MapLibre Tiles: A Next Generation Vector Tiles Format specially designed for OSM data

Markus Tremmel

State of the Map 2024

About Me

Staff Software Engineer and Architect at Rohde and Schwarz in Germany

Lecturer at the Deggendorf Institute of Technology (DIT) on the applied computer science programme

Email: markus.tremmel@th-deg.de

Researching on the next generation map rendering stack

Disclaimer: Presented as an independent developer. Opinions are my own

Twitter: mactremmel

GitHub: mactrem

Email: markus.tremmel23@gmail.com

Introducing Vector Tiles on openstreetmap.org

2024: announcing the year of the OpenStreetMap vector maps

OpenStreetMap will take a large leap forward with the introduction of vector tiles on openstreetmap.org this year. This is the first of a series of blog posts where we will share our progress.

To lead our vector tiles project, the OpenStreetMap Foundation has hired Paul Norman, a renowned figure in cartography and open data, whose journey with OpenStreetMap began 2010 with a chance encounter on the xkcd forums. His role in the community took off with work on OpenStreetMap Carto in 2013. His volunteer involvement with the OSM Founda cluding contributions to several working groups and a tenure on the OSMF board, high

tial positions

seen the

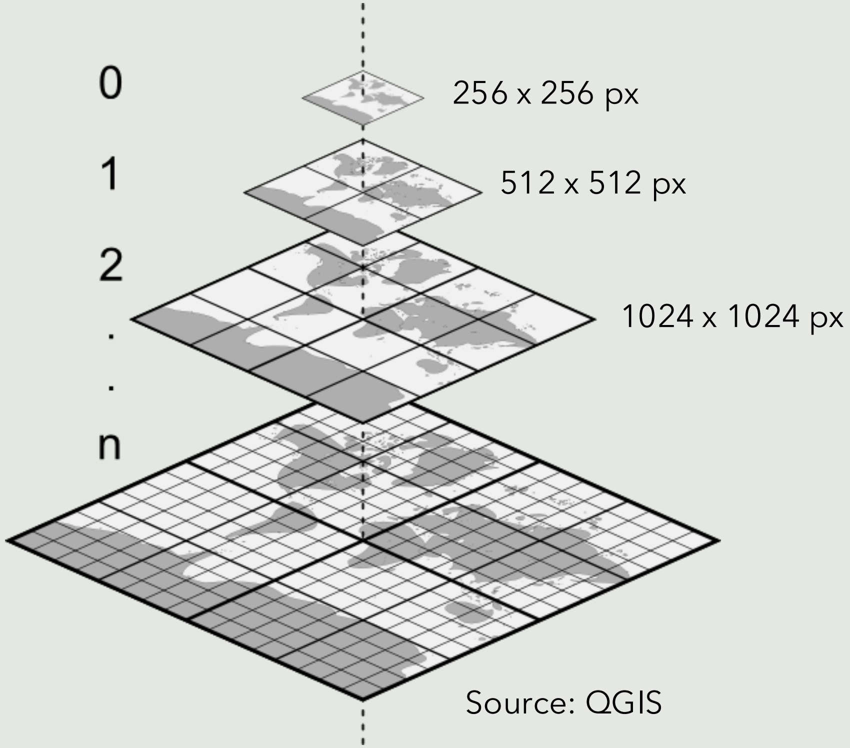
to vecto

Source: Guillaume Rischard, blog.openstreetmap.org

Disclaimer: Neither me nor the MLT format is involved in this activity

Raster Tiles

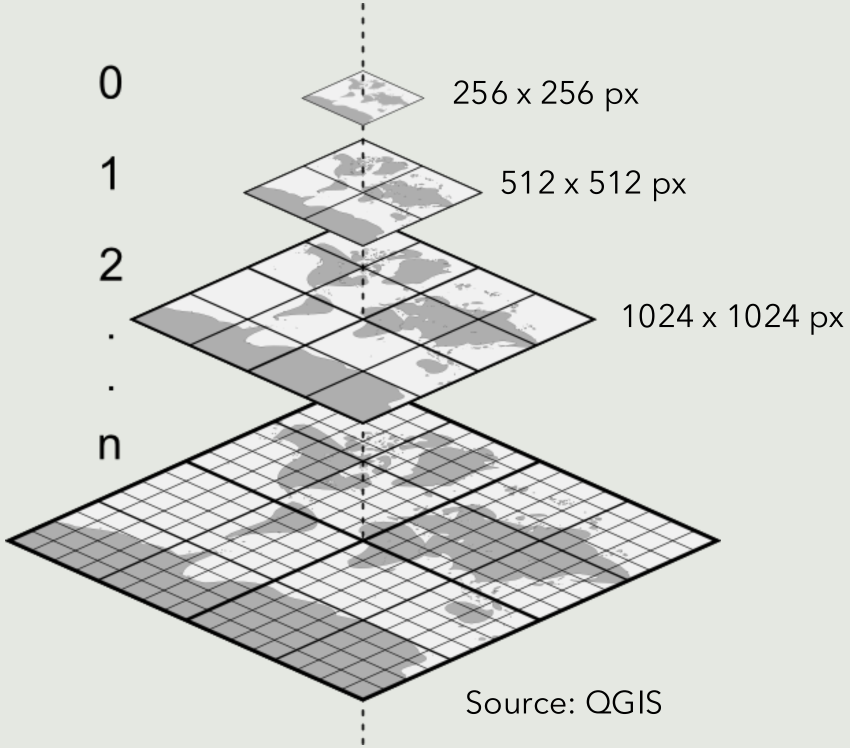
Pre-redered images of the actual vector data encoded as PNG, JPEG, WebP, ...



Source: QGIS

Raster Tiles

Pre-redered images of the actual vector data encoded as PNG, JPEG, WebP, ...



Source: QGIS

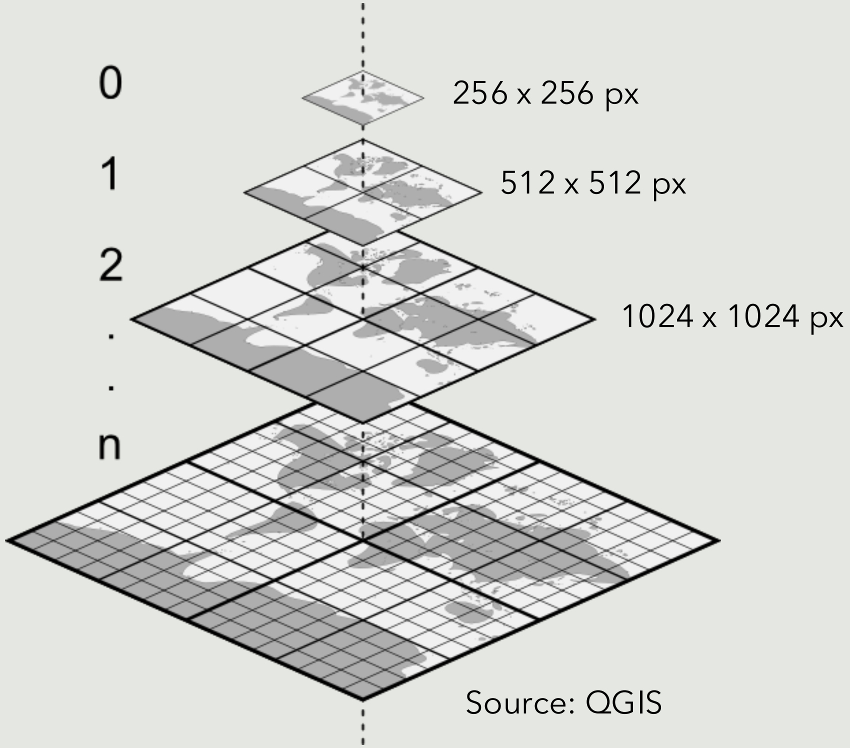
Vector Tiles

Contains vector data (points, lines and polygons) that are rendered on-the-fly on the client, usually on the GPU e.g. in the browser via WebGL

