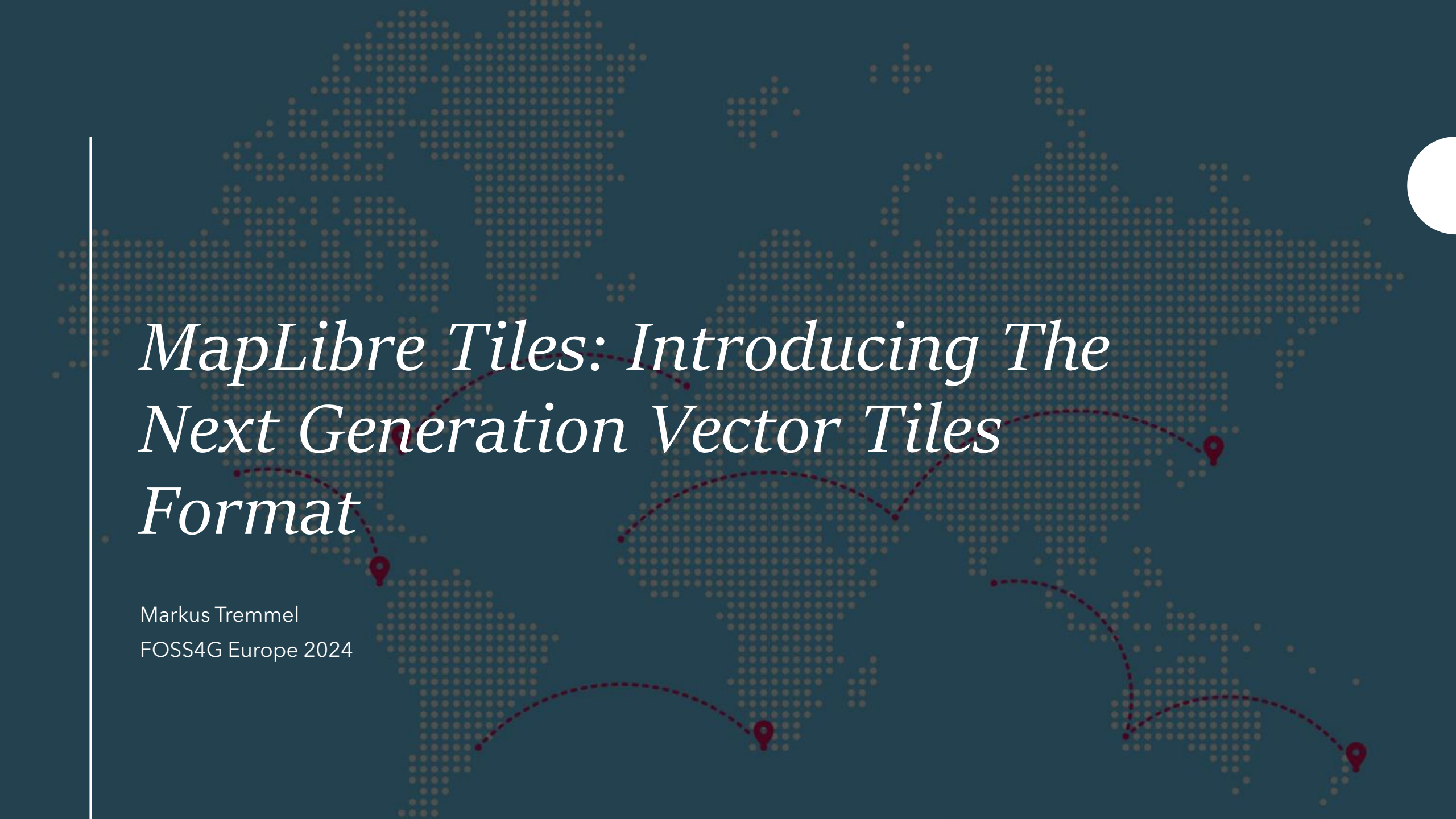


MapLibre Tiles: Introducing The Next Generation Vector Tiles Format

Markus Tremmel

FOSS4G Europe 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

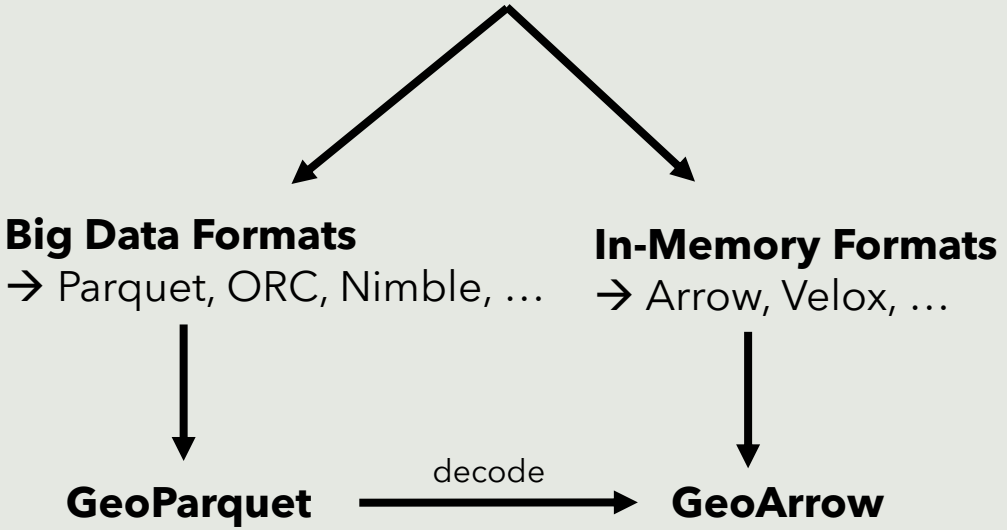
Geospatial Vector Formats

Use Cases: **Visualization (Graphics Format)** **Analysis (Analytics Format)**

Geospatial Vector Formats

Visualization (Graphics Format)

