Stamen

# MapLibre MLT Proof of Concept Final Report

A live version of this document is available at https://sta.mn/hp7.

Contents

## Introduction

Stamen has created a proof of concept JavaScript MLT decoder that can be benchmarked against MVT parsing and integrated directly into Maplibre. This document includes a discussion of the work undertaken, future opportunities, and links to external resources.

## Process

When we started work we found the working code available in the maplibre-tile-spec repo was written in Java and in an experimental state. Therefore, we decided to focus initially on ensuring that the Java encoder and decoder were functional and robust before starting to write a new JavaScript decoder.

In effect, our approach to writing a functioning JavaScript MLT decoder was:

1. To make the Java encoder more robust to ensure we have valid MLTs and can parse MVTs without errors
2. To create a new Java encoder command line tool with a mode that disabled “advanced” (FastPFor & FSST) encodings
    a. This resulted in known downsides including larger tile sizes and less optimal decoding speeds. But the advantage is that it created a more feasible set of encodings to support in Javascript given limited time. And it sets us up in the future to improve the performance and size by adding back in support for the advanced encodings (see the Future work section below for more details)
3. To make the Java decoder more robust to ensure we have a good reference implementation
4. To port the functioning Java decoder from (2) to JavaScript

## Tasks Completed

### Java Encoder and Decoder

The Java encoder and decoder can be found in the /java directory of the maplibre-tile-spec repo. The initial work here was reorganizing the repo, making the code run cross-platform, and asserting no regressions by introducing continuous integration via GitHub Actions.

From here, we hardened, tested, and fixed bugs found by using the encoder and decoder with a variety of input MVTs. All major bugs impacting Bing tiles were fixed in the Java implementation. As a result of this hardening, by mid-June we had a solid decoder implementation that was ready to port to JavaScript.

It’s worth noting that at the moment the only way to create an MLT is with an input MVT–the encoder accepts and MVT and creates an MLT.

### JavaScript Decoder

The JavaScript decoder can be found in the /js directory of the maplibre-tile-spec repo. While there was a JavaScript decoder in this repo previously, it was far behind the Java encoder (approximately six months of work behind) and not a complete implementation. Therefore it was required that we write an entirely new JavaScript decoder to replace the previous one.

## Benchmarking

We have benchmarked the initial performance of decoding both MLTs and MVTs in JavaScript. Our benchmarks compare the JavaScript decoder we wrote for MLTs against the Mapbox vector-tile-js library for MVTs.

The vector-tile-js library was originally written to do only the essential work needed for MapLibre, and therefore is highly efficient. So we expected our MLT decoder to perform slower than vector-tile-js at first (see the Future work section below for ways we could significantly speed up MLT decoding).

The results of our benchmarks indicate that MLT decoding in JS is currently at least 2-4x slower than MVT parsing in JS if we compare average decoding time to parse all the features of a tile into the in-memory representation used by MapLibre.

We see this performance difference reliably across tiles and operating systems. For example, for the Bing tile at 5/16/11:

- On an ARM Apple M2 Mac and 32 GB ram:
    - MVT decoding takes an average of 1.9 ms
    - MLT decoding takes an average of 7.6 ms
- On a Windows Server 2022 machine with 4 cores & 16 GB ram:
    - MVT decoding takes an average of 3.7 ms
    - MLT decoding takes an average of 11.7 ms

These benchmarks can be replicated on any machine by installing node.js 20.x and setting up the benchmark like:

```
git clone git@github.com:maplibre/maplibre-tile-spec
cd maplibre-tile-spec/js
npm ci
```

Then executing the following command to benchmark MVT decoding:

```
node dist/bench/decode-mvt.js ../test/fixtures/bing/5-16-11.mvt
```

And executing the following command to benchmark MLT decoding:

```
node dist/bench/decode-mlt.js ../test/expected/bing/5-16-11.mlt
```

