GPU Web 2020-09-28

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

Tentative agenda

- Multiqueue (Dzmitry, Corentin)
    - Multi-Queue Investigation #1065
    - Strawman Multi-Queue Proposal #1066
    - Multi-queue proposal with explicit transfers #1073
- Method of ensuring GPUShaderModules can contain MTLLibraries #1064 (Myles)
- Should clear values for integers lose precision, or be emulated on D3D12? #1085 (Kai)
- Copying of stencil aspect on Metal #652 (comment) (Austin/Kai)
- PR burndown
    - createBindGroup: Require a superset of the layout's bindings #1061 (Corentin)
    - Add filtered texture and sampler binding types #1076 (Dzmitry)
    - GPUColor: remove sequence overload from the union #1079 (Kai)
    - Device limits/extensions rework
        - Rename "extensions" to "features" #1097 (Kai)
        - Make GPUAdapter.extensions a setlike interface #1098 (Kai)
        - Add a limit-querying API for GPUAdapter #1100 (Kai)
- Agenda for next meeting

Attendance

- Apple
    - Dean Jackson
    - Myles C. Maxfield
- Google
    - Austin Eng
    - Brandon Jones
    - Corentin Wallez
    - Dan Sinclair
    - James Darpinian
    - Kai Ninomiya
    - Ken Russell
- Kings Distributed Systems
    - Daniel Desjardins
    - Dominic Cerisano

- Microsoft
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
- Mehmet Oguz Derin
- Michael Shannon

Administrative

- CW: Have yet to send poll. Two choices of week (because of TPAC):
    - CW: Could do 19th to 23rd of October. After that is TPAC.
    - Or, first week of November due to TPAC breakout week
    - Any concern of 19-23?
    - MM: I will be out starting the second week of November through the new year. 8 weeks.
    - CW: So October 19th to 23rd?
    - KR: There's one breakout session on the 19th about HDR on the web. It's early in the morning though so probably won't conflict.
    - CW: Okay, I'll send a poll for that week.
    - KR: there's also one meeting 9 AM Pacific of Tuesday October 20 that would be a conflict.

Multiqueue (Dzmitry, Corentin)

- Multi-Queue Investigation #1065
- Strawman Multi-Queue Proposal #1066
- Multi-queue proposal with explicit transfers #1073
- CW: Two big questions:
    - 1. Do we want Multiqueue for v1, or later, or never?
    - 2. Do we want explicit vs. implicit synchronization of queues?
- 1:
    - CW: Discussed briefly internally.
    - Lukewarm.
    - Could implement multiqueue for v1. If that's required, we won't block it. Not excited about doing it so soon.
    - MM: if we went with implicit model, would we even need to add anything in the future? Get all available queues, but everything else would work like today.
    - CW: If we go implicit, there's discovery, and there's maybe something for allowing textures to be used as readonly on multiple queues simultaneously.
    - MM: that's true for both proposals right?
    - CW: yes.