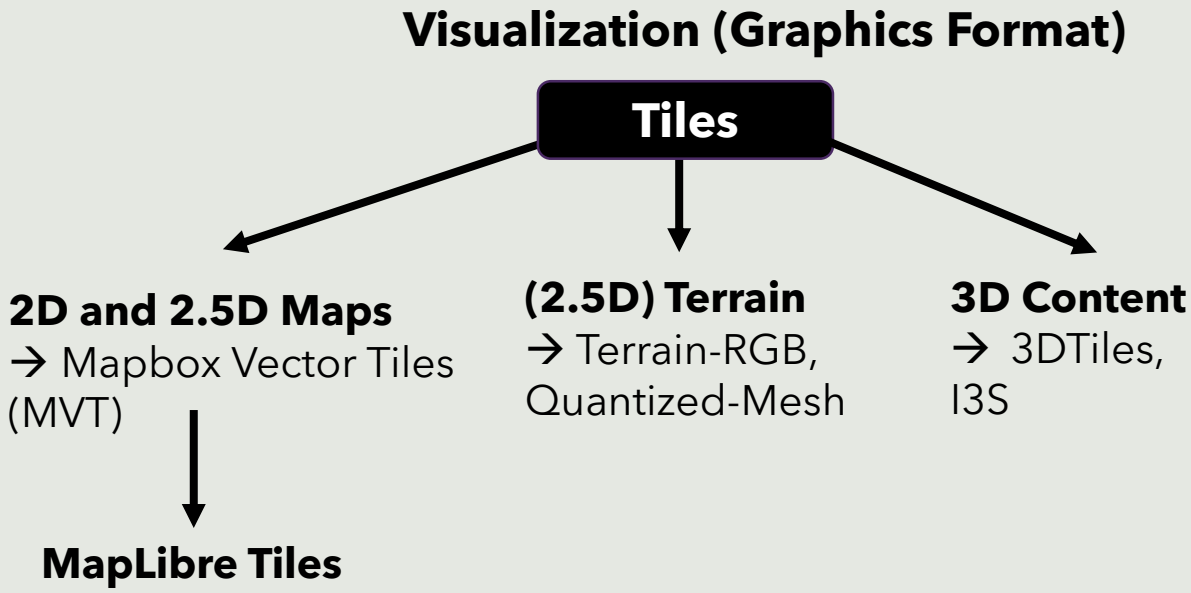
Analysis (Analytics Format)

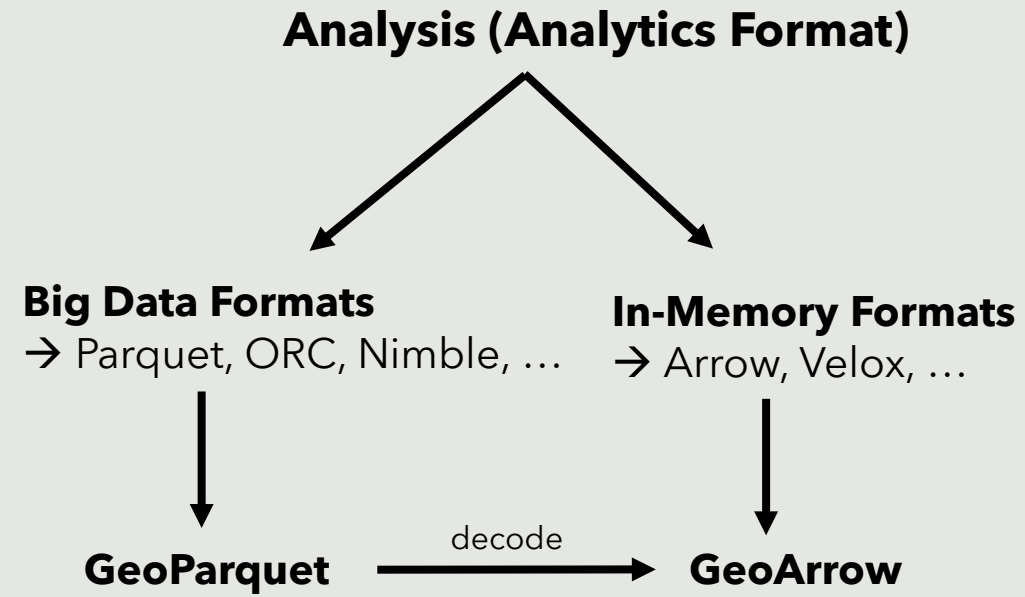


→ Focus on support for efficient spatial analytic queries (spatial joins, range queries, k-nearest neighbor, …)

→ Efficient visualizion of overlays

Geospatial Vector Formats





→ Focus on visualizing large basemaps
→ Basically only support for efficient selection (filerting) queries

→ Focus on support for efficient spatial analytic queries (spatial joins, range queries, k-nearest neighbor, …)
→ Efficient visualizion of overlays

Geospatial Vector Formats

Visualization (Graphics Format)

Tiles

2D and 2.5D Maps
→ Mapbox Vector Tiles (MVT)

(2.5D) Terrain
→ Terrain-RGB, Quantized-Mesh

3D Content
→ 3DTiles, I3S

MapLibre Tiles

Analysis (Analytics Format)