Source: Tilezen

Raster Tiles

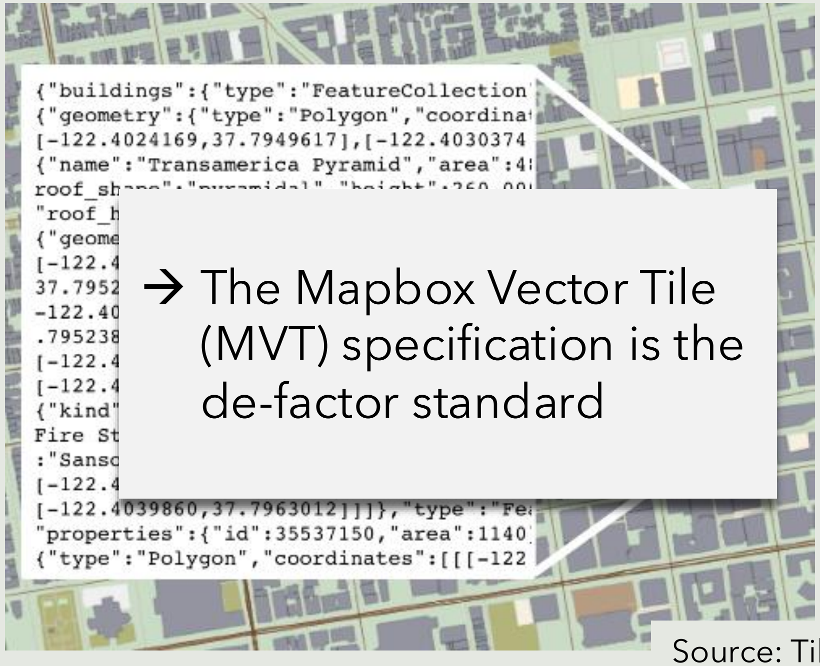
Pre-redered images of the actual vector data encoded as PNG, JPEG, WebP, ...



Source: QGIS

Vector Tiles

Contains vector data (points, lines and polygons) that are rendered on-the-fly on the client, usually on the GPU e.g. in the browser via WebGL



→ The Mapbox Vector Tile (MVT) specification is the de-factor standard

Source: Tilezen

Raster Tiles

- Less powerful hardware needed as tiles are already pre-rendered
- No GPU needed → works on all types of devices and browsers
- Simple map client implementations

Vector Tiles

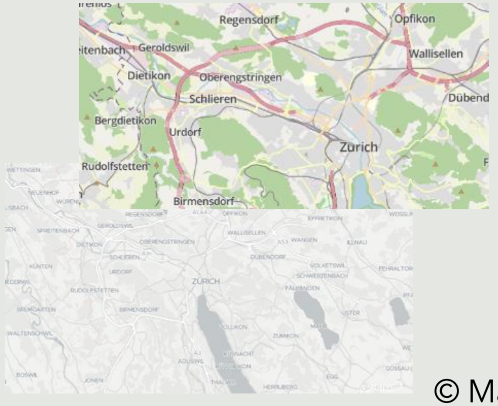
- Dynamic Styling → client decides on styling
- Smooth Zooming
- Interactivity
- Smaller tile size → Offline Maps
- Rotation of the map while labels stay aligned

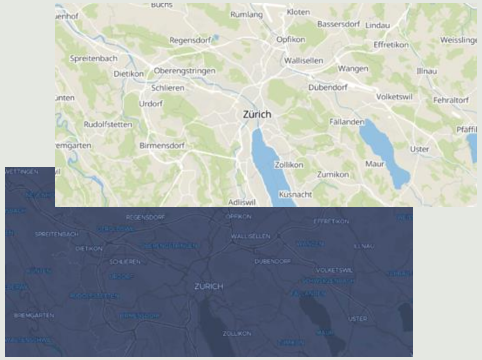
Raster Tiles

- Less powerful hardware needed as tiles are already pre-rendered
- No GPU needed → works on all types of devices and browsers
- Simple map client implementations

Vector Tiles

- **Dynamic Styling → client decides on styling**
- Smooth Zooming
- Interactivity
- Smaller tile size → Offline Maps
- Rotation of the map while labels stay aligned