- DM: also optional explicit ownership handoff that's in the proposal, that would allow Vulkan to save an extra submit upon switching queues.
- MM: wish there were perf numbers for perf improvement if we use explicit model. Had similar discussion about implicit/explicit barriers. Asked for the same thing then. Reasonable to ask for same thing here.
- JG: more clear cut to me here. Cost of extra queue submission late, and waiting to percolate out to GPU - clear that there's overhead. Barriers - can we emit some that are optimal enough that you don't care?
- CW: assuming we want perf numbers - what would we gather? need to measure perf between optimized implicit barrier version, and…
- MM: find case where there would be difference between what auto barrier would do, and a similar program that behaved as if the programmer manually entered barriers themselves. Hopefully different programs. Then can measure perf.
- CW: would be good data point. In a sense - impl of implicit version could batch submits. Don't need to submit when you say submit, just within finite time. Could watch a bit more what happens, then does bunch of submits at end, e.g. at end of JS task. Then can see e.g. 3 submits together, and do fewer small patchup submits.
- JG: I don't think that's viable. Have to hold on to more data. Wait to see what you do, or flush commands when you give them to us? Can keep number of submits down if you hold onto whole frame, but that's a lot more latency.
- CW: difficult to come up with compelling perf argument. Subjective - maybe I'm wrong.
- DM: isn't it as simple as checking CPU time for implicit execution (?) to come in?
- MM: is one idea - if prog does it themself, they can do it better than implicitly? If so, has to be 2 different programs.
- CW: For me, the advantage of the explicit version is that it makes it crystal clear to the programmer that things will run in parallel vs the implicit one where you don't know for sure. You don't know if a code change will break the parallelism.
- MM: we discussed this before - already don't know if things will run in parallel. Willing to be reasonable here. Implicit barrier case has many chances for micro-optimizations. Up to the judgment of whoever writes the perf gathering infra - come up with implicit barrier system that would match how the 2 developing impls would do it. Fair to come up with numbers, I could say "the implicit version could do this", and you could say "ours can't".
- DC: isn't implicit model more secure? Programmer would need to know the capabilities of underlying GPU otherwise.
- JG: don't think I agree with that.
- CW: It doesn't change that. It's more a matter of queue discovery for fingerprinting.
- KN: Queue discovery impacts both what the program does regardless of whether they have implicit or explicit synchronization.

- JG: talking here about multiple explicit queues, but doing synchronization between them implicitly. Not discussing a proposal of completely implicit parallelization.
- MM: There's another answer to this. Let's imagine a GPU with two queues, and a different device with two GPUs which each have one queue. The performance behavior between the two devices is different enough that we should not expose them to look the same.
- KN: with multi-queue, 2 queues can't be on 2 separate GPUs - they assume they're using the same pool of VRAM. In the absence of additional stuff, the queues are on the same GPU./
- MM: I'm saying we'd do implicit copies in the 2-GPU case, would be prohibitively expensive.
- KN: yes, I was agreeing with you.
- DC: in multi-GPU case the underlying impl could group them. Figure out which can be done on the same GPU.
- JG: we're not talking about multi-GPU, but multi-queue.
- DC: Could implement multi gpu with multi queue
- CW: no. Need same pool of VRAM. Have to go through a lot of hoops to deal with multiple GPUs.
- MM: certainly possible, but the perf behavior would be so different that it would be unwise to expose it in this way.
- JG: would be novel research at this point, too. Interesting idea.
- MM: we should definitely investigate multi-GPU, but this level of abstraction is not the right one to do it.
- KR: Stepping back a bit, is the only resource we're talking about sharing textures (between queues)?
- JG: And buffers. Both proposals now you don't have to transfer buffers b.c. you can simultaneously use them.
- CW: Buffers don't deoptimize layout if you want to share. Textures do and need to be declared as shareable so the layout can be accessed by multiple queues.
- KR: If you issue compute work on one queue and rendering work on another that you'll wind up with real race conditions?
- CW: Implicit sync prevents data races. Explicit has state to validate no data races.
- MM: In implicit one, you get some undefined ordering. In the explicit one, you maybe get an error or it might execute in one specific order.
- CW: That's if you have two queue.submits (on the different queues) that race with one another.
- KR: Okay, with WebGL, being able to share any resources was a non-starter because it would be very intrusive and would require locking/unlocking resources. Would need to fail fast in a lot of parts of the API.
- DC: How would multi queue work if you suddenly have to go back to OpenGL?