Big Data Formats
→ Parquet, ORC, Nimble, …

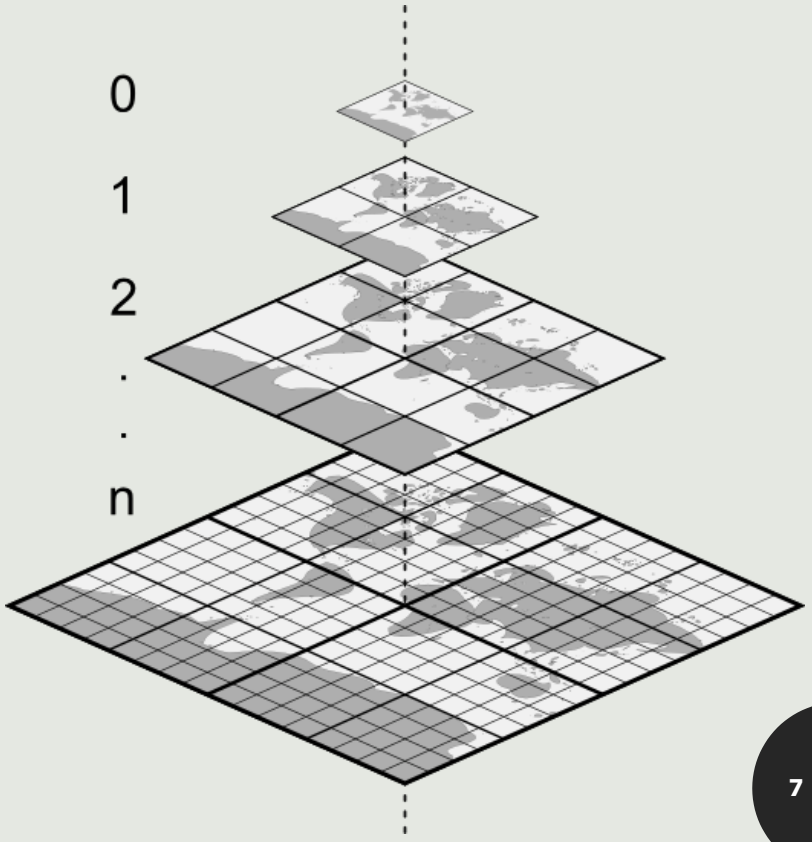
In-Memory Formats
→ Arrow, Velox, …

GeoParquet —decode→ **GeoArrow**

→ Focus on support for efficient spatial analytic queries (spatial joins, range queries, k-nearest neighbor, …)
→ Efficient visualizion of overlays

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 format as well as to be efficiently processed and rendered at runtime



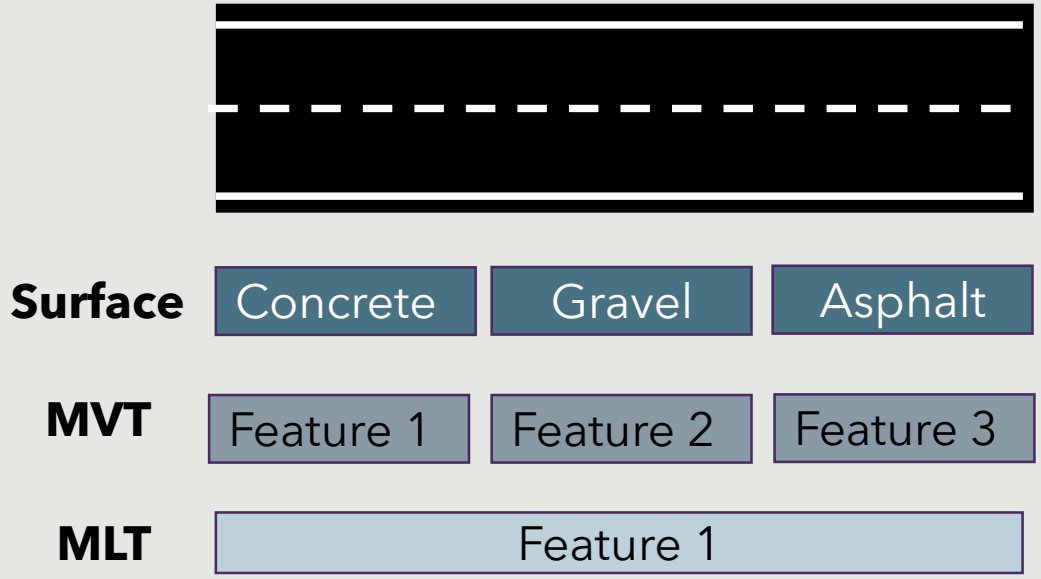
Source: QGIS

MLT Features

- Improved compression ratio: up to 6x on large tiles, based on a column-oriented file layout with (custom) recursively applied lightweight encodings
- 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


```
{
  "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"
    ],
```

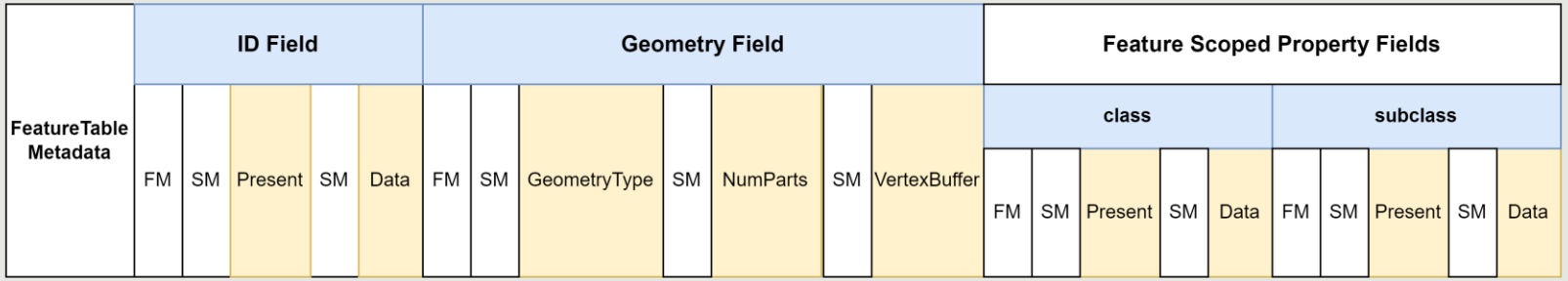
MLT Storage Format

- The storage format is used for the cost-efficient storage and low latency transfer of the vector data over the network
- 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

