So as to not be anonymous animals, the doc is shared for writing with the google accounts that are invited to the meeting. If you can't edit let cwallez@google.com know. Also be sure to be in "Edit" mode and not "Suggest" mode.

GPU Web 2020-06-23 VF2F Day 2

Chair: Corentin
Scribe: Austin / Ken
Location: Google Meet

Tentative agenda

WebGPU

- 11:00 AM: ImageBitmap discussion
- (filler): Add time query on GPUCommandBuffer #870
- 11:50 AM: break

WGSL
To be determined, add your topics below:

- Reminder: looking for proposals
- Discussion of backlog blocking CTS (dneto) slides
- Namespacing & imports & modules (#777)
- Interface matching rules (#644)
- Textures (#573) and Samplers (#574)
- Task: Invent a standard library (#667)
- consider namespaces for WGSL (#777)
- Enums (#661)
- Specialization constants, function constants, or other forms of shader modularity (#572)
- Packed vector types (#780)
- WorkGroup memory size and barriers? (#863)
- A single shader is often split among many source files (#741)
- C-like declaration order (#677)

- Agenda for next meeting

[Agenda/Minutes doc for Day 1]

[Agenda/Minutes doc for Day 3]

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**
  - Fil Pizlo
  - **Justin Fan**
  - **Myles C. Maxfield**
  - **Robin Morisset**
  - Theresa O'Connor
- Google
  - **Austin Eng**
  - **Brandon Jones**
  - **Corentin Wallez**
  - **Dan Sinclair**
  - **David Neto**
  - **James Darpinian**
  - John Kessenich
  - **Kai Ninomiya**
  - **Ken Russell**
  - Shrek Shao
  - **Ryan Harrisson**
- Intel
  - Brandon Jones
  - Bryan Bernhart
  - **Jiawei Shao**
  - **Shaobo Yan**
  - **Yunchao He**
- Microsoft
  - Chas Boyd
  - **Damyan Pepper**
  - **Rafael Cintron**
  - **Greg Roth**
  - **Tex Riddell**
- Mozilla

    - **Dzmitry Malyshau**
    - **Jeff Gilbert**
- Kings Distributed Systems (KDS)
    - **Dan Desjardins**
    - **Wes Garland**
    - **Dominic Cerisano**
- Joshua Groves
- **Matijs Toonen**
- **Mehmet Oguz Derin**
- **Pelle Johnsen**
- **Timo de Kort**

ImageBitmap discussion

- ImageBitmap
- <Shaobo goes through the presentation>
- KR: On the video to texture path, there are optimizations, so that when the GPU accelerated decoding is done, in Chromium, Safari and Firefox, a GPU-GPU copy is happening. Do you know why that's not possible in WebGPU?
- SY: It's because WebGPU is using ImageBitmap, and Chromium always puts the imagebitmap on the CPU.
- KR: The Chromium ImageBitmap implementation should be improved. There's no good reason why Chromium does this, and should be looked into further.
- KR: During the import of the externally produced there is a copy from GL to external Vk. I thought there is interop between GL and Vulkan. Why is a copy needed?
- SY: The reason is the SharedImage system, and there are two backing factories. THe GL and the shareable backing factories. Seems there is a concern with performance always using shareable resources.
- CW: Allocations of textures that can be shared between multiple APIs usually have bad layouts that are not optimal to use. You want to use them for interop, but don’t want to use them for rendering because they will be less performant.
- KR: Okay, so allocating with shared bits set lets them be used for interop, but not for general use.
- CW: Agree with Ken that there’s a bunch of Chromium impl details that make it worse than it should be. The complexity, however is likely to happen in other browsers. Shaobo’s suggestions make it much easier for all browsers to implement fast paths -- and the 0-copy option is very good.
- KR: I don’t follow how that (0-copy) will work. The ImageBitmap seems unlikely to be produced in a way that it is fully shareable.
- CW: createImageBitmap({ forWebGPUDevice }). Then device.ImportImageBitmap(ib) -> GPUTexture. Also good for multi-adapter systems.

