GPU Web 2019-11-18

Chair: Corentin
Scribe: Dean
Location: Google Meet

TL;DR

#501 array buffer creation benchmark & Immediate uploads and friends #426, #481, #491

- Benchmark in Firefox shows ArrayBuffers are cheap, but still concerns about GC.
- Discussion about the need for many small disjoint copies and how they map to buffer mapping and GPUUploadQueue
- Mostly agreement that GPUUploadQueue is a good direction, but some discussion about racy uploads.

#495 (Re:) Expose limits on the adapter

- Should discuss this more in particular around fingerprinting and feature levels.

Tentative agenda

- Immediate uploads and friends #426, #481, #491
- Agenda for next meeting

Attendance

- Apple
  - Justin Fan
  - Myles C. Maxfield
- Google
  - Austin Eng
  - Corentin Wallez
  - James Darpinian
  - Shrek Shao
  - Ryan Harrisson
- Intel
  - Yunchao He
- Microsoft
  - Damyan Pepper
  - Rafael Cintron
- Mozilla
  - Dzmitry Malyshau

    - Jeff Gilbert

#501 array buffer creation benchmark & Immediate uploads and friends #426, #481, #491

- DM: Can create a millilon to a million and half ArrayBuffers per second. Talked with our JS team and there's more we can optimize with creation. ArrayBuffers are important for WASM as well so I think optimizing their creation is a good path to go.
- CW: So can we treat these as free?
- DM: Maybe? More than previously.
- JG: Once concern about treating them as completely free is the garbage collection. I don't have anything written down, but I have experience with GC pauses being an issue. It's not the creation time that's an issue -- it's that GC takes a random amount of frame budget from each frame. In Gecko it can take most of the frame. One of the big reasons WASM took off so well is that it took away garbage collection. Even thin wrappers of typed array views of the right size and type for WebGL created undesirable GC overhead. That's why we added WebGL 2 entrypoints that didn't need new objects.
- JG: What you would need to change about the benchmark is to measure FPS with synthetic load and watch for dropped frames. It's unfortunate but true for these tiny wrapper objects.
- CW: Should we worry about things like the RenderPassEncoder / CommandEncoders?
- MM: Why did you write this benchmark? What's the background?
- DM: One of CW's concerns with the Upload PR is that returning an ArrayBuffer for every upload would be too slow.
- MM: I guess the question is how much garbage are we expecting? 1 frame isn't so bad. If there is an alternative, then applications with a ton of data transfers can use the alternative.
- CW: ~~As a data point: I believe Babylon does at least one upload per object. Could be optimized, but right now they have two command encoders open. On one they encode copies and do setSubData a bunch of times to put stuff into a staging buffer.~~ Nevermind - confusing versions of code.
- CW: Discussion is because UploadPass creates garbage -- potentially as much garbage as there are draw calls. Whereas RenderPassEncoder and CommandEncoders are bigger granularity and not so bad. That's why I think we should keep Buffer mapping as it is now and have GPUUploadPass in addition to it.
- DM: Don't think that would justify keeping the existing mapping. We don't have numerical evidence to suggest this particular piece of the API is going to be slow. We have some anecdotal evidence from previous WebGL experience which I would love to review. The users are always able to create one large upload and do multiple copies. It doesn't force many array objects, that's one way to use it; not the only way.
- MM: In current IDL, GPUMappedBuffer has an ArrayBuffer inside. People will be making multiple of these so I don't think there's additional garbage.

- CW: With GPUUploadPass if you want to scatter writes to multiple buffers, you need to have multiple ArrayBuffers. If you do ring-buffer-of-buffers you can have a single upload and scatter.
- RC: Can't you still do that with the UploadPass if you pass the buffer..? as the userdata.
- CW: Sure, that's more copies though. Because you have the initial copy into the array buffer, copy into staging, and copy into resource. Whereas if you can map, you remove the copy to the staging, because what you get is directly staging data (or shmem).
- RC: IF you don't use userdata, couldn't the browser give you back the same ArrayBuffer every time in the version where the browser gives you the ArrayBuffer.
- CW: I don't think we would spec that -- it would be a new JS object every time. Shouldn't return *maybe* the same buffer with the same extended properties, etc.
- DM: One alternative would be to request a large 1MB chunk upload into the buffer and fill in the ArrayBuffer as you go through the objects?
- MM: Are you describing separating the backing store for the data from the set of copy commands? As in, have a big blob of data, and separately record "copy these bytes into buffer X, and these other bytes into buffer Y, etc."?
- DM: Asking if a more direct approach would make sense. Request a large upload and get one ArrayBuffer for it all.
- CW: With that, you still can't have your initial 64Kb and split it so the initial copy goes to a different place.
- DM: YEs you can't split it, but other than that there's no extra copies. If you expect to upload 1MB of data, then you choose a chunk size that's like.. 100k so an extra 50k doesn't make much difference.
- MM: That works pretty well if all the data you're writing to is contiguous. Which is probably usually? true. But you probably can't have a really big ArrayBuffer and write/read little bits into it.
- CW: Current buffer mapping allows you to copy disjoint things, whereas the GPUUploadPass is contiguous.
- DM: Yes, I see. We were talking about the linearly allocated use-case where you use dynamic offsets to point to it. For non-linear data what is the use case? When you do scattered updates in actual APIs, you heavily rely on persistent mapping.
- CW: You certainly want something like this for LOD streaming, etc. Every frame, there's dozens, if not hundreds, of copies to update and upload tiny bits of textures or buffers.
- MM: Are we talking about textures or buffers?
- CW: UploadPass has both. Buffer mapping allows you to update both textures and buffers as well.
- RC: Buffer mapping neuters so in the existing case you still need to return a new ArrayBuffer every time. Does that mean both proposals are the same in terms of GC?
- CW: No, because with one mapped buffer of 1MB, you can copy to as many different resources as you want with the optimal number of copies. With GPUUploadPass, if you want to write to multiple buffers you need multiple ArrayBuffers.
- MM: Why can't you do that with DM's proposal?