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	

Example File Layout

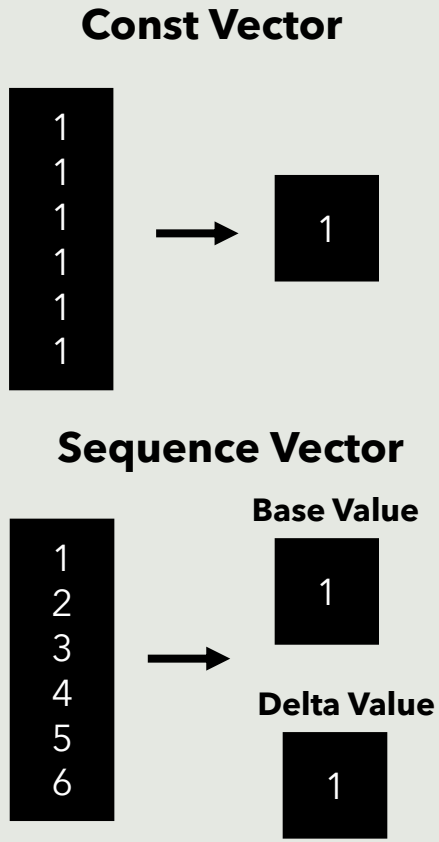


MLT In-Memory Format

- 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, 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 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

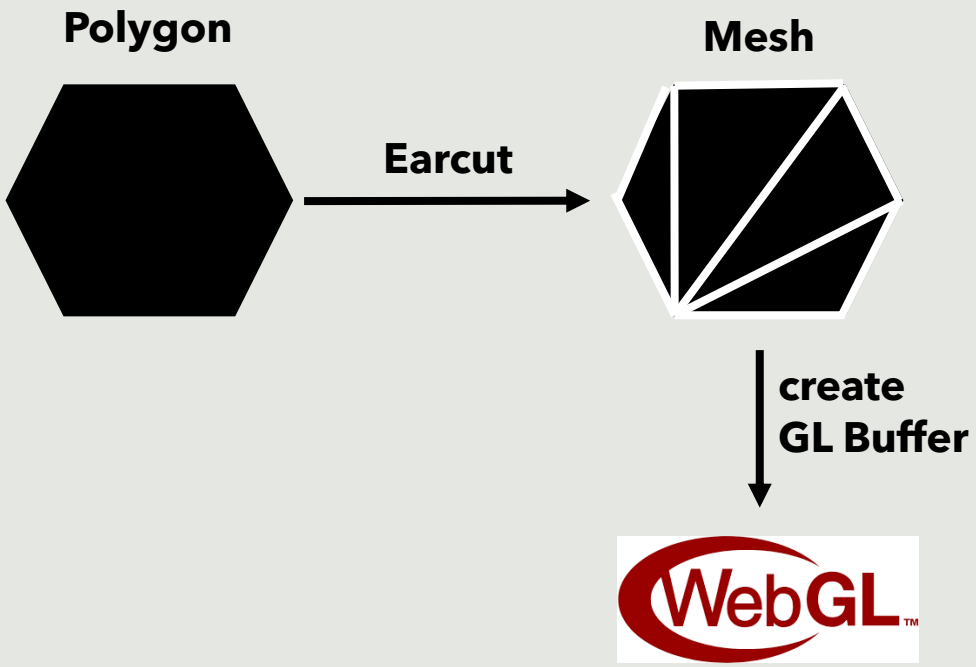
Map Rendering Bottlenecks

and how MLT can improve on that issues …

Profiling MapLibre GL JS Worker

Self Time		Total Time		Activity
241.4 ms	6.5 %	244.0 ms	6.6 %	readUtf8TextDecoder
182.7 ms	4.9 %	201.5 ms	5.5 %	Minor GC
173.2 ms	4.7 %	1366.6 ms	37.0 %	addFeature
142.3 ms	3.9 %	1273.5 ms	34.5 %	Function Call
137.6 ms	3.7 %	158.3 ms	4.3 %	isEarHashed
136.5 ms	3.7 %	392.4 ms	10.6 %	earcutLinked
				addHalfVertex
				addLine
				populate
105.8 ms	2.9 %	112.6 ms	3.0 %	readTag
83.1 ms	2.3 %	91.9 ms	2.5 %	reserve
80.9 ms	2.2 %	128.1 ms	3.5 %	loadGeometry
80.7 ms	2.2 %	675.0 ms	18.3 %	performSymbolLayout
80.4 ms	2.2 %	118.1 ms	3.2 %	serialize
65.0 ms	1.8 %	77.9 ms	2.1 %	insert
62.9 ms	1.7 %	64.0 ms	1.7 %	sortLinked
61.0 ms	1.7 %	150.7 ms	4.1 %	evaluate
55.9 ms	1.5 %	74.4 ms	2.0 %	Major GC