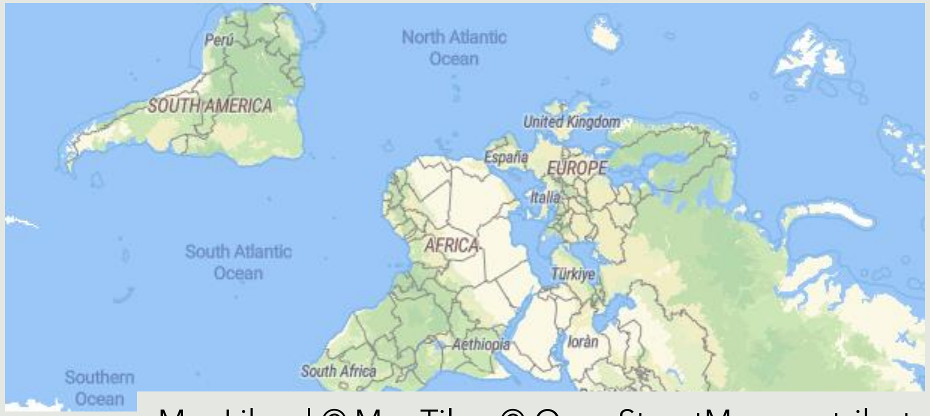


Raster Tiles

- Less powerful hardware needed as tiles are already pre-rendered
- No GPU needed → works on all types of devices and browsers
- Simple map client implementations

Vector Tiles

- Dynamic Styling → client decides on styling
- Smooth Zooming
- Interactivity
- Smaller tile size → Offline Maps
- **Rotation of the map while labels stay aligned**



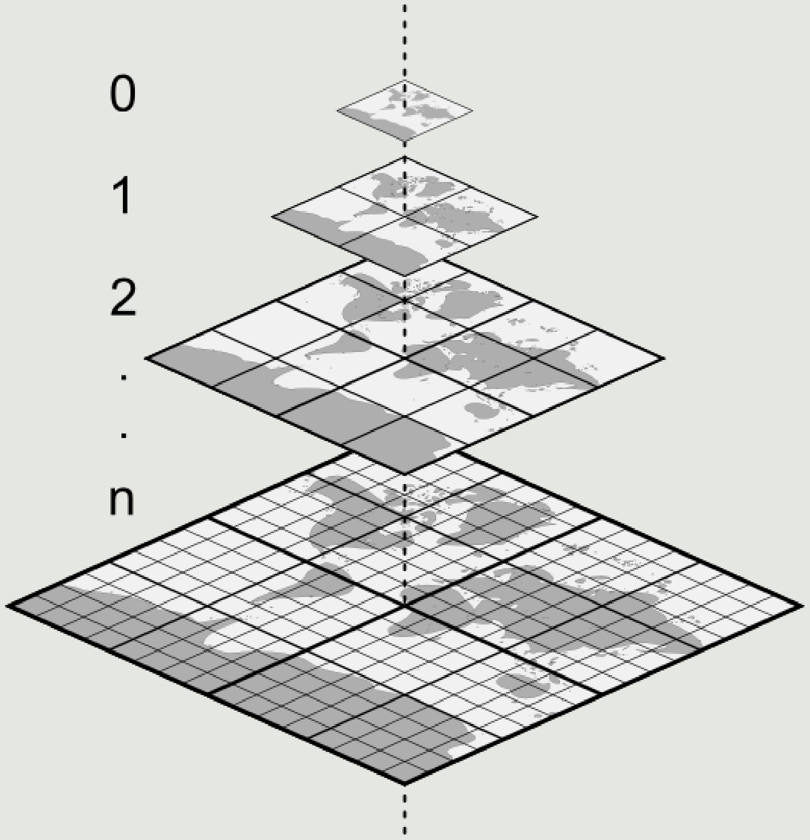
MapLibre | © MapTiler © OpenStreetMap contributors

Why a new vector tiles format?