- KR: Can we also think about multiple contexts? Should consider having it work as well for 2D canvas, WebGL, etc. Take a list of compatible contexts.
- MM: Is there precedent in the web for API-specific flags in different APIs?
- CW: Canvas in context-creation has WebGL-specific attributes.
- JG: That's for creating a webgl context. WebXR creation can take a WebGL context which is similar to this. You want WebXR to operate with WebGL so you specify it.
- BJ: Aren't there parallels with how media streams interact with videos? This was a topic of conversation in the XR group. Looking at the docs, I know there's interaction between WebAudio, getUserMedia, works with video/audio tabs. Seems like some cross-API interaction as well.
- CW: To make one detail more precise: whatever forWebGPUDevice or compatibility list, etc. You would still be able to use it everywhere -- it is a hint to the browser to optimize for a certain use case. The intent is not to limit it.
- DM: I'm confused. Are we putting it into the image bitmap or the producer of the image bitmap?
- CW: I think they're produced using the constructor of ImageBitmap
- KR: Would be a dictionary member for that constructor.
- SY: how do FF and Safari support ImageBitmap?
- JG: From video element to WebGL via texImage2D, it is accelerated GPU-GPU. Everything else is CPU, unfortunately.
- KR: If you had a video element and created an image bitmap, could it be GPU backed?
- JG: Maybe in late 2020. We have no architecture for that now.
- KR: For WebGL, we've used big union types of HTMLImageElement | VideoElement | ImageBitmap. Trying to unify around ImageBitmap. It seems that this isn't actually super performant today. Do you all think this was a mistake? Or should we continue to try to unify around ImageBitmap?
- CW: I think the ImageBitmap is appealing because it already has transfer semantics. This helps a lot when doing the zero-copy approach. For HTMLVideoElement it's more unclear. Maybe you need a copy always.
- KR: That's fair. The WebCodec effort is aiming to get an even lower level API for video streams that could be used on other threads as well. I think it uses ImageBitmap as well and could give you YUV planes. Some customers want to do the color conversion themselves.
- SY: Yes, I think this is what is most useful for the zero-copy path. Also, ImageBitmap generated from the video frame has two options for CPU/GPU backed. I think HTMLVideoElement could have a similar strategy.
- CW: What do other browsers think?
- DJ: We have a fair bit of work to completely implement ImageBitmap in Safari, but I think it is the right approach.
- CW: Using ImageBitmap as the only interop thing for textures and the rest of the web platform?
- DJ: I could go either way. Not sure about *only*, but it should be one of them.
- CW: What do you think about the hints? And ProduceGPUTextureFromImageBitmap?

- DJ: No strong opinions yet.
- JG: It's hard because this area hits on some of the harsh realities of minimizing number of copies and extra work we have to do. Conceptually it would be awesome if you could just create an ImageBitmap which referred to the underlying image. And then you could sample from it directly in WebGPU or WebGL. We have some efforts toward an extension like this for videos for WebGL. It's promising. It's okay because uploads are usually immutable, so reading from it works pretty well. I think ImageBitmap is supposed to be the container that doesn't let you get at the data directly so it's great for GPU-side resources or possible-GPU-side resources. It's probably fine, but I think it would help if we had a stronger use case driven scenario. It's hard to reason about what is best without centering on use cases.
- CW: I think one big one is the perf of video interop.
- SY: For TF.js ResNet demos, there is a camera feed, and if we use ImageBitmap from video element, we can't get as good perf as WebGL.
- JG: One of the concerns could be: is it too late by the time you're at ImageBitmap creation? Wouldn't you want it to be for the video decoding?
- MM: That's what I was confused about. If it's on creating the ImageBitmap, it doesn't completely help because you still need to decode it and then send it to a different device.
- SY: .. asnyc API could hide
- MM: I thought this flag would only change performance, and not behavior.
- SY: Yes.
- KR: having the available contexts at creation time still makes it easier for the browser to know the type of resource to produce.
- KN: Arguably, the place where such optimization should go is WebCodecs. We shouldn't necessarily make this optimization here if WebCodecs is going to make this fast.
- DJ: I agree. It's hard for us to do that anyway because the video draws into a sequence of images that are sent to the compositor. We have to copy out of that queue anyway.
- KN: I believe that's the same of our architecture and generally true if you have to deal with hardware video decoders.
- CW: Think it still helps eliminate one or two copies.
- KN: Yes, when you create the image bitmap you need to create an image bitmap anyway, so you want to do the correct copy at that time.
- CW: Okay, how do we move forward on this topic.
- JG: What are we actually missing? Don't think we're missing anything except performance. You can still make demos with it being not great. There's room to make it not terrible without spec changes.
- CW: I think the ProduceTextureFromImageBitmap.. copyImageBitmapToTexture has copy in the name. It will always have a copy. I think WebGPU is missing the ProduceTextureFrom.. path. Then afterward we could see about hints for making the ImageBitmap faster..
- KR: I think we should see how fast it can be made. WebGL has done pretty well on video texture uploads. Seeing all of these pitfalls in the ImageBitmap upload makes me think we should do a prototype of the compatible context options, and the zero-copy wrapping.

