GPU Web 2018-05-16

Chair: Corentin
Scribe: Ken
Location: Google Hangout

Minutes from last meeting

TL;DR

- We can start sending the list of questions to partners and collect feedback
- Validation constraints on implicit barriers
    - Group agreed to not insert barriers inside passes
    - Should we prevent multiple writes to the same UAV in a compute pass?
    - Agreement to prevent read-write hazards in passes.
    - Unclear if we want to prevent write-write hazards
    - SPIR-V declares whether a UAV is read-only or readwrite. This would be matched against the bindgroup layout and can be generated from HLSL source easily (RWStructuredBuffer vs. StructuredBuffer)
    - Concern additional restrictions will mean we start and end too many passes.
    - We should support combinations of ready only states
    -

Tentative agenda

- Review questions for partners
- Define validation constraints on implicit barriers
- C header / test suite discussion.
- Agenda for next meeting

Attendance

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

    - Ken Russell
    - Victor Miura
- Microsoft
    - Chas Boyd
    - Rafael Cintron
- Mozilla
    - Dzmitry Malyshau
    - Jeff Gilbert
    - Markus Siglreithmaier
- Elviss Strazdiņš
- Joshua Groves

Review questions for partners

- Jeff Gilbert's list of questions:
    - https://docs.google.com/document/d/1yZ631Ww4ko7F22IObiLia8XiHqAHB0cptPB7OkDOnTE/edit
    - Most of the questions we want answered are for ISVs. But would be good to have IHVs look through them and get meta-feedback ("why are you asking this? this is terrible") too.
- DM: who are the ISVs?
    - JG: Engine teams. Unity, Epic, Roblox, …
- CW / DJ: need to make sure who's on the list.
    - Definitely want Unity and Epic. Targeting the web today.
    - Roblox.
    - Valve. Has done work with MoltenVk.
    - Mr. doob
    - Sketchfab
    - Cesium
    - Maps
    - PlayCanvas
- Discussion on how we should gather feedback on the questions.
- JG: They can make a copy of the doc and answer inline
- KN: Or we copy paste in an email and have discussion on mailing list.
- CW: sending through email through partner mailing list sounds good, to get people to join the list

Define validation constraints on implicit barriers

- CW: at F2F we agreed on implicit memory synchronization and barriers. Want to avoid data races, so what are constraints? WebGL, for example: no rendering feedback loops. How do we explain this concept in WebGPU?

- MM: group agreed: we should not be inserting barriers in the middle of passes. Is that right?
    - CW: only one not 100% sure about: copy pass, if any. But think it is the right call.
- MM: did some research about barriers as they exist today. Metal somewhat irrelevant. D3D has aliasing barriers; not relevant. UAV barriers: imp't for R/W hazards. Transition barriers: imp't for e.g. sampling from texture and using as render target (feedback loop case).
    - Pretty different concepts.
    - UAV barriers: need some way to determine UAV will be read from / written to in dispatch / draw call. Then can say, once resource has been written to, it can't be read from in the same pass.
- CW: if you have, say, compute pass, want multiple dispatches to act on the same UAV. But don't want to end the pass necessarily. There are other cases where you have multiple dispatches acting on the same UAV. Think we want to preserve this kind of parallelism.
    - But if you have a writable UAV, should not be able to use it as a vertex buffer.
- MM: OK. So if we consider UAVs, agree, and Metal has distinction between compute and render passes, and will insert barriers between these kinds of passes. WebGPU could have similar implicit barriers.
- DM: thought we would not insert barriers between dispatches of a single pass. Thought barriers were on the sides of the passes. Can treat it like multiple vertex invocations of the same primitive.
- KN: think I'd prefer that behavior between render and compute passes, but don't know why metal chose this.
- MM: it's because of examples like CW pointed out. People drawing to same destination – draw calls are independent, so don't need to do state tracking for that. But for compute passes, adjacent dispatches often will be working on the same memory.
- CW: so people using UAVs in render passes know what they're doing more than in general.
    - More people want to do pure compute passes than compute and render work in the same pass.
- DM: what about transfer ops in Metal? Flushes in BlitEncoder?
- MM: not sure.
- CW: think sync should be between passes. Work between passes should be parallel.
- MM: sounds like advocating for both sides. What do you suggest?
- CW: gut feeling: we should prevent read/write hazards. Just think UAV sync might be too expensive for key use case of compute. Someone (Arseny?) got ~30% improvement by using UAV barriers.
    - DM: don't remember
- CW: preventing two consecutive dispatches to act on the same writable UAV will leave too much perf on the table for compute cases.
    - DM: now remember. Want Metal to have these small tweaks to avoid sync between adjacent compute dispatches.

