GPU Web 2017-07-26

Chair: Dean
Scribe: Ken
Location: Google Hangout

Minutes from last meeting

Tentative agenda

- Administrative stuff (if any)
- Individual design and prototype status
- More on render passes.
- Memory barriers
- Agenda for next meeting

Attendance

- Dean Jackson (Apple)
- Julien Chaintron (Apple)
- Myles C. Maxfield (Apple)
- Austin Eng (Google)
- Corentin Wallez (Google)
- Kai Ninomiya (Google)
- Ken Russell (Google)
- Daniel Johnston (Intel)
- Ben Constable (Microsoft)
- Rafael Cintron (Microsoft)
- Dzmitry Malyshau (Mozilla)
- Fernando Serrano (Mozilla)
- Jeff Gilbert (Mozilla)
- Alex Kluge (Vizit Solutions)
- Doug Twilleager (ZSpace)
- Elviss Strazdiņš
- Tyler Larson

Administrative items

- Details on the September meeting: https://lists.w3.org/Archives/Member/internal-gpu/2017Jul/0004.html

- Reply to Corentin's email if you're going to attend
- CW: Deadline for replying is September 3rd.
- Software license update
    - DJ: told by Apple's legal team; meeting next week with Microsoft, Mozilla and Google. Seems to be going well

Individual design and prototype status

- Google
    - CW: Kai has been working on making actual SwapChains in NXT
        - Can later try to port this on the web front and render to multiple canvases with the same device
        - Austin has been adding a bunch of fixed function state: primitive topology, … .
        - Trying to add essential features for a demo or game-like thing.
    - DJ: have you done this for WebGL? ImageBitmap, etc.?
        - Yes Justin Novosad has been working on this:
            - Either render on offscreen canvas, get an imageBitmap and transfer it.
            - Either give control to the offscreen canvas and render to it. Path is currently broken.
        - Not sure what the intended model is for rendering to multiple GPUWeb canvases.
        - In other words, do we need OffscreenCanvas at all for GPUWeb if we have SwapChains that can be targeted?
        - CW: Discuss at F2F?
            - Can ask for a context that's an "NXT Surface" for a canvas and get a swapchain object from it.
        - KN: create an NXT device out of thin air, and get SwapChains for multiple canvases from it.
            - Make it so SwapChains can be transferred to workers and used there. Should be like OffscreenCanvas.
        - DJ: sounds pretty similar to how Apple's original prototype worked. Made it look like WebGL way of setting things up (fetch context, etc.), and then ask for next texture to render into. Felt a bit weird.
- Mozilla
    - DM: lot of internal work in the graphical abstraction layer, but not a lot of new features or milestones to report
- Microsoft
    - RC: no code contributions yet (to NXT, etc.)
- Apple
    - DJ: No news, but started internal discussions on shading languages, trying to get info on the sampleMask issue.

CW: JohnK will be discussing shading languages with us in 2 weeks (August 9), so will get info straight from the source.

Render passes

https://github.com/gpuweb/gpuweb/issues/23

- DJ: mostly had consensus on this last week
    - What was left? Inheritance? Subpasses?
- CW: people mostly agreed on having “multi-pass” render passes
    - Mostly to see if people had more to say after investigating internally
    - Tentatively: let’s do multi-pass render passes.
    - DJ: fine with Apple.
    - RC: did look at Corentin’s presentation. As long as it’s possible to implement efficiently on D3D12, it’s fine.
    - MM: conceptually render passes are a good idea. Were discussing which pieces of state should be part of a render pass.
- DJ: what are next steps?
    - Should Corentin or someone define an API? Or move on with the assumption that we have an agreement of the overall shape of this part?
- KR / MM: seems too early to define an API
- DM: agree, also tied in to memory barriers
- CW: maybe we can make a comprehensive list of things we want to talk about before looking at the shape of the API
- DJ: Could do this on the wiki/github
- MM: an issue per issue, then close them?
- DM: Github milestones?
- KR: spreadsheet?
- DJ: volunteers?
    - MM: I can start!