If it's good, then we can think about pursuing spec for it. The zero copy path here sounds similar to tearing off the front buffer for low latency canvas. The challenge would be providing enough metadata to the ImageBitmap so you can use it correctly in your app. If we find that createImageBitmap should have the compatible contexts member, we should push that up to WebCodecs.

- RC: Agree on a list of contexts.
- CW: I guess Chromium can try a prototype then.
- DM: I have a question. You mentioned the Promise-like ability for uploading data from ImageBitmap? Without a Promise it seems if you go a slow path via CPU readback it will be blocking a lot and not be useful.
- CW: It depends where the copy happens. It's fine to do the copy. Blocking is when you rely on file system or IPC, but doing a copy is not considered blocking.
- KN: Sort of considered blocking. createImageBitmap returns a Promise because it could be decoding a jpeg. So it lets you know when that is done.

WGSL Needs Proposal:

- DJ: We have a lot of things that have no proposal.
    - select (#826)
    - discard/kill (#361)
    - numerical precision (#674)
    - short circuiting (#675)
    - expansion (#600)
    - pointers/references (#672)
    - typing rules (#664)
    - others :)
- DJ: how can we get these assigned?
- DN: Dan and I are over a hump and think we can get to text writing. E.g., short-circuiting is just a bit of work. The table of contents on the wiki is good. The structure of the WGSL doc is hard to understand where to put new text. I'm volunteering for more text writing over the next while.
- DJ: I support you volunteering for more work. :)
- MM: I should spend more time writing too.
- DN: wondering whether editors were having discussions offline, but recently learned this isn't the case. :) Having a place to discuss who's going to grab what would be helpful
- DJ: I like that idea of an async channel, to take a spare hour and write something.
- MM: two possibilities, could use Assignee field in Github issues, and emails. I prefer the Assignee field.
- DN: makes sense.
- DJ: that's fine. You grab it by taking the Assignee. We should encourage people that if you even just have rough ideas about it, take it as the Assignee, and other people can contact you about taking it over if they want to.

Making Progress on the CTS

- DJ: first real topic: how can we work on WGSL to get the conformance test suite unblocked?
- DN: slide deck
- DN: 5 milestones
- DJ: picked some topics that were slightly open-ended, good for discussion. Textures / samplers, workgroup size, were on the agenda. Like the idea of using these two milestones as the driver for our work.
- DN: agree. Think we need concrete text we can discuss.
- MM: for milestone 2 sounds like we'll need uniformity to do that?
- DN: I was thinking to not necessarily have all the validation done. Can write code, and if it's trusted code, we can write the CTS. Negative testing needs to be everywhere. Texture sampling can only be done in uniform control flow.
- MM: are you planning to ship an MVP without all the validation?
- DN: no.
- DJ: is there any reason you picked compute over rendering first?
- DN: I think there's fewer moving parts. Attaching resources for example.
- DC: can I suggest that you use matmul as the test case for compute?
- DN: my traditional one is element-wise array addition - math is usually quite fast. Want "pipe cleaner" to make sure that the entire flow works end-to-end.
- DC: the command buffer API, doing batches with the compute API, is crucial for e.g. TensorFlow. Can't do it with WebGL.
- DN: that'll probably drive workgroup size.
- GR: what is a pipe cleaner?
- DN: the first example that works end-to-end. Think of the implementation pipeline, not the graphics pipeline.
- ...
- DN: think some things are held back from lack of proposals.
- DJ: would encourage editors to look through open proposals. Define what matrix accesses mean, for example. Not marked as MVP. Some might have been done, or have some text. If you have a chance, go through open issues and mark them as MVP.

Define the interface of an entry point in a WGSL program (#774)

- MM: is there a proposal?
- DJ: thought there was...looks like no.