- Continuously growing amount of data based on the advances on geospatial sensors and software technolgogies (AI)
- Novel algorithms to improve data compression and speedup decoding as well as processing performance in the latest scientific publications
- New more complex geospatial source formats
- New modern and next generation Graphics APIs

MapLibre Tiles

- The MLT format is mainly inspired by the Mapbox Vector Tile specification but has been redesigned from the ground to address key challenges such as continuously growing geospatial data volumes and the more complex next-genaration spatial source formats
- MLT is specifically designed for current and next generation graphics APIs to enable fast rendering of large (planet-scale) 2D and 2.5 basemaps
- As with MVT, the geometries are based on the Simple Feature Access (SFA) model of the OGC and are defined in a screen coordinate system
- To combine a good compression ratio with fast decoding and processing, the MLT format is split into an in-memory and a storage format
- The format is designed to enable fast transcoding of the storage into the in-memory



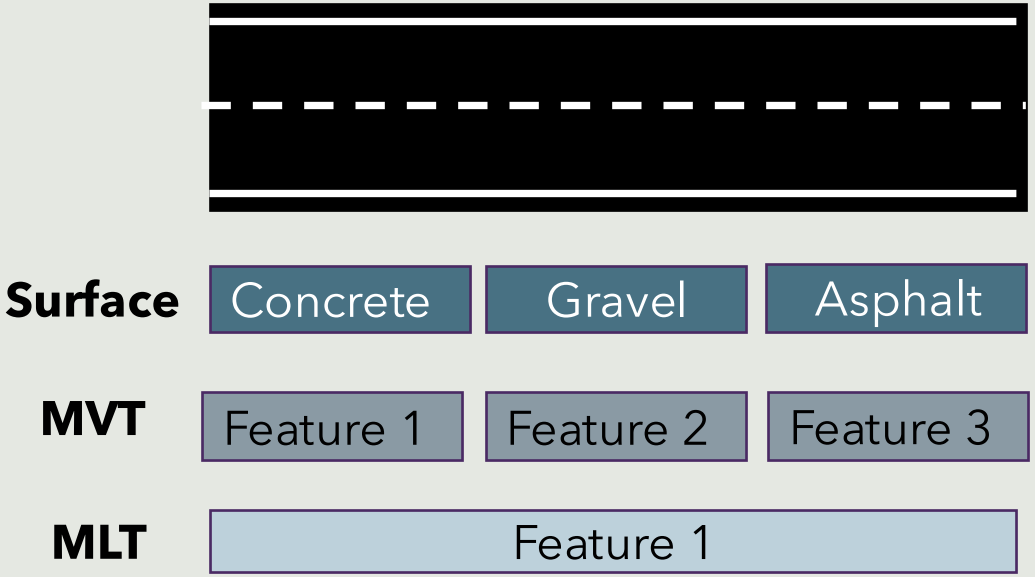
Source: QGIS

MLT Features

- Improved compression ratio: up to 6x on large tiles, based on a column-oriented file layout with (custom) recursively applied lightweight encodings → making a additional heavyweight compression (Gzip, Brotili, …) unnecessary in some cases
- Better decoding performance: fast lightweight encodings which can be used in combination with SIMD/vectorization instructions
- Improved processing performance: based on an in-memory format that can be processed efficiently on the CPU and GPU
- Support for linear referencing and m-values to efficiently support the upcoming next-generation source formats such as Overture Maps (GeoParquet)
- Support for 3D coordinates, i.e. elevation
- Support for complex types, including nested properties, lists and maps

MLT Features

- Improved compression ratio: up to 6x on large tiles, based on a column-oriented file layout with (custom) recursively applied lightweight encodings
- Better decoding performance: fast lightweight encodings which can be used in combination with SIMD/vectorization instructions
- Improved processing performance: Based on an in-memory format that can be processed efficiently on the CPU and GPU
- **Support for linear referencing and m-values to efficiently support the upcoming next-generation source formats such as Overture Maps (GeoParquet)**
- Support for 3D coordinates, i.e. elevation
- Support for complex types, including nested properties, lists and maps