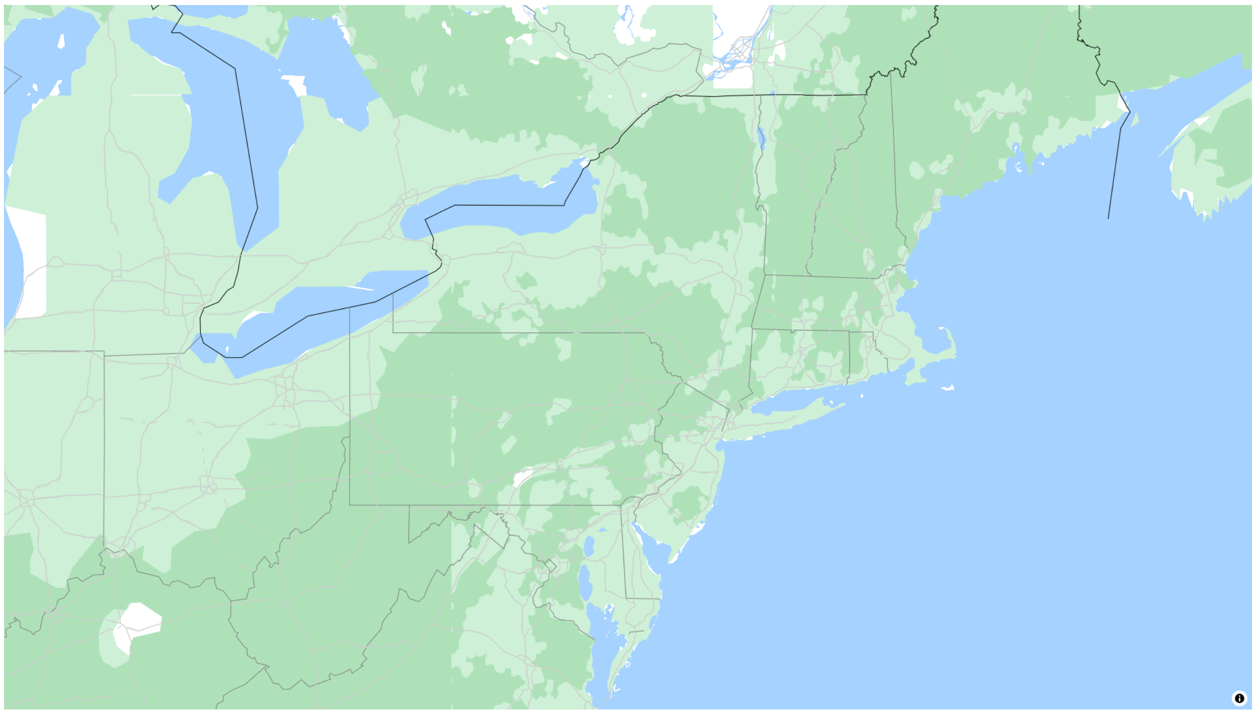
Our focus for these benchmarks was ensuring that we were measuring the differences between MLT and MVT parsing accurately and creating tools that could be used to do so in the future with any input tiles. There are other benchmarking targets we wanted to get to but did not, some of which were mentioned by the team at Microsoft. These include:

- Benchmarking sparse and dense tiles and comparing performance.
- Benchmarking tiles at different zoom levels and comparing performance.
- Extensive benchmarking in the browser such as using a headless browser.

## Maplibre GL JS Initial Support

Along with a functioning JavaScript decoder, we have also written a thin wrapper with the same API as the Mapbox vector-tile-js library. As mentioned above, this library is the key dependency in MapLibre GL JS that parses MVTs and provides them to the rest of the library to be rendered. So with relatively few changes we can render MLTs directly in MapLibre GL JS.

Note that while an initial proposal suggested that we would use `addProtocol` in order to convert MLTs to MVTs, then render them, this work skips the conversion to MVT. MLTs are therefore being rendered directly in MapLibre GL JS. This is an exciting step for us and significantly exceeded the expectations of the MapLibre community for this round of work. You can view a demo of this work at mltdemo.stamen.com.



*An example of MLTs generated from Bing tiles rendered in MapLibre GL JS using a simple style.*

One notable aspect of the MLT spec is that it currently requires an auxiliary metadata file in order to decode an MLT tile. While this has the advantage of reducing tile size, we found that in practice this leads to an inefficiency in the ability to integrate the code into MapLibre because it currently requires fetching 2 assets per tile (the tile and the metadata). We've raised this issue with the community and discussions are underway to solve it.

## Future Work

We are excited about the amount we have been able to accomplish within a relatively short time frame, and we are energized by the momentum building in the community around the MLT spec.

It is clear to us that the MLT specification is novel, expertly designed, and has major promise to unlock increased performance for MapLibre users in the future. With the work of the POC in place, now we are ready to turn our attention to the future work needed to unlock that increased performance.

First of all, we see a major opportunity to focus on enhancing the JavaScript decoding implementation by optimizing the performance and efficiency of the code, both in terms of the speed of execution and the number of memory allocations. Some of these optimizations can happen purely in the JavaScript code and others may require enhancements to the specification in terms of how data is structured. Some optimizations will be very logical, predictable improvements to process data more efficiently. Others should be considered exploratory investigations to carefully and critically examine the viability of major new features that exist in the specification. For example, the spec suggests that pre-tessellation *could* happen and *could* save time on triangulation and the spec has already been structured to be able to store either normal geometries or pre-tessellated triangles. However, the encoder and decoders have not yet implemented this specification feature and exploration is needed to evaluate whether the decoding speed and size of tiles containing pre-tessellated geometries are advantageous enough to warrant this approach.

Please see the document
https://github.com/maplibre/maplibre-tile-spec/blob/main/js/future.md for a detailed discussion of areas of the greatest potential for improvement in the future.

## Appendix: Code Delivered

### maplibre-tile-spec

Most of the code worked on is on the main branch of the maplibre/maplibre-tile-spec repo. We have added a tag, poc-2024-06-28, to create a snapshot of the code at this milestone. Specifically you will find the Java encoder and decoder in `java/` and the JavaScript decoder in `js/` within the repo. Javascript decoder tests are in `js/test/` and benchmarking code is in `js/bench/`.

### maplibre-gl-js

As a demonstration of the minimal effort required to load MLTs with MapLibre GL JS, we made a demonstration pull request on Stamen's fork of the repo. This code is experimental and is not intended to be ready yet to provide to the main MapLibre GL JS repo.