- CW: In current buffer mapping proposal, you have many small pieces of data. You map a large buffer and then submit a bunch of copy large-buffer to small-buffer. With DM's proposal you have to do a bunch of calls to get a bunch of small staging buffers. If you don't want to have a bunch of small staging buffers, you have to ask for a large staging buffer and then submit the copies. You normally have one extra copy there.
- DM: I don't think there's an extra copy there. You create an ArrayBuffer for staging, fill it out, and issue the same copies.
- CW: You're not copying directly from staging area into scattered resources.
- DM: I see what you mean. If you try to copy into a buffer not in use by the GPU, it will try to give you the direct mapping immediately.
- CW: We discussed this, and I don't believe we want to do this sort of optimization to know which segments of a buffer are in use by the GPU at any point in time. Secondly, it's an optimization that's invisible to the app.
- DM: THis is the whole buffer -- not segment tracking
- MM: Don't you need to do this tracking because you need to know when to destroy?
- CW: No, we just wait for all GPU ops.
- DM: Example with textures is N/A. Most of the time you copy from large staging buffer into multiple texture pieces -- unless you're on UMA and lots of assumptions about swizzling, usually you're copying from a buffer anyway.
- CW: Seems like just a different way to do buffer mapping that's on the queue -- but it forces the application to know if the buffer is in use by the GPU to get the optimal path.
- MM: Right, I was about to say that. Hard for applications to know if they're going to get on the fast path.
- JG: In WebGL we issue warnings when you don't hit the fast path.
- RC: Can we make the return value be nullable? If it's in use, return to me null, and then the application can make another buffer.
- MM: Pretty important because browsers that are double buffered will have a different lifetime from those that are triple buffered. So it's going to be different. But also having applications not know when they're on the slow path isn't good.
- CW: to @JG hopefully we can make an API that is the fast path and not rely on warnings.
- MM: But a developer that only tests on double buffered will never see triple buffered browser warnings.
- JG: Inherently racy. In WebGL we issue warnings when a buffer *may still be in flight*. We actually check that you waited on the fence before trying to readback. If you did writes after a fence, then we generate a warning. Statically know you didn't observe the writes via a fence.
- CW: Popping the stack. Seems like the question is: does GPUUploadPass replace buffer mapping? I don't hear strong concerns about the UploadPass itself (aside from the exact shape of the API) but not against the concept. Does everyone agree they would be happy with the direction with/without buffer mapping? or both?