Pipeline state objects

- DJ: do we have complete consensus on pipeline state objects?
- CW: still details like sampleMask
- MM: probably should include depth/stencil state
    - Needs to be intersection of the three APIs. Because Metal’s doesn’t include this, but WebGPU’s probably should since the other APIs do have it included.
- CW: whatever makes backing APIs do no extra work at run time.
    - All the things that are given at pipeline creation state time in the underlying APIs should be part of the pipeline state creation in WebGPU.
    - Except for Vulkan’s scissor state which is insane.

- DJ: Metal team said they had little feedback from developers about needing sampleMask. Think it is fine to leave it out in version one.
    - CW: Sounds fine to leave it out, can add it easily later.
- DJ: Trying to get contacts of developers from the Metal team so we can talk with them.
- DM: In Vulkan, some pipeline state like separate blend state is gated on some features in Vulkan, and might not be available on all hardware. Would ofc require some Vulkan features to stay sane.
- DJ: any hardware that doesn't support it?
- DM: Vulkan has a very rich set of flags for device features
- CW: can look at the Vulkan hardware database and see if a feature is available universally, ignoring obsolete hardware
    - For example, if ARM doesn't support a feature, definitely want to support it on ARM
    - Agreement from Apple, Mozilla

Memory barriers

Continuation from last week's meeting

- CW: were talking about implicit vs. explicit barriers
    - Implicit barriers make it easier on the app, give more leeway to the implementation to optimize
    - Explicit: give more control to the application to batch memory barriers together & have less CPU overhead
- MM: why do they have less CPU overhead?
- CW: if you have implicit ones and want to batch them, then if you're the driver, it's somewhat easy. You have the command buffer in memory and can go backward, say "want this memory barrier to be bigger".
    - But if you're doing this in an encoding API, you have to walk backwards in the command buffer and then walk back forward.
    - Have looked at the Intel and AMD open-source Vulkan drivers.
    - Barriers in these drivers are just a couple of opcodes. Can edit if you submitted them in the past.
    - But you can't do this with any of the backing APIs.
- MM: this is an argument for why the driver's better at this than the browser, not that the browser's better at this than the author.
- JG: automated batching is not an easy problem.
- CW: Metal can do this more easily
    - The argument "Metal does this, so it's possible" is not a good argument because we don't have as much control over this as the driver does.
- MM: sounds like somebody would need to try it. Having a fundamental disagreement about how difficult this is.

- CW: what do we want to know exactly? Build two prototypes, one with implicit and one with explicit transitions?
- DJ: it's difficult to ask a browser to implement implicit transitions and batching to prove that it works…
- MM: at some point you have to check all the resources are in the right state, as late as possible. WHy not just issue the barriers there?
- CW: barriers are "flush the whole L1 cache", for example. Want to coalesce them.
- MM: coalescing would happen as late as possible. Checks you would do are at the same time that the coalescing would occur.
- CW: disagree. Example: doing some dynamic mesh with compute. Have writeable buffers. Then start using as vertex buffers. Barriers have to be "inside" the command buffer, not at encoder start/end.
    - Analysis sees a draw, looks at needed resources. Sees a vertex buffer. Issues a barrier.
    - It sees many such draw calls. Can't do this in the past.
- MM: you have a command buffer. Add as a writeable attachment. Then issue a draw. Are these not the times you would both check, and issue a barrier?
- CW: e.g. using 3 writeable vertex buffers
    - Use buffer 1: have to issue a barrier.
    - Don't know the future: or have to do an analysis pass first, then encode.
- KN:
    - (W)rite (T)ransition (R)ead resource 1/2/3
    - Batched: W1 W2 W3 T1 T2 T3 R1 R2 R3
    - Just-in-time: W1 W2 W3 T1 R1 T2 R2 T3 R3
- MM: would have to do this analysis at commit time
    - Don't think this would be as complicated as CW says
- BC: D3D has had runtimes which did this both implicitly (<= D3D11) and explicitly (D3D12)
    - The code that does this implicitly is large and complex, in particular handling multithreading
    - Would have to have basically the same system for Vulkan and D3D12
    - Easy to do for the simplistic case. But Corentin and others have already pointed out where you'd need a complex system to make this correct. Not over-adding barriers, and never missing one.
    - D3D has experience doing this: debug layer. This is a significant chunk of work, and don't have to care about performance.
    - Alternate proposal of making the barriers explicit would only require Apple to no-op them.
- DJ: what if the barriers were forgotten in Corentin's example?
    - CW: validation error.
- DJ: what about seeing when the user should have put in a barrier but didn't?
- CW: decided to validate resource usages / memory barriers.