- DN: investigated whether there's a built-in variable declaration rule. No implication, because WebGPU doesn't have the features that would imply that. Only vertex shader in the geometry stage.
- DJ: where's that comment?
- DN: ~~not sure.~~ #644.
- MM: say you have vertex/fragment shader, linked together, VS writes attribs, FS reads them. Presumably these need explicit IDs and you use the location field?
- DN: yes. These are user attributes. Vulkan uses location and component. Interface matching rules are all about lining these up by type and number.
- MM: surprised this is a tuple of information. Example where both fields are useful?
- DN: passing 4 floats down a single location. Location == vec4. FIrst is a vec3, second is a scalar, but needs a component to tell that it tucks into a vec4.
- MM: can we not do that? Tell them to make it a vec4?
- DN: I don't know how common it is. Not sure if there are issues with mixtures. You're limited in your bandwidth between shaders which is why engines do this packing.
- MM: Metal doesn't have this concept, that's why I'm asking. Does D3D have this concept?
- GR: mismatching inputs / outputs?
- DN: where you're passing user attribs, want to use 2 diff variables to do it, and want to jam scalar along with vector.
- MM: the thing that gets interpolated is joined together from multiple variables?
- DN: yes. It might be flat, e.g., not interpolated.
- MM: I've never heard of this thing before. Metal doesn't have it, doesn't seem particularly valuable, increases complexity of the interface matching rules.
- DN: I'd be happy with an MVP without this.
- MM: OK, let's use scalars. You can use arbitrary ones too? One is 17, the next 600?
- DN: not sure - not an area I'm familiar with.
- MM: if Vk has this requirement we could renumber.
- CW: don't think it has this requirement. At least in BindGroup, we have the rule that the binding declaration can be any number.
- DN: this is a literal value. Easy for us to remap it. Would preserve order user gave us.
- DM: would be nice if it were somewhat limited, e.g., to 32 bits. Otherwise painful for us to support bizarre cases.
- MM: agree, we shouldn't need to use BigInt to implement WebGPU.
- DN: OK.
- MM: think that's it?
- DN: weird rule where VS passes vec4, FS consumes only vec3. Valid in Vulkan.
- MM: think the types should match up. That's what most authors expect. Implementations can be simpler, everything will just work.
- DM: agreed.
- MM: that doesn't affect typedefs. If typedef'd, should match on the resolved type.
- DN: also you have to have written the output before. Think we have enough decisions to write the spec for intra-pipeline.
- MM: if there is an output never written to, that's a bug in the shader?

- DN: no, outputs are always initialized.
- MM: I think that's a good design. If input you don't read from, that's fine.
- DN: if you have an input, must be written from output from previous stage
- DM: are we sure it's a good idea to auto-initialize outputs?
- MM: how would this be more error-prone?
- DM: you define a value, don't assign to it, get the desired result?
- MM: doesn't seem like a bug.
- CW: also things are initialized to zero.
- DM: interp qualifiers: in GLSL you have to use the same on both sides. An extra to the location.
- MM: don't think we have those yet.
- DN: yes, don't yet.
- MM: when we add text for them we can say that they have to match. Types, locations, qualifiers must match.
- GR: the reason why we'd want interfaces to be a bit more relaxed is to avoid the shader explosion problem. Not saying these are bad decisions, but might be a tradeoff we have to take into account.
- MM: I've heard that in bunch of different situations. Eg., shader that draws 5 lights rather than 6, or my material has 2 rather than 3 textures. Haven't seen this used when trying to attach multiple different FS with different interp qualifiers to the same VS.
- TR: case where all of the outputs are declared on the inputs. FS that doesn't use all the outputs of the VS has to declare them all - would be problematic.
- MM: makes sense that inputs of FS can be subset of outputs of VS.
- TR: that's the way it is in D3D. Matching prefix subset, then can chop it off.
- DM: that's the point of argument - we don't want to support the chopping off semantics. You can skip some outputs, but those declared have to match.
- ...Discussion...
- MM: pretty powerful since you can define the location for the inputs. Only VSs that output things in the right location are valid.
- DN: have to do coordinated compression of location space. Messes up the scheme.
- MM: yes. VS and FS have to agree on how you're doing internal mapping.
- KN: have to do that compression during pipeline creation.
- TR: don't you have that requirement already? If subsetting input, have to determine layout based on pairing of stages?
- MM: makes sense.
- DM: don't understand...
- TR: D3D requires the actual layout is the same in all stages. Doesn't repack based on usage in subsequent stages. They can only chop things off so the prefix is packed the same way. Have to pack things greedily. No later input can change the packing location of an earlier input.
- DM: we aren't talking about structures. We're talking about varyings.
- TR: I'm talking about how things are packed in memory. Uses order of declaration. Must match, up to the last used thing.