Polygon Tessellation



Self Time		Total Time		Activity
241.4 ms	6.5 %	244.0 ms	6.6 %	readUtf8TextDecoder
182.7 ms	4.9 %	201.5 ms	5.5 %	Minor GC
173.2 ms	4.7 %	1366.6 ms	37.0 %	addFeature
142.3 ms	3.9 %	1273.5 ms	34.5 %	Function Call
137.6 ms	3.7 %	158.3 ms	4.3 %	isEarHashed
136.5 ms	3.7 %	392.4 ms	10.6 %	earcutLinked
131.9 ms	3.6 %	169.9 ms	4.6 %	addHalfVertex
112.4 ms	3.0 %	356.4 ms	9.7 %	addLine
112.1 ms	3.0 %	1531.1 ms	41.5 %	populate
105.8 ms	2.9 %	112.6 ms	3.0 %	readTag
83.1 ms	2.3 %	91.9 ms	2.5 %	reserve
80.9 ms	2.2 %	128.1 ms	3.5 %	loadGeometry
80.7 ms	2.2 %	675.0 ms	18.3 %	performSymbolLayout
80.4 ms	2.2 %	118.1 ms	3.2 %	serialize
65.0 ms	1.8 %	77.9 ms	2.1 %	insert
62.9 ms	1.7 %	64.0 ms	1.7 %	sortLinked
61.0 ms	1.7 %	150.7 ms	4.1 %	evaluate
55.9 ms	1.5 %	74.4 ms	2.0 %	Major GC

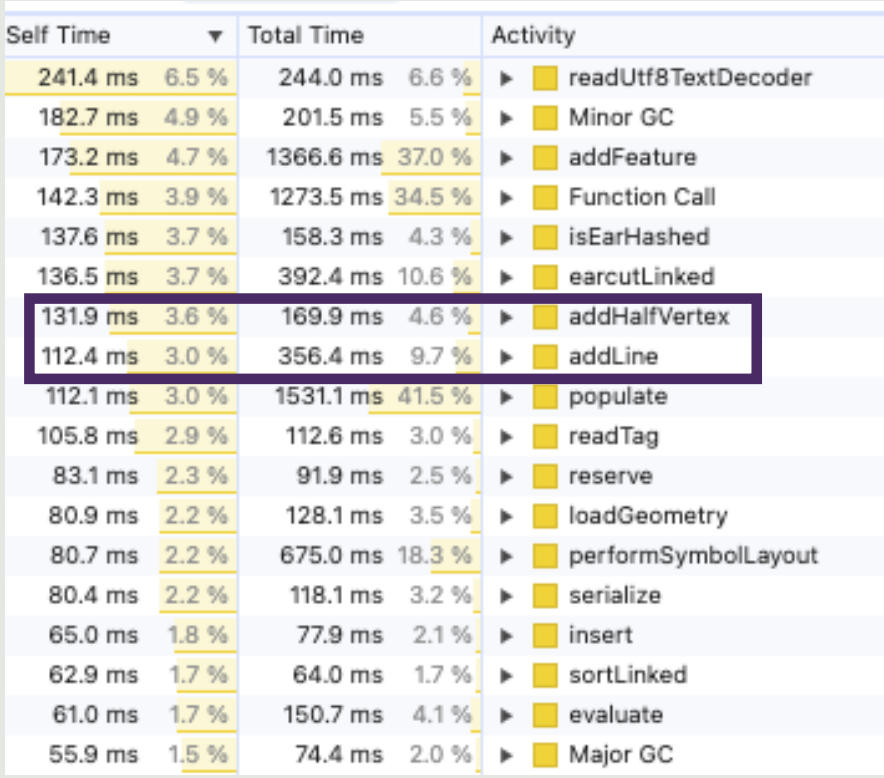
Polygon Pre-Tessellation

- MLT supports the storage of the polygon meshes directly in the tile to avoid the expensive tessellation step on the client at runtime
- MLT uses the indexed geometries approach inspired from the glTF format by adding an IndexBuffer in addition to the VertexBuffer
- The IndexBuffer and VertexBuffer can be directly copied to GPU buffers without further processing
- The existing recursive applied lightweight integer encodings are used to reduce the size of the additional buffers

Stream name	Data type	encoding	mandatory
GeometryType	Byte	see availabe integer encodings	✓
NumGeometries	UInt32	see availabe integer encodings	×
NumParts	UInt32	see availabe integer encodings	×
NumRings	UInt32	see availabe integer encodings	×
NumTriangles	UInt32	see availabe integer encodings	×
IndexBuffer	UInt32	see availabe integer encodings	×
VertexOffsets	UInt32	see availabe integer encodings	×
VertexBuffer	Int32 or Vertex[]	Plain, Dictionary or Morton Dictionary	✓

What about Lines?

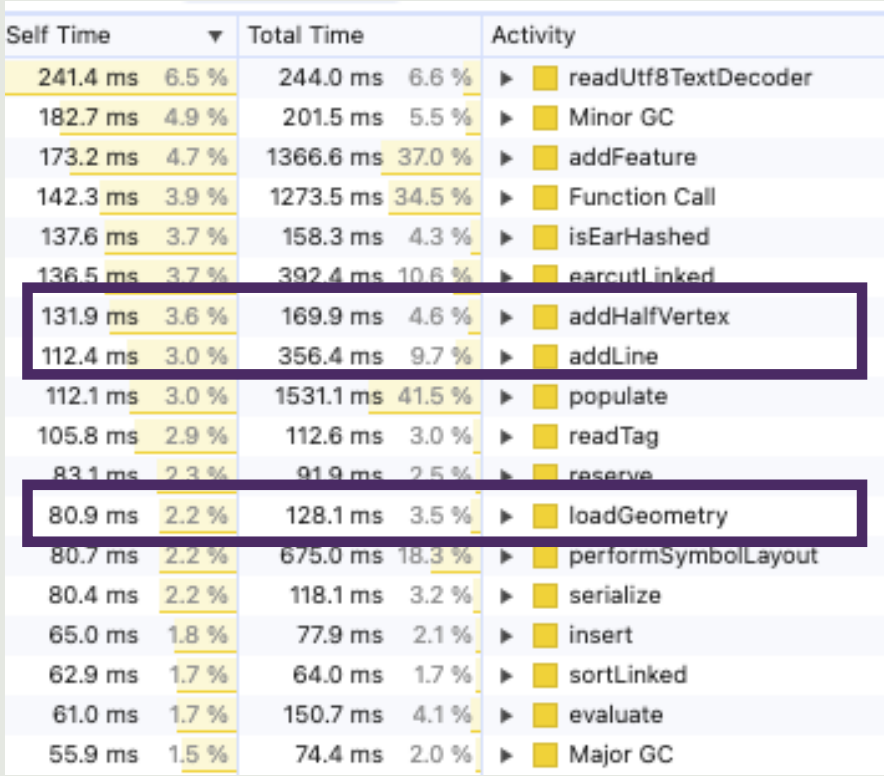
- Drawing lines in WebGL is very limited, therefore lines also have to be triangulated
- Since in modern cartography the structure of LineString geometries (caps, joins, …) can be changed at runtime based on the map styles, storing pre-tessellated geometries is not practicable
- Based on the WebGPU compute shader, this computation can be moved to the GPU in the browser in the future to take advantage of the massive parallel capabilities