    - Believe it doesn't add much cost do some other feature -- like destroying textures but still have them around.
    - On every command buffer, have to check liveness of textures. At the same time, on the same cache line, can validate states.
- KN: have to insert / execute some code to do this validation.
    - Think it's roughly the same amount of time as the just-in-time usage transition.
    - As we've learned from the Intel and AMD driver, this isn't efficient. Have to coalesce. Coalescing is expensive.
- DJ: is the coalescing benefit in Corentin's example that you would only insert a barrier for buffers 1, 2, 3 since you know you're done with them?
    - CW: yes. Rather than: barrier 1, draw 1, barrier 2, draw 2, barrier 3, draw 3.
- DJ: b/c you're doing dependency tracking anyway, have to track that you've put the barrier in for B1, can't you just do them all?
    - CW: no! you don't know how they're going to be used in the future. And we shouldn't guess what the hardware does.
- BC: heard people mentioning cache flushes, etc. But another thing that GPUs can do is change the compression format based on usage. Can see lots of thrashing on some GPUs. On all hardware Ben knows about, these transitions are needed.
- DM: like to expand on explicit case. Not just barriers. Also: putting things at the end of the usage as a resource, so that you give the hardware time to do the transition.
    - AMD: use split barriers. Barrier start -> barrier end. Do decompression, then flush caches at appropriate time. These sorts of optimizations are not easy to automate. But: still easy to validate that the transitions are in place.
    - Validation becomes increasingly easier than automatically putting transitions in place.
- DJ: leaning toward what Ben suggested, which is to go forward with explicit barriers, and Metal will no-op them. Would still like to hear from major developers using Metal and other APIs what performance hints they're hitting with Metal doing it behind the scenes. Dean assumes that Metal may have some code for this but that it'll never be as optimal as what the developer could have done themselves.
- MM: it's not a no-op. The validation still has to occur. (Agreement)
- CW: there's CPU overhead on Metal, and there's mental overhead for the developer.
- DJ: Metal wouldn't have to do anything more than any other implementation, just not actually call the barrier.
- MM: DM just gave an example of a particular graphics card. Requiring the web author to know the optimization possibilities of all the cards out there is not reasonable. Think it would be better for the browser to do a "good enough" job.
- DJ: turning it around: it would be terrible if the developer optimized for one piece of hardware and it behaved poorly on other hardware.
- BC: have seen developers port things to D3D12. Have seen things they do to make things performant pay off on multiple kinds of hardware. Don't see things behave radically differently on Intel, AMD, NVIDIA, etc. In general are able to treat all the hardware types roughly the same.

    - Even UMA vs. non-UMA devices: there's no way to do this in a performant way with the same set of code.
    - You'll have to test on: tiled/non-tiled, discrete/UMA.
    - You have to do this with modern graphics implementations.
    - This is why Vulkan/D3D12 exposed the knobs.
- CW: something written with WebGPU should be portable. But we're talking about performance portability right here.
- MM: seems unreasonable to ask every web author to test on 5 different platforms.
- CW: look at WebGL right now. NYTimes cares about it running; not the last 5% of performance.
- MM: if we only care about a few percent, the browser can just do it automatically.
- CW: one case is using Three.js, and you don't care about the last ounce of performance. Compare to Unity/Unreal, which would want coalescing, etc. If you do implicit barriers, you prevent the engine authors from optimizing.
- KR: sounds a lot like compiler optimizations like software pipelining, inserting prefetches, etc. It's hard. Seems a lot easier to just validate transitions the developer inserted earlier.
- BC: barriers *are* some of the harder things to deal with.
- CW: also, this is an online problem -- affects every command buffer submission.
- DJ: have to adjourn discussion now -- continue August 9.

Agenda for next meeting

- Start discussion on shaders - August 9th
- F2F: Swap chain v offscreen canvas