- MM: you're saying if I have a VS that outputs 3 varyings, and an FS that skips the middle one, it'll get messed up.
- TR: yes.
- MM: in Metal this would naturally be solved with function constants. We'd make these location fields function constants, and at pipeline creation time spec what they are. Can that solution be used for other APIs?
- TR: requires some fixup stage at the backend at least. Need encoding that shuffles these things around.
- DM: that requirement seems new to me. I need to do more research.
- TR: I'm not suggesting this API has to have that strict a requirement. Having the used things be a subset and have to match sounds OK, just be aware that things may need to be recompiled on the backend.
- MM: if you were implementing this on top of HLSL how would you do it?
- TR: the subsetting rules?
- MM: yes
- TR: in HLSL you'd just match the semantics. This is the behavior of the mesh shader pipeline that we shipped. Because mesh shaders have 2 different frequencies of produced attributes, and want to share pixel shaders with traditional pipeline, all ... are in the layout of the mesh shader. We match by semantic name and index.
- DM: you match it in the driver? Is this something we do?
- TR: it's done in the backend - in the driver.
- MM: if we can assign distinct semantics to every output of VS and input of FS, would that work?
- TR: I believe so, yes.
- DM: scheme we talked about would work in HLSL, then?
- TR: not saying it's a blocker. When we produce traditional pipelines, we have to know both connecting stages to do the shader generation.
- MM: does that mean creating a pipeline would be more expensive than it should be, compared to shader module creation?
- TR: pushes the price to a different part of the pipeline. Can't translate and compile shaders independently from pipeline state. May also vary based on other things in pipeline state.
- MM: I see. In D3D backend, the GPU shader module object just has a string in it.
- MM: we were hoping much of the cost could be in shader module creation.
- TR: I doubt that will be the case for this implementation, without other work to do more work up front.
- MM: any design decisions we could make to have this behavior?
- TR: not sure.
- GR: if we adopted the same rules as HLSL, now.
- TR: right. That'd be part of it. There are other details that have to be considered in translation. E.g., Vk -> D3D/HLSL model. Arguments between stages isn't the only thing.

- DM: I'd like to know more. In practice that'd limit us from creating DXIR at shader module creation. Should we consider only being able to drop location indices that are "last", not in the middle? Would like to see what other restrictions are needed.
- TR: I'd have to review. Something about offsetting resource accesses, something about how D3D handles views. Something about interacting with other command lists, and timeline linkups. Not sure if that can be fully implemented. Equivalent would be creating a view, not sure you can do that on the same timeline in the same way. Would have to look into other details.
- MM: think we have one impl (Dawn) in the works that does run on D3D, is that using SPIR-V cross? How does that work?
- DM: we run on D3D just fine. We use SPIR-V cross. Problem mentioned with dynamic offsets, we do modify shader generation at pipeline creation. SPIR-V cross patches the offset somewhere. Can't do at shader module creation. We can work with it. Consider spec'ing WGSL that dynamic buffers have the notion that they are dynamic buffers.
- TR: that's true. Maybe also other restrictions would be necessary. Only use the ones more easily mappable? Maybe not all cases can easily be mapped from that feature.
- MM: an expert on D3D probably should take an AI to figure out what it would take to move most of WGSL compilation to shader module creation.
- CW: I don't think that's possible in the general case. Always have to link in info from layout. Can still precompile a bunch of stuff, maybe even generate DXIL, but have to adapt to pipeline layout. Eg., read-only storage texture in shader. Depending on whether it's used as a R/W or R/O storage texture, type will be UAV vs. SRV, declaration in HLSL is different. Will have to update shader with layout.
- TR: this is an example where we might be able to get to DXIL, but root signature info isn't available until you have pipeline state. Some drivers might generate a form that can be fixed up more easily.
- MM: can you get to DXIL without root signature?
- TR: yes, DXIL's independent of that.
- MM: what % of final compilation time is DXIL conversion?
- TR: think the driver compile will dominate.
- MM: if that's true we should give up and just say shader modules hold a string.
- CW: it's the same thing in Metal. Bulk of cost should be in the GPU compiler.
- GR: it's opaque what the module contains?
- MM: yes.
- GR: maybe they choose to make it a string then.
- MM: the benefit is predictable performance. On Metal that's not true, you can create 2 distinct Metal functions that hold GPU-specific blobs and join them together inside a pipeline.
- DM: I don't like the idea of just storing a string. Want to do as much work as possible during shader module creation.
- KN: you'll always be able to go to WGSL AST. Any validation in createShaderModule has to be done, so you'll parse anyway.