MLT Features

- Improved compression ratio: up to 6x on large tiles, based on a column-oriented file layout with (custom) recursively applied lightweight encodings
- Better decoding performance: fast lightweight encodings which can be used in combination with SIMD/vectorization instructions
- Improved processing performance: based on an in-memory format that can be processed efficiently on the CPU and GPU
- Support for linear referencing and m-values to efficiently support the upcoming next-generation source formats such as Overture Maps (GeoParquet)
- Support for 3D coordinates, i.e. elevation
- **Support for complex types, including nested properties, lists and maps**

Places Theme of Overture Maps

```json
{
  "id": "overture:places:place:1",
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      0,
      0
    ]
  },
  "properties": {
    "categories": {
      "main": "some_category",
      "alternate": [
        "another_category"
      ]
    },
    "confidence": 0.9,
    "websites": [
      "https://www.example.com"
    ],
    "emails": [
      "info@example.com"
    ],
    "socials": [
      "https://www.twitter.com/example"
    ],
```

MLT Storage Format

- The storage format is used for the cost-efficient storage and low latency transfer of the vector data over the network
- MLT defines a platform-agnostic representation to avoid the expensive materialization costs in particular for strings
- It is based on a column-oriented layout compared to the record-oriented approach used in MVT
- The columns are compressed based on lightweight SIMD-friendly encodings that can be recursively applied
- A logical column is separated into several physical streams (sub-columns) inspired by the ORC file format which are stored next to each other

Example File Layout

FeatureTable Metadata	ID Field					Geometry Field							Feature Scoped Property Fields									
													class					subclass				
	FM	SM	Present	SM	Data	FM	SM	GeometryType	SM	NumParts	SM	VertexBuffer	FM	SM	Present	SM	Data	FM	SM	Present	SM	Data

Encodings

DataType	Logical Level Technique	Physical Level Technique
Boolean	Boolean RLE	
Integer	Plain, RLE, Delta, Delta-RLE	SIMD-FastPFOR, Varint
Float	Plain, RLE, Dictionary, ALP	
String	Plain, Dictionary, FSST Dictionary	
Geometry	Plain, Dictionary, Morton-Dictionary	

MLT Storage Format

Original GeoJson

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "geometry": { ... },
      "type": "Feature",
      "properties": {
        "hello": "world",
        "h": "world",
        "count": 1.23
      }
    },
    {
      "geometry": { ... },
      "type": "Feature",
      "properties": {
        "hello": "again",
        "count": 2
      }
    }
]}
```

MVT

```
layers {
  version: 2
  name: "points"
  features: {
    id: 1
    tags: 0
    tags: 0
    tags: 1
    tags: 0
    tags: 2
    tags: 1
    type: Point
    geometry: ...
  }
  features {
    id: 2
    tags: 0
    tags: 2
    tags: 2
    tags: 3
    type: Point
    geometry: ...
  }
  keys: "hello"
  keys: "h"
  keys: "count"
  values: { string_value: "world" }
  values: { double_value: 1.23 }
  values: { string_value: "again" }
  values: { int_value: 2 }
  extent: 4096
}
```

MLT

*SM Stream Metadata
*FM Field (Column) Metadata

FeatureTable Metadata	ID Column			Geometry Column					Feature Scoped Attribute Columns																		
									hello							h							count				
	FM	SM	Data	FM	SM	GeometryType	SM	VertexBuffer	FM	SM	Present	SM	Length	SM	Data	FM	SM	Present	SM	Length	SM	Data	FM	SM	Present	SM	Data

MLT Storage Format

Original GeoJson

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "geometry": { ... },
      "type": "Feature",
      "properties": {
        "hello": "world",

        "h": "world",

        "count": 1.23
      }
    },
    {
      "geometry": { ... },
      "type": "Feature",
      "properties": {
        "hello": "again",

        "count": 2
      }
    }
]}
```

MVT

```
layers {
  version: 2
  name: "points"
  features: {
    id: 1
    tags: 0
    tags: 0
    tags: 1
    tags: 0
    tags: 2
    tags: 1
    type: Point
    geometry: ...
  }
  features {
    id: 2
    tags: 0
    tags: 2
    tags: 2
    tags: 3
    type: Point
    geometry: ...
  }
  keys: "hello"
  keys: "h"
  keys: "count"
  values: { string_value: "world" }
  values: { double_value: 1.23 }
  values: { string_value: "again" }
  values: { int_value: 2 }
  extent: 4096
}
```