    - MM: so Metal injects too many barriers?
    - CW: yes. Have to wait for all compute dispatches to finish, then flush.
    - CW: this is my only concern. Agree with everything else you said.
    - CW: maybe writable shader storage buffers can be used multiple times in compute passes. Can't be used for anything else.
- DM: if we restrict usage to be constant across the pass, that would limit things for Metal. In Metal you don't say which usage you want. If used as writable UAV in first pass, can't use it anywhere else in the same pass. Would enforce same rule as Metal but would avoid flushing caches.
    - Think if we put this restriction in place, then we should not introduce any synchronization between calls within a pass.
    - If people want to use results of previous call, would require a different usage.
    - CW: not really. Writable shader storage is still readable.
    - DM: people could still shoot themselves in the foot. But if their shader only reads something then we can assume they only use the read usage.
        - We have this usage in SPIR-V. Know whether the UAV is writable or readable only.
    - CW: want to talk about splitting R/O and R/W shader storage.
- CW: seems like harder question.
- MM: think keep it simple for now (disallow more cases), and if we find we're leaving too much perf on the table then we turn it back up.
- CW: OK, so writable resources can only be used once in a pass.
- MM: yes. But don't feel very strongly about it.
- CW: what about if you have a texture, want to write to it multiple times?
    - MM: make different passes.
    - DM: what would happen with render passes? e.g. accumulation buffer? Wouldn't synchronize between those in the render pass anyway.
    - MM: (1) Use UAVs, write directly to resource. But if in render pass, (2) attach another target to framebuffer. Well-defined, well-ordered.
- CW: can't fetch from framebuffer in general case.
    - MM: true in general case, but for accumulation buffer it works.
- CW: we didn't think about the write usage to be restricted to writing a single time.
    - Copies of buffers to texture: Vulkan/D3D, these are racy.
- MM: try to be as safe as possible, and if too slow, turn it back up.
- JG: centering on: doing most memory barriers ourselves, but allow you to choose to rearrange them. (?) Discussing how we want to restrict things to let people do what they want without shooting themselves in the foot. Part of answer might be, do best effort to inject barriers, but if app thinks it can do better, it can specify where it wants the barriers (as long as their placement makes sense). Instead of doing too many heuristics.
- KN: so on top of the implicit system, you could insert transitions etc.?
- JG: yes, if you want to transition from writable to readable, insert this yourself.
- DM: problem is, we can't say we don't want a transition. If we do want it, end current pass and start a new one.

- JG: concerned we'll do that all the time because our API's too restrictive.
- KN: could be bad; other reasons you want larger render passes (LoadOps).
- DM: supposed to not transition inside render passes anyway.
- JG: meant memory barriers, not transitions.
- CW: right now: deciding do we enforce write/write data races? These are more pervasive. Once shader starts using shader storage (even frag shader), can only use that once in a render pass. Every time you want to reuse it, have to restart a render pass.
- MM: you can have a write/write hazard in a single draw / dispatch. Not trying to eliminate hazards; trying to find the safety / speed line.
- CW: from the other direction: do we agree we want to avoid R/W hazards, except maybe on UAVs? So if you write to an image in a copy pass, you can't read from it until the next pass. Maybe, can write to it twice.
- MM: you're talking about transitions, distinct to UAVs. Agree on transitions more. Think right model is: Vulkan and D3D have descriptions of what states a transition can be from and to. Union of them; in single render pass, can only use a resource as one of those.
- CW: what about combination of read-only states? Pretty sure that's easy to support.
- MM: ok with me.
- KN: you mean "uniform buffer" and "array buffer"?
- CW: yes.
- KN: what if it causes image layout transitions?
- CW: in Vulkan, always have the "general" image layout you can use.
- JG: if you have a texture, and an array buffer, not the same access pattern.
- CW: discussion of Vulkan particulars of texture layouts
- JG: think it's the other way around but not that important
- MM: agree, shouldn't take the union of every state. Group things into fewer states to make it easier for people to use this library.
- CW: so, in this API, can use a resource either as one of the writable states, or a combination of the readable states.
- MM: sounds good
- CW: does this apply to UAVs? Can you write twice in the same pass?
- DM: strongly believe we should be able to use it twice in the same pass. If we don't, people will work around the API by making mega-shaders and combining multiple draw calls into one. We would encourage badly written code that won't run efficiently.
- KN: agree.
- CW: we haven't thought about it before but at this point think we might want to allow write/write hazards. Don't want to cut everything and start over.
- KN: would also give us R/W hazards?
- CW: no: have one ImageView, doing multiple writes to it, but you can't copy from that image in the same pass. Bit more complicated.
- MM: starting to get into territory of, from shader, determining whether resource is read from, written to, or both.