- TR: I think we can have a form of DXIL that's almost finalized. Not sure how much that alleviates the driver compile.
- CW: driver compilers reoptimize based on the whole pipeline. VS output not used in FS; dead code eliminates it. Shader runs faster.
- DJ: this has been productive. Right now we have an example 1 in the spec, a simple program; would like it if it could be examples 1 and 2 matching up with milestones 1 and 2, even if they're incomplete. Adding 2 arrays as compute shader. For rendering, a vertex and fragment pass that do basic rasterization. See a lot of the interface up front in the spec. Sound like good idea?
- MM: maybe DM and TR should do some investigation to see if it's possible to move bulk of compilation into createShaderModule on D3D?
- GR: I'd say, what design decisions we can make to move that? Rather than impl decision. I agree that'd be something good. That should be something on Microsoft.
- TR: I'm not sure we can get there, but can see what's in the way.
- GR: maybe not "bulk" but "as much as possible".
- TR: how much does it buy us? If we can get final DXIL at that point, but don't have root signature or final driver code, what advantage is it to do a bunch of work up front?
- DM: first, you may see people creating pipelines on the fly, but shader modules are not typically created on the fly; and second, expect modules to be reused across multiple pipelines, creating with same shaders. Moving the cost helps multiple times.
- TR: think it's impl detail if we can get close to final DXIL. Just fix something up at the end. Cost can be mostly front-loaded. Don't think it will be a very large cost. Pipeline state creation will be the most. If the API doesn't expose that fact and the perf costs are unexpected that's problematic.
- CW: it's expected. We're attempting to add createReadyPipeline.
- TR: sounds like it'd be better to have the cost in pipeline creation.
- KN: shader module creation isn't async, but it effectively is since it returns only a handle.
- GR: surprised nobody used "compile" and "link" like in OpenGL. Lots of regrets about design. :)
- Action: Microsoft: come up with a description about what's happening.
- MM: we'll block discussion of intra-module linking requirements on that AI.

Task: Invent a standard library (#667)

- MM: Went through the standard libraries of the three languages, based on specifications not on writing sample programs. At least one of the backends doesn’t list all of the functions in the standard library. There’s:
    - Trig functions - the APIs all match
    - Vector operations - all, any - pretty natural
    - Didn't include "lit" from GLSL, not implementable everywhere, doesn't seem important

    - Barrier functions - issue about them isn't resolved yet. Four barrier functions I list are common between 3 APIs
    - as-casts - already in the language
    - Math functions - grab bag - there's one in here that's not in SPIR-V - log10, I think. Reason it's in there is that you can just divide by the constant. Maybe precision won't be as good as it could be. Maybe ok.
    - Derivatives - I took out the coarse versions, they aren't implementable on Metal
    - Atomics, textures and quads - deferred to those issues
    - Packing ops - pretty useful. Vertex pulling, for example. None of the packing functions exist in HLSL, but these exist in both spirv and Metal. In HLSL authors can write these themselves.
- MM: like HLSL and GLSL, would like to propose that we don't have to include specific headers like "trigonometry" and "vector" - either one include, or no includes, to get all of these. If group wants separate includes, the grouping in the issue is most natural.
- JG: I think these should just be included. I'd prefer if they're builtins.
- MM: I agree.
- DJ: there's a namespacing topic later. If the user's defined a function of that name, do we defer to that?
- MM: maybe defer topic of importing these until later. Encourage you to consider if there's anything in these lists that's missing, or anything you think shouldn't be here.
- JG: thanks for writing this. Couple of little things. isFinite with two different capitalizations.
- MM: I took the union of my research and what's already in WGSL, oops.
- JG: curious, pack/unpack - looks like it packs a single half float into unorm4x8? or 2 half-floats into 4x8?
- MM: in MSL it has the name I wrote, in spirv it's unpacksnorm4x8.
- DJ: HLSL doesn't have those, what's the overhead?
- MM: that's a question for someone from Microsoft.
- CW: what functions? ANGLE already shims them.
- Discussion about whether GLSL has these. #version 450 seems to.
    - ESSL3 p92: https://www.khronos.org/registry/OpenGL/specs/es/3.0/GLSL_ES_Specification_3.00.pdf
- DJ: not necessary to ask my question.
- Link to ANGLE's implementation.
- DJ: just wondering if there's significant overhead.
- MM: not sure. There are some packing functions in spirv that aren't in MSL, and vice versa. These aren't included. If implementing these in-browser isn't expensive, we might want to take the union.
- DJ: in JS lang proposals, they provide JS code to implement it as well. Maybe we should do the same here. Maybe those unsupported by core libraries.
- GR: these packing functions do exist in GLSL in a way - like the 16-bit ones. The 8-bit ones are a bit lacking.

- TR: not the ones that convert between unorm and snorm. HLSL only has half/float conversions.
- DN: spirv has packhalf2x16
- MM: I didn't include that, not in MSL
- DN: we'd talked about fp16 support. If bottleneck to mem is limit, treat as array of 32-bits on the GPU, but sending in/out as half. Would want that supported in std library.
- MM: please add to issue.
- DN: will do.
- MM: most of these functions in all APIs are overloaded. Transposing matrix, can transpose any matrix type / dimension. Think we want the same thing.
- DJ: what would need to be done to put this list in the spec? Link to the spirv definitions?
- MM: hadn't considered. Think best thing is to take suggestion earlier and just write an impl of the function.
- DJ: really want to do that for sin/cos?
- MM: maybe not. Not the hard part - work the same way everywhere except for precision.
- DJ: all I want is a link to say, here's the definition of sin().
- JG: I like the way ESSL does it. Tables of syntax and descriptions. Makes up a thing "genType". The language doesn't support templates but the descriptions assume overloads.
- MM: question is whether English language description is enough to nail down the corner cases.
- JG: think ESSL spec can be looked to for inspiration.
- DJ: I found writing out the type rules a good way to enumerate which expressions are valid for which operators. Then can have description of whether something works pointwise for vector, etc.
- MM: sounds good, if we find ambiguities can make them more rigorous.
-
-
-
-
-

consider namespaces for WGSL (#777) / WGSL needs a plan for expansion (#600)

