MapLibre Native Metal Status Report: Sep 2023

This is the seventh status report for the MapLibre Native Metal project. For a deeper dive into the project's background, consult the first report.

Overall the status is **Green**. We've made our Q3 goals and are looking at performance and secondary project goals.

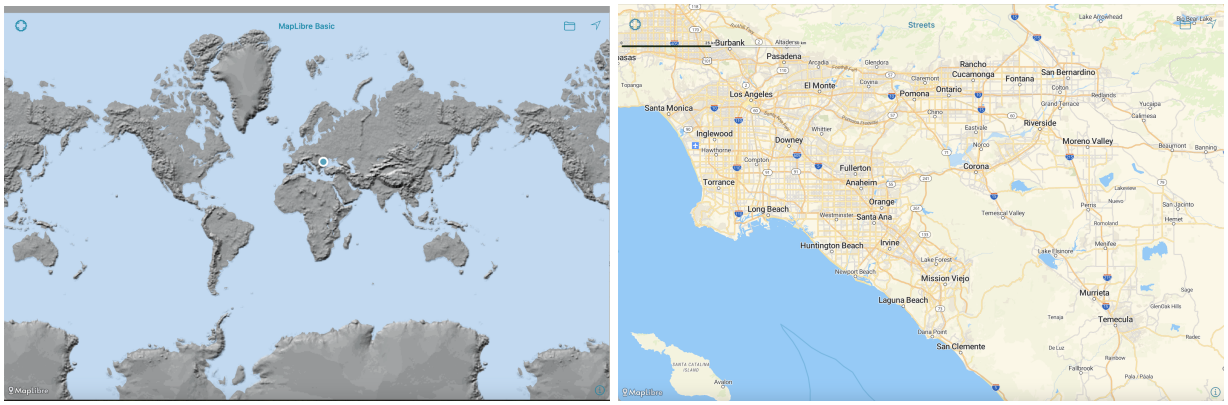
Background

As a quick reminder, we're updating the MapLibre Project to support Metal for iOS and generally modernizing the renderer.

We've wrapped up the first phase, Renderer Modularization with the acceptance of the merge by the community. The team has made great progress on the second phase, the Metal Port. We'll dig into the details below.

Status

We're **green** and we met the Q3 goals, which are discussed later.



This last month has been tracking down bugs, finishing Metal layers, improving the build system, addressing performance and starting the secondary goals.

Q3 Goals

From the June report we introduced a few Q3 goals. Let's see how we're doing.

- **Finish the Mega Merge**
  Accepted by the community and done last month.

- **Direct Version of Metal Port**
  It's working. A few more things to wrap up and performance, performance, performance.

- **Equal Performance**
  We said 'maybe' on this one…. And then I had a look at the June report. It turns out I was referring to OpenGL. So we hit this one too.

- **Try it out with a big user**
  We had public feedback from a number of small users and one big one at Amazon. It needs to be faster, but no crashes so far. I was expecting crashes, so that's great!

We made all our Q3 goals! There's certainly more to do, but still a great milestone. We'll look at Q4 goals later.

Metal Pre-Release

The big effort this month was getting ready for the pre-release. This was an explicit iOS build we shared with the community and for various reasons the 22nd was our deadline. We made it with a day to spare.

Initial feedback was:

- It works!
- It's slower than OpenGL
- There are some obscure problems

Strangely, no crashes yet. No doubt some will turn up, but a very good pre-release!

More on what's up with the performance below. We're also tracking down the obscure problems.

Metal Implementation

The Metal implementation is working, but it's not done. There are still a few features missing, fairly obscure things that most users won't notice. But they are in the tests, so we'll be finishing those next month.

This month the team got the offline rendering working with Metal. That's essential for the rendering tests, most of which seem to be running. Running was the first part, passing is the second. We'll be looking into that next.

Metal Performance

We have some ways to go with Metal performance. Let's compare with the new OpenGL implementation. We use frame times as they don't clip at 60fps. For comparison's sake, 16ms is 60fps. These numbers are from an iPhone 13 Pro Max.