- JG: GL fallback would either serialize, or there are different shared contexts and we handle synchronization between them. Likely serialize.
- MM: have we agreed that an OpenGL fallback is a design consideration?
- JG: only if we find something we can't do in OpenGL. We could discuss that then.
- CW: it would be nice to have WebGPU run on OpenGL ES 3.1 and D3D11, but would sidetrack the group. We have an ES backend, and some of a D3D11 backend. If we add a couple of validation rules, could keep this. After MVP, we could come up with a list of things to work on in the future.
- MM: we can discuss that when it comes up then.
- CW: Okay, now regarding performance. We can think about how we could benchmark on our side.
- MM: I think it's required. If the goal is to improve perf, we have to know how much we're improving it.
- CW: that's my point - this feature is for performance. Doom 2016 preso said, gained 25% perf by doing this. But the design doesn't have to just be for perf. Usability is imp't too. Think usability would be better with explicit version.
- MM: I think I disagree that usability's better with explicit. We could debate that a long time - and have already.
- MM: In the implicit model, you can still have a well-defined graph of operations, you just have to do it yourself and manage ordering explicitly using some other mechanism. Like your own tracking in JS.
- CW: Do you get a signal that this is actually potentially happening in parallel.
- MM: because not all devices have different HW queues, think the only signal an author can reliably use is perf numbers. Don't think they can rely on the API.
- CW: if I'm gamedev and want async compute, want to know whether it's working.
- JG: it's different. I agree with Corentin here.
- CW: imagine you have complex graph of ops. Submits A, B, C on compute queue. Want to know they're happening in parallel with rendering. Signal that things are happening in parallel would mostly solve this problem. Then can discuss late submits and their perf characteristics.
- MM: On a lot of devices, A, B, C aren't really going to happen in parallel. If they really were parallel on most devices, then I think explicit might make sense.
- DC: There's GPUCommandBuffer executionTime and timestamp queries?
- RC: It's already the case that dispatches on a single queue can happen at the same time. That's why we have UAV barriers on native APIs. I don't understand why we're over indexing on whether or not things can actually happen in parallel. For best sanity of web developers, we should have explicit transfer of things -- at least for writing.
- DM: The concurrency within the single queue is subject to your barriers. … the point of having a different queue is that while some of the resources can stop the stages on one queue, the others.. without blocking. ..
- CW: RC was talking about multiple dispatches. Technically they can and do run in parallel.

- CW: We can try to gather perf numbers, but it would take a while.
- MM: Could write on D3D to cut down the number of lines of code.
- DM: No it doesn't have that problem. The extra submit only affects Vulkan.
- CW: We could prototype something.
- AE: think the usability argument we're discussing will be more significant than the perf difference.
- MM: how do we make progress then?
- CW: could ask people who use async compute: what's your use case? Basically need to reach out to the big game engines.
- MM: If we go with the implicit model, is it possible to get better performance.. by accident?
- DM: I don't think so.
- CW: depends. If you write explicit version by serializing everything, maybe the browser could have done a better job.
- MM: if the programmer serialized things that weren't necessary?
- JG: problem then would be - automatically splitting across multiple queues.
- CW: naively - I submit things to 2 queues, because WebGPU has support for it - would I get better support with this than with the explicit version?
- DM: It could be the case when you do two submissions that happen to not actually depend on each other. When writing explicitly you might unnecessarily put a signal, transfer, wait between them.
- DC: as a user of async compute - as we are with our DCP - that's what we do. We break up the problem into tasks, send the tasks out to available devices. Little interdependency in the data. We pass the buffers, they work on them, and pass back. We look for as much parallelism as possible between the tasks.
- MM: Do you know the dep graph up front or is it really complicated and you have code to figure it out?
- DC: we have schedulers that break up the data beforehand, parcel them into small tasks that can be run on workers, in-browser or natively.
- KN: my understanding - there's no dependency between the queues.
- DC: yes - that's a single use case. There are other cases. For us it would be simple to have implicit ordering, or do multi-queue, because there isn't shared data.
- MM: Sounds like you have many (100s or 1000s of) tasks. Does each have its own resources or do you have a few large buffers and each task uperates on a chunk?
- DC: they're independent. It's broken up by the scheduler. Programmer writes JS that has the entire solution in it. Scheduler breaks up the data to be passed out. This is for distributed apps; designed to be solvable this way. Particle simulations, render farms. Pass out to workers.
- KN: one way to use multi-queue in this workload: treat each queue as a separate device. There are other ways to use multi-queue though.