- MM: posted a couple of hours ago about this. Discusses options. We have a weak preference for one of the options.
- DS: wasn't fifth option, but someone did. I agree with it: always require the namespace. That's effectively how import works now. Just change "import glsl::std450 as std" to "import ‘wgsl.std.1’ as std".

- MM: surprised to hear you say that - earlier David complained about ObjC "NS" prefix. Think "wgsl::" prefix will be painful.
- DS: having ported things over, not that bad.
- JG: I don't want that.
- MM: me neither.
- DS: it's the most forward looking one.
- MM: the four options I proposed handle that.
- DN: I like Perl / Python's optional "using" declaration at the same time. Know I'm opting into the less verbose version.
- MM: that's option 2.
- MM: pros with option 2: gives authors the choice. Import everything into namespace, might break in future; or future-proof things with wgsl:: . Incentivizes authors to break things in future. C++ suggests "using std". So if we change wgsl in the future we might break users. Don't think breakage will be that common. JS tries to not add colliding functions.
- KN: in JS your declaration shadows the default one. THat's option 3. Think there's a way to do option 2 without breakage. Explicit namespaces, wgsl:: trig:: etc , can add things without breaking. Have using statements either explicitly specify things they import (verbose) or some way where we guarantee we won't add anything, so using it is safe.
- TR: what about ...
- MM: don't think that solves the problem. One breakage: i thought I was calling std library, then upgraded browser, now calling something else. Other: thought i was calling std library, updated browser, program now doesn't compile. Both not good.
- JG: do we expect to version the language orthogonally to std lib? If versions of language, could we just wrap updates to std lib in version bumps to lang? E.g. adding while loops to next version, can't use them now, but declare "I'm using this thing now"? One way to get versioning of the stdlib is saying that has extra functions.
- TR: I don't see why it has to be split.
- KN: I think that solves the problem but there's a maintenance cost. Have to support multiple versions forever. Think we can do this without that. Hope the language can be versionless, not have breaking changes, or very few breaking changes. Maybe a "1" and "2" of the language but expected to be many years apart.
- GR: there's no 1.1? It's just frozen? If we want to make change to stdlib and someone who has local function with same name, that's breaking.
- KN: I think we can come up with way to upgrade things in a non-breaking way.
- JG: it's a good point, would be nice to not have to parse different syntaxes, one version of WGSL forever. If we used versioning of lang as versioning of stdlib, seems like straightforward compromise. You have to update shaders to use new things, but backward compatibility is minimal effort. Maintaining list of builtins we support even for a dozen versions of wgsl isn't that big of a deal.
- KN: I agree with that. I have something in mind which I tried to describe, just

- KR: feature detection in the shading language is really difficult. you'd need to compile to test. In hindsight, not requiring a #version in WebGL version 1 was a mistake. I suggest we require a versioning hint in WGSL from the start
- DN: I support that as long as it doesn't need to appear at the top of the file. It only works after it is found in the file
- MM: could it be part of the shader module parameters
- JG: part of the issue, I don't know how to compile C++ file because I don't know what version you're using. I prefer the opposite, super useful to have version at the very top.
- MM: if it's not at the top where should it be?
- DN: after comments.
- MM: / JG: that's more OK.
- TR: or before you use some new feature. Before any language features. Only comments before it.
- MM: this is a pretty good direction, i think.
- JG: Kai probably wants to formalize another alternative before we go down this route.
- KN: what I have in mind doesn't necessarily work well with feature detection. In my proposal there would be no std library directive. You'd have to expose some version at the API level so apps can discover if they can use those functions.
- MM: right, that's part of option 4. A number on the GPU device saying what version it supports.
- DN: worried about monolithic view of library / language. Want smaller feature additions. 16-bit float support, raytracing.
- mM: those are extensions.
- CW: have the concept of features on a device. Could have feature that adds calculus operators to stdlib.
- KN: I'm imagining global version number, this is the WGSL version i can parse. And feature flags for extensions.
- KR: SGTM
- MM: SGTM
- RM: no strong preference, some sort of versioning is probably best path forward. Lack of it in JS has led to naming conflicts. Don't see good reason to version stdlib and language separately.
- MD: quad operations work strictly on the shader side, but are exposed on API level for all of the APIs.
- MM: that's a good design. Think we don't want 2 diff types of extensions; just "these are the extensions". TO get them, here's where you enable them, in one place. Vk has this design already. There are spirv extensions, but there are Vk extensions which enable spirv extensions.
- JG: you have to declare them in the shader too.
- TR: was about to ask whether we should have explicit declarations of features alongside the version number.
- MM: question is, are these declared automatically by browser or not.
- JG: it's easier to test. Then you have to create a device with a different version.