- CW: in bindgroup: define how resource is used. Writable ones: shader storage buffers, UAVs for buffers, UAVs for images. Other way is as attachments. Think inside bindgroup layout, should be able to distinguish between “using as UAV, R/O” and “using as UAV, R/W”. Then during pipeline compilation, require that resource is R/O if shader uses it as R/O. No runtime instrumentation of shader.
- MM: (starting to talk about shading languages again) agree that most shaders in the world are HLSL, which doesn’t have “const” keyword. Who should be generating that constant information? Web site author’s hands? Browser’s hands?
- CW: need to look again, but in group signatures, you can declare some stuff read-only in D3D12. Have shader, combine with root signature. Think you can say R/O in that signature.
- MM: so you’re saying, the concept of R/O should be done at the API level and not the shading language?
- CW: no, just explaining D3D12 behavior today.
- CB: in D3D12 can handle some of this in the shader itself, like defining register for sampler.
- DM: why are we talking about root signature? There are R/O vs R/W buffers and textures. Very clear how to translate to SPIR-V.
- CW: that’s right.
- MM: then next quetsion: web author said RW buffer, but never writes to it. Should we treat it as writable?
- CW / DM / KN: yes.
- CW: example, writable buffer for CUDA printfs. Only written if you call printf. Still OK.
- JG: GLSL spec also uses static use. As long as there’s text in the shader that writes to a thing, it’s still considered written.
- CW: in SPIR-V you say this is a R/O buffer. During shader validation, writes would be a validation error.
- JG: you could say, if not R/O, consider it writable.
- CW: think we made progress. Have some unanswered questions about writing multiple times to the same resource in the same pass.
- MM: not willing to put up a fight about it; sounds like we’re all in agreement.
- CW: RW hazards? Not sure whether we should allow them myself. Needs more research / discussion.
- JG: leave it in open questions.
- RC: if from shader we knew writable vs. const, could we put correct barriers in between dispatch calls? Thought dispatches could be rearranged. If we could know in shader if something’s ever read from / written to, do we think we could put barriers between calls in the same pass?
- CW: it’s like you do two dispatch calls wtihout a barrier. It’s arbitrary which one goes first.
- DM: R/W barriers or W/W barriers?
- RC: lots of reordering possible.
- CW: in compute pass, dispatches can be rearranged. But, can a writable resource be used only once in a compute pass, and allow racy dispatches?

    - Third possibility: allow explicit UAV barrier in compute passes. Without flushing the whole pass.
- KN: compute passes are not a concept in most of the APIs. Too destructive to do BeginComputePass / EndComputePass?
- CW: would be more work on some APIs.
- MM: curious about what RC just said. One of goals is to insert barriers where needed. Can only be inserted at compute pass boundaries. So in D3D12, there would be no barriers in compute passes, and driver could reorder everything? This seems difficult to use. In such a system, developer would have to create lots of passes.
- RC: today in D3D12, there's no "pass". Make a Queue, it's compute, stuff in commands. With no barriers, dispatch/dispatch, it'll let you do that.
- MM: so passes in D3D impl, indication from programmer "do your sync here". In example where programmer has bunch of dispatches and needs barriers, programmer would create lots of passes?
- RC: yes. Have to say, this pass has these outputs.
- CB: if in WebGPU we say we have to preserve ordering by default, then a higher layer could insert these barriers for the programmer.
- CW: 3 solutions:
    - 1. Do like Metal, insert UAV barriers auto inside compute passes.
    - 2. Don't insert UAV barriers auto, but have an explicit "compute pass UAV barrier command".
    - 3. If you want to reuse a UAV you have to end/begin compute passes.
- RC: how's 3 diff from 1?
- CW: WebGPU impl ends and begins compute passes. In (3), the app does it.
- DM: when thinking about this: please be consistent within passes. Sync in render pass should be similar to compute and transfer passes.
- KN: agree with goal but Metal is different.
- CW: UAVs are different too. Could break tiling.
- KN: for (3) the impl of End/BeginComputePass on Metal doesn't have to end and begin compute pass on Metal because of impl barriers. Not incurring background cost on the underlying API. On D3D, would insert a barrier. Just incurring cost of doing it implicitly.

C header / test suite discussion.

- CW: put up C header showing how we could translate some IDL concepts to C.
- CW: When talking about this, got too far into it. Should have been: what language is the test suite written as? JS or WASM?
    - We think WebAssembly. But this is a question that had to be asked before talking about the C header.
    - Still useful to have a C header. But it's not a matter of standardizing it.
- MM: people should comment on the mailing list?
- CW: yes.

Agenda for next meeting

- JG: Push out the IHV/ISV questions.
- DM: should have separate IHV document. Don't think we have an IHV mailing list.
- CW: would be same.
- DM: so we send all questions to the same people, and some know IHV side more?
- CW: yes. Their answers might be more private b/c of being performance sensitive.
- CW: Khronos / D3D / Metal folks should reach out through those channels.
- CW: do we talk about barriers again?
    - MM: think in a week we'll have more thoughts.
- CW: think about sub-resources. Can you render to a mip level while sampling from another? How do we validate that?