- CW: Not sure how to make progress - it's a bit like the barriers discussion. Back then a lot of ISVs were telling us not to do barriers but IHVs were telling us yes to do barriers.
- JG: one useful piece of info would be: confirmation that a late patch up queue transfer would be a problem on Vulkan. If it's not a problem then we don't need it. If it is, then we need to talk about it. Someone has to write a Vulkan program with a #define that can run it 2 different ways.
- CW: doesn't address the usability discussion though.
- MM: think I agree with what Jeff just said.
- JG: if it is a perf problem, then we need the explicit thing. That's forward progress.
- DM: then still left with: do implicit with a hint, or do explicit. Doesn't fully solve the question.
- JG: right, but we won't completely solve it.
- MM: this topic is a little different than the barrier discussion we had a few years ago - barriers were needed to do any reasonable work at all. Here, only need them if you're using multiple queues. This is a smaller set of programs.
- RC: when I asked Babylon.js team about things they wanted out of WebGPU, it was this - actually, multithreading.
- KN: multithreading is an orthogonal feature.
- RC: it's related
- KN: no.
- CW: it's related in the sense that there are races - you can get these in JS with async event handlers. Multi-queue and multithreading are separate issues. We want multithreading in the first WebGPU version. Resources shallow-cloneable, etc.
- MM: do you consider multithreading a solved problem in the WebGPU design?
- CW: no.
- MM: sounds like we should talk about it soon then.
- JG: we have ideas about how to make it better.
- MM: background of what I said - I've also heard that MT is more imp't than MQ.
- CW: should we shelve MQ for now, and talk about MT?
- JG: I think it's useful, not just for async compute, but queue transfer uploads. Think it's worth talking about. But should talk about things further down the agenda. Should probably time-box this topic to e.g. half the meeting.
- CW: OK. This was a new topic, so hard to make progress.
- CW: AI: make a benchmark; reach out to people to talk about usability.

Method of ensuring GPUShaderModules can contain MTLLibraries #1064 (Myles)

- MM: discussed at most recent WGSL meeting. Agreed to discuss this at a future WebGPU / WGSL call.
- CW: that would essentially bring this to the virtual F2F.
- DM: I've gathered some new data. Would be a waste if we waited for 3 more weeks.
- MM: ok, let's discuss it.
- DM: there were concerns about numbers Myles got from perf shaders: not open-source, etc. Took some real-world measurements from DOTA 2 run. The numbers are not far from Myles'. Biggest cost today - by far - today is creating the Metal shader. Creating a library takes about the same time as creating a pipeline. If you have 2 shaders where you need libraries that'll be 2x as large as the pipeline. The data is attached to a comment in the issue.
- DM: does seem there's a big potential in trying to reuse MTLLibraries, and it won't be solved by caching.
- MM: one assertion made in this thread: this doesn't apply to D3D. I think it does. If the user writes a big WGSL program, would be great if we wrote a big HLSL program representing everything in it, compiling that to DXIL. Same design as I'm proposing.
- CW: where do you expect the gains in D3D12 if that happens?
- MM: not running HLSL -> DXIL compiler multiple times.
- CW: we can produce DXIL directly.
- KN: DXIL compilers also can't do multiple entry points simultaneously today. Have to compile multiple times.
- CW: dxc compiler only knows how to handle main() entry point. Theoretically, DXIL can do multiple entry points.
- MM: if I open new project in Visual Studio and add new HLSL file to it, does it have to have only a single entry point?
- DM: no, compiling DXIL has to have a single entry point. But you can generate DXIL manually. Even if you don't build DXIL manually, you can still generate HLSL. Takes a lot of time (for VkPortability) to go from SPIR-V to text format. Having HLSL by hand is already a good win.
- CW: The thing is, there's whole program optimization that runs in the driver anyway. Scheduling and register allocation that happens for each entrypoint which helps GPU performance a lot. In the end, when we try to generate a single MSL or HLSL code, what we're trying to win is frontend parsing time which doesn't exist in the DXIL path and doesn't exist in the SPIR-V path. Unfortunately it exists in the Metal path unless we attempt to produce Apple IR.
- DM: shouldn't your point be confirmed by numbers? How long does it take for DXIL, etc.?
- CW: I'll look at the benchmark.