Self Time		Total Time		Activity
241.4 ms	6.5 %	244.0 ms	6.6 %	readUtf8TextDecoder
182.7 ms	4.9 %	201.5 ms	5.5 %	Minor GC
173.2 ms	4.7 %	1366.6 ms	37.0 %	addFeature
142.3 ms	3.9 %	1273.5 ms	34.5 %	Function Call
137.6 ms	3.7 %	158.3 ms	4.3 %	isEarHashed
136.5 ms	3.7 %	392.4 ms	10.6 %	earcutLinked
131.9 ms	3.6 %	169.9 ms	4.6 %	addHalfVertex
112.4 ms	3.0 %	356.4 ms	9.7 %	addLine
112.1 ms	3.0 %	1531.1 ms	41.5 %	populate
105.8 ms	2.9 %	112.6 ms	3.0 %	readTag
83.1 ms	2.3 %	91.9 ms	2.5 %	reserve
80.9 ms	2.2 %	128.1 ms	3.5 %	loadGeometry
80.7 ms	2.2 %	675.0 ms	18.3 %	performSymbolLayout
80.4 ms	2.2 %	118.1 ms	3.2 %	serialize
65.0 ms	1.8 %	77.9 ms	2.1 %	insert
62.9 ms	1.7 %	64.0 ms	1.7 %	sortLinked
61.0 ms	1.7 %	150.7 ms	4.1 %	evaluate
55.9 ms	1.5 %	74.4 ms	2.0 %	Major GC

Tile processing performance

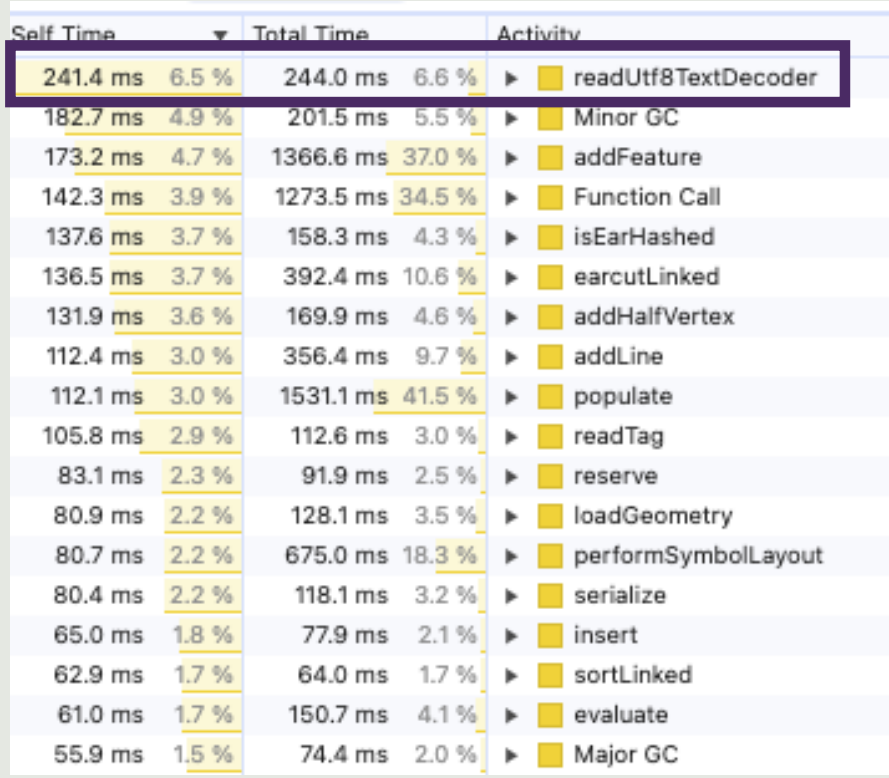
- **A column oriented format enables faster processing of geometries through better cache locality and the usage of SIMD instructions**
- Directly work on compressed vectors to avoid the expensive materialization of strings
- A column oriented layout (SoA) adds less pressure on the garbage collector compared to the Array of structures (AoS) approach where a lot of small objects are created
- Use the vectorized query execution model to speed up the filtering performance