- MM: Reason I opened the issue is helping authors get uploading data right. In order to judge proposal on that criteria, I'd like to see an example of what uploading.. 16 bytes per frame would look like. If it's easy to get right, then looks good to me.
- CW: In current shape, it's sort of similar to setSubData -- except on different object, etc. I think it has been very natural to people to use it. Didn't really hear concerns about setSubData at all.
- JG: Won't hear complaints about it because we're not at the stage where we see minor complaints yet. Also for a while, you didn't have a way to upload other than setSubData. When it's good-enough for early prototypes -- that's one thing. But the bar is higher for good-enough-to-ship. ISVs will run into more issues. Hard to make good benchmarks for everything. That's why we need to design a good architecture.
- DM: Think the one concern is from MM is that it's hard to hit the fast path all the time. Don't think that an app will hit the fast on a double buffered browser but not a triple buffered browser. If they care, they will use fences.
- MM: Fair point. With this proposal, when is it valid to call the upload buffer? Can we go over that?
- DM: At any point.
- MM: Does it get enqueued at a particular point?
- DM: Queue op so it gets ordered with respect to all other ops on the queue.
- CW: Okay, so everyone is happy with the direction -- does it replace buffer mapping?
- JG: I would still like to not go this direction. It still seems to me like the direction of upload heuristics which I think we should be aiming to avoid.
- CW: Can you and DM take an AI to talk about this this week?
- RC: So JG, with mapWriteAsync, you think that will hit the fastest path?
- JG: ..
- RC: So you think the best one is the one we don't have yet?
- JG: I think we got stuck on the racy SharedArrayBuffer. If we unlatch that concern, then it becomes simple.
- CW: That's non portable. Applications will break.
- JG: We ship this already.
- CW: The one way to upload data should not force people to walk around with a footgun.
- JG: It doesn't. I'll make a proposal. I think that generally the fears expressed here about how hard it is to do this properly are overblown. I'll make a concrete proposal to discuss this.
- CW: Okay.
- MM: look forward to it!
- RC: Will it allow raciness? Will it allow a way to follow certain rules and not be racy?
- JG: Yes, yes.
- DM: If Jeff's suggestion works out, we'll be able to scrap everything and just take it. But I think it's far and we need a bunch of research first. I would prefer the simple upload pass.
- MM: In addition to buffer mapping?
- DM: Just the upload. I don't think buffer mapping gives us much in addition.

- DM: Thanks to all the reviews and comments. I agree that the atomic queue ops are better than the confusing pass concept. ANd you get separate objects and can tell WebGPU you're done with it. Would allow more convenient multithreaded usage of the queue.
- CW: So I understand:
  queue.uploadToBuffer -> ArrayBuffer
  What's the lifetime of the ArrayBuffer?
- DM: Until the next queue.submit -- or if the user detaches it explicitly.
- JG: Sounds very similar to just being able to map a subrange with write discard.
- DM: Except you don't have a thing to map. It hides the staging belt machinery.
- JG: If it's not inside a pass, you can't do the staging belt because you need to FIFO
- DM: The queue is the FIFO
- JG: Then by the time you do submit, then everything is forcefully unmapped.
- MM: So say that happens and a user partially uploads data, then calls submit. Are we going to do a partial upload?
- CW: Probably the ArrayBuffer will be filled with zeros. If you don't write to all of it, then you get zeros.
- MM: That means apps need to know there's an interaction between two functions called.
- CW: Yes, that's the unfortunate part. The other concern is creating the staging belt on one thread and submitting on the other.
- DM: That's a tricky scenario I think.
- JG: My understanding of why this is better than setSubData is that it wouldn't require a general allocator. But if you can map/unmap in any order, then it's not any disimilar.
- CW: The reason this improves setSubData is that it allows less copies so you can get staging area.
- JG: Why is it useful to be a pass?
- CW: We're now suggesting it's queue ops directly -- not a pass.
- JG: I like that. The shape is effectively then queue.mapBufferRange and then unmap with write_discard.
- DM: And it works for textures too.
- JG: Similar to what I was going to propose where you map the buffer except you get a shadow buffer… we'll see.
- CW: How about we look forward to JG's proposal for next week's meeting?

PR Burndown

- #495 (Re:) Expose limits on the adapter
  - MM: Internally coworkers had miscommunication. Sorry that this happened.
  - MM: About exposing device limits. This is a little bit scary because there are so many different devices with capabilities and limits. If we do it wrong, it could be a fingerprinting concern. Our thought is that we shouldn't expose limits. There is fundamentally a difference between a phone and a giant graphics card. Web authors should hopefully make use of the difference. However, we don't want to

expose every detail of the device to arbitrary websites. Hoping to come to some middle ground and hopefully at least agree there is a problem if we expose every limit.
    - CW: Agreed. Internally we've discussed things like.. trusted context or whatnot. I expect all the limits to be validated, so it will be an implementation / user-agent choice whether it exposes the full range or only the ones it thinks is most useful, or the granularity it thinks is useful. Like only two tiers of 4K and 8K textures if you think that's all that matters.
    - JG: From WebGL, particularly from Tor browser for how to reduce / eliminate some fingerprinting concerns. One things we thought about for WebGL which I wanted to look into was creating implicit feature levels. A couple of tiers of capabilities of graphics cards. If you only have 4 samples, but 16k textures, you might be demoted to low-tier of only 4k textures.
    - MM: That was almost exactly our thinking. Weren't thinking about explicit vs implicit -- but bucketing into just a few buckets should get us the best of both worlds. I think that the definition of the bucket shouldn't just be in the spec The actual API surface of exposing what the limits are is probably okay. The point we're trying to make is the values that are returned from that API.
    - CW: Would like to discuss more whether buckets are defined by the spec or by implementations. It's a great item to put on the agenda.
    - MM: Cool if they were agreed upon in the group and in the spec.

Agenda for next meeting

- GPUUploadPass
- Racy buffer mapping

- In two weeks discuss limits and feature levels