- DM: do you already have the DXIL generator working?
- CW: no, but it's planned for.
- MM: if you were right that 99% of the cost is in pipeline state creation, then your argument would be totally right. But the numbers don't agree with that.
- CW: that's why I need to make a benchmark explaining what I mean.
- MM: I also think that if 99% [I think I actually meant 99.9%] of the time is in pipeline state creation, there's no need for a 2-stage compilation model - but we can cross that bridge later.
- CW: also discussion of how much state we need. Layout / preprocess method. What amount of data needs to be in there?
- MM: that's reasonable. Think it's my responsibility to enumerate that set of info. Think CW already did though.
- CW: concern: adding preprocessed version sounds fine. Just a hint. Concern with requiring add'l data at shader module creation time: 1 shader module = 1 MTLLibrary, then any element of render pipeline descriptor that affects shader has to be present at compilation time. Right now: sample mask, …, and the whole vertex descriptor.
- Might grow to be more things, with more workarounds.
- If we need to give add'l data, has to be whole RenderPipelineDescriptor, or any part of the descriptor that can be provided at that point. Impractical to provide that at shader module creation time.
- MM: you are right, more information would be needed to do this in WGSL than in Metal. One piece: vertex stage descriptor. For DispatchIndirect, we agreed it had to be possible to run a compute shader ahead of time to do validation. If CS detects out-of-bounds, can change indices, set number of vertices to 0, etc. That approach would work here. If we used it, wouldn't need to specify vertex stage descriptor in Metal shader source code.
- CW: if impls could do that, yes.
- MM: doing validation inside the shader is possible. Also possible, given previous discussions, to do validation outside shader.
- CW: yes.
- MM: an impl which wants to put a MTLLibrary inside a GPUShaderModule would do the validation outside shader source code.
- DM: would be nice if that were up to the impl to decide. We're going to do manual vertex pulling on Metal. Putting in this restriction would constrain us.
- CW: We're already doing that as well.
- MM: you could still do source code generation later (in the shader source code). Just would give impls that want to do it earlier (outside the shader source code) the possibility of doing that.
- CW: we've had nice progress on this issue; have better understanding. Will have AI to make a small benchmark to explain / see how much is in backend compilation vs. elsewhere.

Should clear values for integers lose precision, or be emulated on D3D12? #1085 (Kai)

Copying of stencil aspect on Metal #652 (comment) (Austin/Kai)

- CW: Could someone from Apple describe what happens?
- MM: will try to do this by end of coming meeting.

PR burndown

- createBindGroup: Require a superset of the layout's bindings #1061 (Corentin)
- Add filtered texture and sampler binding types #1076 (Dzmitry)
- GPUColor: remove sequence overload from the union #1079 (Kai)
- Device limits/extensions rework
    - Rename "extensions" to "features" #1097 (Kai)
    - Make GPUAdapter.extensions a setlike interface #1098 (Kai)
    - Add a limit-querying API for GPUAdapter #1100 (Kai)

Agenda for next meeting

- Should clear values for integers lose precision, or be emulated on D3D12? #1085 (Kai)
- ~~Copying of stencil aspect on Metal #652 (comment) (Austin/Kai)~~
    - Resolved during the week
- Capability-querying APIs for GPUAdapter
    - Make GPUAdapter.extensions a setlike interface #1098 (Kai)
    - Add a limit-querying API for GPUAdapter #1100 (Kai)
- Rename "extensions" to "features" #1097 (Kai)
- ShaderModules
    - MM: do you think you'll have numbers for the next meeting?
    - CW: think I can have them by then.
- More multi-queue?
- Multithreading?
    - Revive Kai's proposal for making / reviving handles?
    - Statements about needing to do this at TC39
    - KR: recommend subsuming it into this API
    - JG: agreement that we can start things here, like we did with Typed Arrays in the WebGL spec

Additional Notes:

- There's a proposal under discussion for how to allow WebXR and WebGPU to communicate. It's currently in a very early stage but based on a more mature solution for doing the same thing with WebGL. Feedback is appreciated!
- BJ: under proposals in Immersive Web group. Seems like sanest place for it - focused on WebXR's needs, but want lots of communication between both groups. Appreciate your feedback!
- CW: Three.js started implementing WebGPU stuff upstream! Going at its own pace, but they have stuff rendering already!