Self Time		Total Time		Activity
241.4 ms	6.5 %	244.0 ms	6.6 %	readUtf8TextDecoder
182.7 ms	4.9 %	201.5 ms	5.5 %	Minor GC
173.2 ms	4.7 %	1366.6 ms	37.0 %	addFeature
142.3 ms	3.9 %	1273.5 ms	34.5 %	Function Call
137.6 ms	3.7 %	158.3 ms	4.3 %	isEarHashed
136.5 ms	3.7 %	392.4 ms	10.6 %	earcutLinked
131.9 ms	3.6 %	169.9 ms	4.6 %	addHalfVertex
112.4 ms	3.0 %	356.4 ms	9.7 %	addLine
112.1 ms	3.0 %	1531.1 ms	41.5 %	populate
105.8 ms	2.9 %	112.6 ms	3.0 %	readTag
83.1 ms	2.3 %	91.9 ms	2.5 %	reserve
80.9 ms	2.2 %	128.1 ms	3.5 %	loadGeometry
80.7 ms	2.2 %	675.0 ms	18.3 %	performSymbolLayout
80.4 ms	2.2 %	118.1 ms	3.2 %	serialize
65.0 ms	1.8 %	77.9 ms	2.1 %	insert
62.9 ms	1.7 %	64.0 ms	1.7 %	sortLinked
61.0 ms	1.7 %	150.7 ms	4.1 %	evaluate
55.9 ms	1.5 %	74.4 ms	2.0 %	Major GC

Tile processing performance

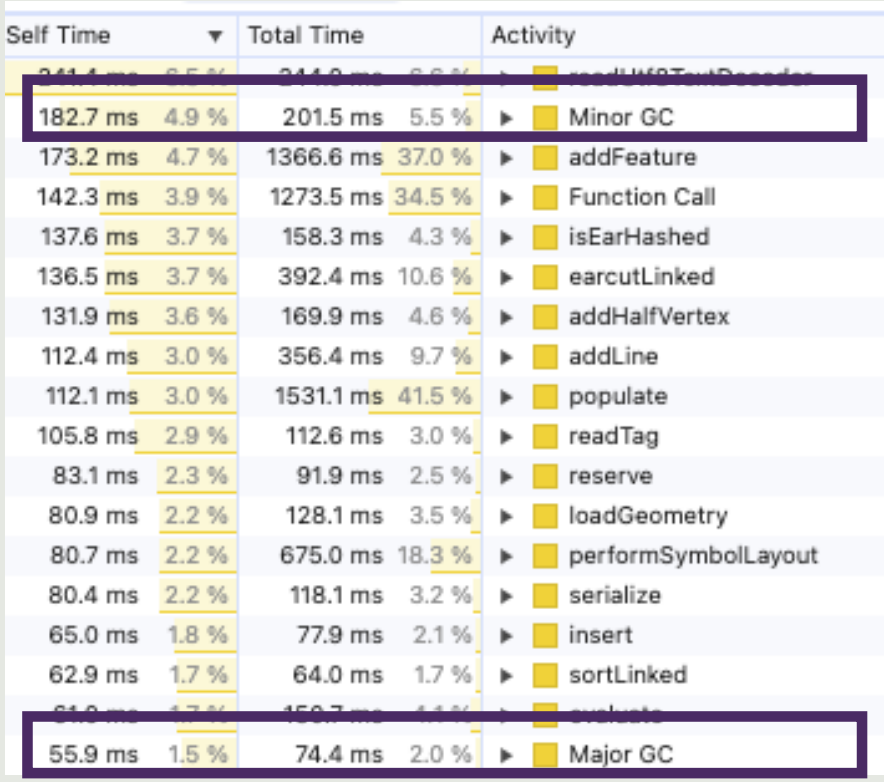
- A column oriented format enables faster processing of geometries through better cache locality and the usage of SIMD instructions
- **Directly work on compressed vectors to avoid the expensive materialization of strings**
- A column oriented layout (SoA) adds less pressure on the garbage collector compared to the Array of structures (AoS) approach where a lot of small objects are created
- Use the vectorized query execution model to speed up the filtering performance



Self Time		Total Time		Activity
241.4 ms	6.5 %	244.0 ms	6.6 %	readUtf8TextDecoder
182.7 ms	4.9 %	201.5 ms	5.5 %	Minor GC
173.2 ms	4.7 %	1366.6 ms	37.0 %	addFeature
142.3 ms	3.9 %	1273.5 ms	34.5 %	Function Call
137.6 ms	3.7 %	158.3 ms	4.3 %	isEarHashed
136.5 ms	3.7 %	392.4 ms	10.6 %	earcutLinked
131.9 ms	3.6 %	169.9 ms	4.6 %	addHalfVertex
112.4 ms	3.0 %	356.4 ms	9.7 %	addLine
112.1 ms	3.0 %	1531.1 ms	41.5 %	populate
105.8 ms	2.9 %	112.6 ms	3.0 %	readTag
83.1 ms	2.3 %	91.9 ms	2.5 %	reserve
80.9 ms	2.2 %	128.1 ms	3.5 %	loadGeometry
80.7 ms	2.2 %	675.0 ms	18.3 %	performSymbolLayout
80.4 ms	2.2 %	118.1 ms	3.2 %	serialize
65.0 ms	1.8 %	77.9 ms	2.1 %	insert
62.9 ms	1.7 %	64.0 ms	1.7 %	sortLinked
61.0 ms	1.7 %	150.7 ms	4.1 %	evaluate
55.9 ms	1.5 %	74.4 ms	2.0 %	Major GC

