM: more a question of style in how the spec's written. It's easy for someone to scan the list and think they can use these.
    - JG: it'd be hard to read these formats and not also see the comment.
    - KN: this is up for editing - think it doesn't have to do with this PR, and can be done without too much difficulty.
    - CW: there's probably an algorithm "ValidateTextureFormat". Is this an extension format? If so, if extension isn't enabled, throw.
    - JG: if I'm reading the spec I'll understand this by the comment, not by the detailed algorithm, which is more for UA implementers.
    - KN: agree, it's more up to the editors how to describe that this list is extensions.
    - MM: One suggestion is to change the background color or font or color.
    - KN: have been planning to do that for all extension stuff. Maybe checkboxes turning extensions on/off, too, so you can read the spec without extensions?

Agenda for next meeting

- More depth stencil sampleability / copyability?
- minBufferSize
- Please take a look at https://github.com/gpuweb/gpuweb/pull/823
    - Please weigh in and resolve offline
- Texture compression, if we have all investigations?
- Whatever else is in the ForDiscussion section of the Github project.