- CW: also, problem on stack overflow of wgsl shader pasted, doesn't work, don't know what features was compiled with.
- JG: I'd prefer extension annotations as well as version annotation in wgsl.
- MM: set of extensions in wgsl is strict subset of webgpu extensions? then authors repeating themselves.
- TR: this only impacts shader features
- MM: they're still repeating themselves.
- TR: makes it clear that wgsl requires that feature.
- JG: like if you put at the api level, "use wgsl version 4", and then just compiled your shaders, all of your shaders will have to be version 4.
- MM: I'm happy to accept this and move on to something else.
- DM: I'd like to be able to use fully qualified namespace for std functions. Will simplify tooling. Don't want to have name collisions of "refract" when compiling from spirv.
- MM: you can name things however you want.
- DM: you'll want to use debug annotations.
- DN: have this problem converting any language to any other.
- sqrt, main, etc.
- JG: minimal concern to me.
- DN: some people will like fully qualifying every time.
- MM: earlier we were going down the path of not putting these in namespace and just using versioning.
- DN: i like the idea of a namespace and "using" to pull them in. stdlib doesn't have to appear too special.
- JG: opinion, for wgsl we don't need namespaces. nice to have, but not requirement for shipping.
- MM: I'd like them for final shipment, maybe not MVP.
- RM: I think I agree namespaces aren't required, but need at least 1 of namespaces and versioning.
- DN: how about people who want higher-accuracy math or trig (or lower-accuracy)? OpenCL C has this. Don't want warts upon warts for this. Maybe an "hpc" namespace, known slower, but gives you the option.
- MM: we're in the middle of investigation of how fastmath would work.
- DN: sure, but stdlib's not so special.
- MM: point I'm trying to make is that you don't need namespaces for fastmath.
- JG: I think I agree, could be extension. Requirement for namespaces is underscores in variable names, and we have that. "using namespace" isn't possible, but diff versions of cos don't.
- DN: we didn't need it in 1976 so why now! :D
- TR: brings up another issue, if stdlib has overloading and we don't support overloading on user functions, then the stdlib is special. Can't just include your thing and pretend you've overwritten stdlib functions.
- MM: allow overloading for user functions then!
- TR: I don't think this lang should have overloading for user functions.

- MM: then how to write your own stdlib?
- JG: if you have overloading and no implicit conversions, overloading's not so bad. We know types of all expressions. All expressions are clearly typed, so the overload used should be known.
- TR: can you pass user defined types to user functions?
- JG: probably. Decays into actual type. Not a new type.
- TR: if it's a structure, decays into flattened out version of the structure? It gets complicated.
- JG: how does that work anyway?
- TR: we have to convert this to mangling.
- JG: if we can figure out the mangling without weird type conversion stuff in the compiler - type coercion and template decay are the hard parts - should be fairly straightforward for us to handle this function overloading.
- TR: the problem's user defined types. C++ mangling's defined in terms of type names. 2 things that don't match in 2 different modules but the type names match.
- JG: if definition of function is (func_name, func_types...) you can do operator overloading.
- TR: think we should discuss this in depth.

DJ: should we meet tomorrow?

- DM: want to talk about textures and samplers.
- DJ: ok, we meet tomorrow.
- CW: maybe tomorrow, first half WGSL, then API?
- DJ: sure.
-
- CW: administrivia: W3C sent the advance notice of creation of WG. Formal request for comment on charter. Sent last week. Lasts for 6 weeks. Your W3C people will look at it, but please make sure they double-check and look at the charter.
- DJ: they don't need to vote or anything now. But if there's feedback, better if it comes now.
- JG: right, this is preflighting, expect to have no surprises.
- DJ: yes, if someone in the formal vote votes against it strongly, that's a problem.
- CW: ok, please have your folks look at it regardless.
-
-
-
-
-
-
-

Interface Matching Rules (#644)

Agenda for next meeting