Tile processing performance

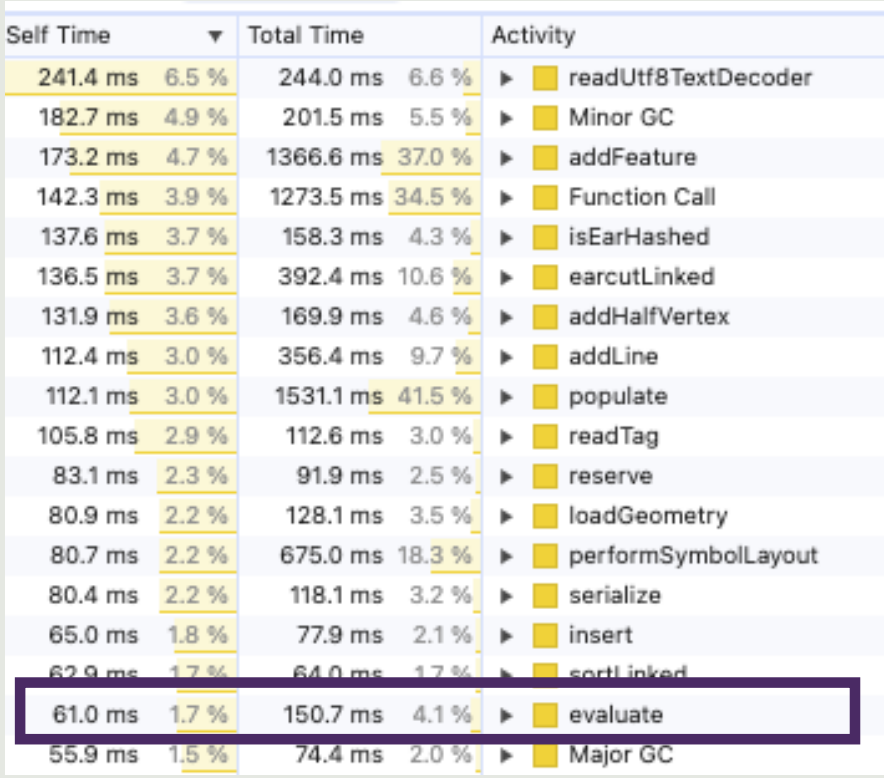
- A column oriented format enables faster processing of geometries through better cache locality and the usage of SIMD instructions
- Directly work on compressed vectors to avoid the expensive materialization of strings
- **A column oriented layout (SoA) adds less pressure on the garbage collector compared to the Array of structures (AoS) approach where a lot of small objects are created**
- Use the vectorized query execution model to speed up the filtering performance



Self Time		Total Time		Activity
182.7 ms	4.9 %	201.5 ms	5.5 %	Minor GC
173.2 ms	4.7 %	1366.6 ms	37.0 %	addFeature
142.3 ms	3.9 %	1273.5 ms	34.5 %	Function Call
137.6 ms	3.7 %	158.3 ms	4.3 %	isEarHashed
136.5 ms	3.7 %	392.4 ms	10.6 %	earcutLinked
131.9 ms	3.6 %	169.9 ms	4.6 %	addHalfVertex
112.4 ms	3.0 %	356.4 ms	9.7 %	addLine
112.1 ms	3.0 %	1531.1 ms	41.5 %	populate
105.8 ms	2.9 %	112.6 ms	3.0 %	readTag
83.1 ms	2.3 %	91.9 ms	2.5 %	reserve
80.9 ms	2.2 %	128.1 ms	3.5 %	loadGeometry
80.7 ms	2.2 %	675.0 ms	18.3 %	performSymbolLayout
80.4 ms	2.2 %	118.1 ms	3.2 %	serialize
65.0 ms	1.8 %	77.9 ms	2.1 %	insert
62.9 ms	1.7 %	64.0 ms	1.7 %	sortLinked
55.9 ms	1.5 %	74.4 ms	2.0 %	Major GC

Tile processing performance

- A column oriented format enables faster processing of geometries through better cache locality and the usage of SIMD instructions
- Directly work on compressed vectors to avoid the expensive materialization of strings
- A column oriented layout (SoA) adds less pressure on the garbage collector compared to the Array of structures (AoS) approach where a lot of small objects are created
- **Use the vectorized query execution model to speed up the filtering performance**



Self Time		Total Time		Activity
241.4 ms	6.5 %	244.0 ms	6.6 %	readUtf8TextDecoder
182.7 ms	4.9 %	201.5 ms	5.5 %	Minor GC
173.2 ms	4.7 %	1366.6 ms	37.0 %	addFeature
142.3 ms	3.9 %	1273.5 ms	34.5 %	Function Call
137.6 ms	3.7 %	158.3 ms	4.3 %	isEarHashed
136.5 ms	3.7 %	392.4 ms	10.6 %	earcutLinked
131.9 ms	3.6 %	169.9 ms	4.6 %	addHalfVertex
112.4 ms	3.0 %	356.4 ms	9.7 %	addLine
112.1 ms	3.0 %	1531.1 ms	41.5 %	populate
105.8 ms	2.9 %	112.6 ms	3.0 %	readTag
83.1 ms	2.3 %	91.9 ms	2.5 %	reserve
80.9 ms	2.2 %	128.1 ms	3.5 %	loadGeometry
80.7 ms	2.2 %	675.0 ms	18.3 %	performSymbolLayout
80.4 ms	2.2 %	118.1 ms	3.2 %	serialize
65.0 ms	1.8 %	77.9 ms	2.1 %	insert
62.9 ms	1.7 %	64.0 ms	1.7 %	sortLinked
61.0 ms	1.7 %	150.7 ms	4.1 %	evaluate
55.9 ms	1.5 %	74.4 ms	2.0 %	Major GC

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 without the usage of SIMD instructions*
  - *First benchmarks of the Js decoder show 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:

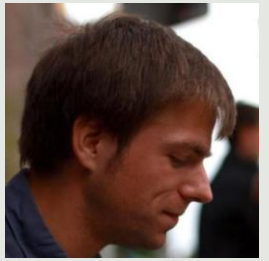
Microsoft

Stamen

MapLibre

Yuri Astrakhan

Dane Springmeyer



Eric Brelsford

Ante Viducic

Felix Fischer