Location	OpenGL (new)	Metal
boston	5.68ms	10.5ms
paris	4.96ms	16.4ms
paris2	4.91ms	16.4ms
alps	4.96ms	16.4ms
us east	4.30ms	5.84ms
greater la	5.00ms	6.48ms
sf	5.02ms	16.4ms
oakland	4.87ms	16.4ms
germany	4.96ms	16.4ms

We're around 3 or 4 times slower in most cases. Metal is quite happy to tell us why.

Metal Buffers

Most people think modern graphics SDKs are about polygons or shaders. I like to say they're really about memory management.

It's not that surprising if you think about it. You're dealing with a very specialized graphics architecture, running little programs, feeding those programs little (or large) chunks of data. How you organize that is obviously going to be important.

What's happening here is we're allocating too many tiny data buffers. MTLBuffers have more weight to them than the GL equivalent and consequently more cost. But Metal points you to a number of solutions.

The most comprehensive is the MTLHeap, the "Leroy Jenkins" of memory management. Which is to say very little management at all. It's fast, but you want to keep a non-Heap mode around for debugging.

The other solution is hinted at in the various draw encoding calls. You can pass in offsets within a given MTLBuffer. That's a really broad hint that you're not supposed to be allocating one per use. Ideally you allocate one per…. something and then share it.

The other other solution is to just sling memory around. There's a mode in the draw encoding calls for that, but it's probably not what we want here.

The OpenGL side likes a lot of little bitty buffers. So this is a pattern shift we're just starting on now.

Secondary Goals

I'm not too worried about finishing up Metal at this point. We have two months to solve the performance problems, sort out the bugs, and finish the missing functionality. So let's look at the secondary goals we've made some progress on.

Custom Layers / Drawable Builders / Buckets

There's more on this in last month's report, but the basic idea is this. We want all the tools we use to build the visuals available to other developers. This will make it easier to build overlays, add new data formats, and new layer types. Hopefully also make it all a bit easier to understand.

We're just starting on this now, using the opportunity presented by wide lines in the Fill Layer. Without going into detail, it's a perfect test case for reusing our internal objects outside of their homes. If this works we can replicate the approach for other builder objects and redo the User Location layer as a test case.

Threading

Moving layer creation and loading to another thread would be a big win, but we don't have time for it on this project. I've pushed it off and put together a proposal for the community to consider.

Q4 Goals

Here's what we're looking at in terms of goals for Q4.

- Metal Release Candidate
  We'll want several more pre-releases followed by a version we'd consider releasing. That will need to have at least equivalent performance to OpenGL.
  The actual release mechanism is controlled by MapLibre's maintainers and Bart (native maintainer) has been working closely with us.

- Equivalent Performance on Android.
  By changing the rendering core we've changed the OpenGL implementation and thus the Android version. But we have not looked too closely at it. We need to fix any performance problems we may have caused.

- Several Style Functions on the GPU
  This was discussed in more detail in the last report. The future of the toolkit is likely to be more work done on the GPU. It's doubtful that would be desirable for every function and even more doubtful we could complete that goal in the remaining time. But several of the most common style expressions would improve performance and pave the way for the rest.

- Easier Expandability for MapLibre Native
    - Custom Layers that use our tools for building display geometry
    - Replace & add new shaders without recompiling the toolkit
    - Replace & add new layers without recompiling the toolkit
    - Examples of each for documentation

Wins

The pre-release is the big win this month. Getting the toolkit bundled up and having other people incorporate it in their apps is a huge win! The team deserves kudos for that.

Challenges

As I've mentioned before, the secondary goals are the big challenge now. Moving more functionality to the GPU, opening up the Layer creation, all that good stuff.

We also need to address toolkit size. Though we're probably under the 5% increase that people consider to be "reasonable" I'd rather get to 0% if we can.

Insights

It really is all about memory management. Even on the OpenGL side much of the CPU time is spent on wrangling uniform buffer objects, the parameters you pass in to shaders.

Moving computation to the GPU is also an exercise in doing less memory management and more logic in the shaders.

Risks

The Metal implementation is pretty low risk at this point. Testing could be a problem, though. We need to sustain interest in the pre-release to get a few rounds of testing out of app developers.

To facilitate that we're going to only do a few formal pre-releases, though they can always grab new versions as they're built. The next one we make noise about will focus on performance.