MLT

*SM Stream Metadata
*FM Field (Column) Metadata

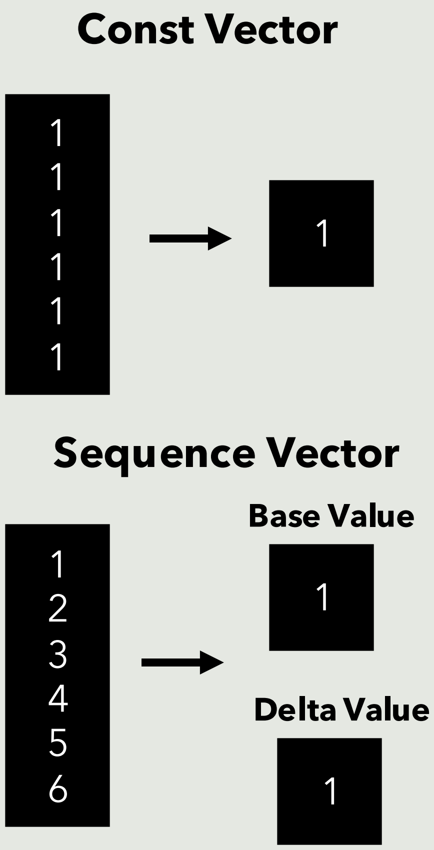
FeatureTable Metadata	ID Column			Geometry Column					Feature Scoped Attribute Columns																		
									hello							h							count				
	FM	SM	1 2	FM	SM	Point Point	SM	x1 y1 x2 y2	FM	SM	true true	SM	5 5	SM	world again	FM	SM	true false	SM	5	SM	world	FM	SM	true true	SM	1.23 2

MLT In-Memory Format

- Is inspired by the in-memory analytics formats such as Apache Arrow as well as the 3D graphics format glTF and is tailored for the map visualization use case to be efficiently processed and rendered at runtime
- Allow random (constant time) access to all data, so it can also be parallel processed on the GPU (e.g. on WebGPU compute shader)
- The data are stored in Vectors that are arrays of data of the same type
- The vectorized query execution model will be used to process the Vectors in batches instead of single records
- Vectors can be stored in compressed form for faster decoding and efficient processing → flat, const, sequence, dictionary and fsst dictionary Vectors

MLT In-Memory Format

- Is inspired by the in-memory analytics formats such as Apache Arrow as well as the 3D graphics format glTF and is tailored for the map visualization use case to be efficiently processed and rendered at runtime
- Allow random (constant time) access to all data, so it can also be parallel processed on the GPU (e.g. on WebGPU compute shader)
- The data are stored in Vectors that are arrays of data of the same type
- The vectorized query execution model will be used to process the Vectors in batches instead of single records
- **Vectors can be stored in compressed form for faster decoding and efficient processing → flat, const, sequence, dictionary and fsst dictionary Vectors**



Source: DubckDB

Current Status

- First initial draft of the specification, see https://github.com/maplibre/maplibre-tile-spec/tree/main/specs
- POC integration of MLT into the MapLibre GL JS rendering library thanks to Microsoft, Stamen, Yuri Astrakhan, Dane Springmeyer and Eric Brelsford
- Compression ratio: Up to 6x reduction on large OpenMapTiles scheme based tiles and up to 2x reduction on large Bing Maps tiles (without the usage of a heavyweight compression)
- Decoding Performance:
  - *First benchmarks of the Java decoder show an on average about 4x speed of MLT against MVT up when MLT is decoded into the proposed in-memory format*
  - *First benchmarks of the Js decoder shown an about up to 2x slower performance of MLT against MVT when MLT is decoded into the MapLibre GL JS native in-memory representation mainly because of the additional expensive transformation step from the column-oriented to the row-oriented layout*

Thanks for listining

And To:

Microsoft

Stamen

MapLibre

Yuri Astrakhan

Dane Springmeyer

Eric Brelsford

Ante Viducic

Felix Fischer