Cloud-optimierte Geoformate

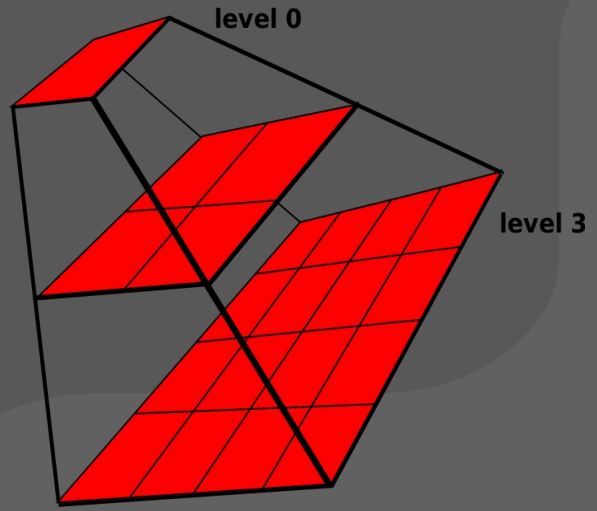
Pirmin Kalberer
@implgeo
Sourcepole AG, Zürich
www.sourcepole.ch

Webzugriff – wo liegt das Problem?

- Geodaten Files können sehr gross sein
- Lesen von Daten über das Netzwerk viel langsamer als von lokaler Disk
- Laden des gesamten Files für lokalen Zugriff unpraktikabel

Lösung #1: Tile caches

- Aufsplitten der Daten in Kacheln
  → Es müssen nur sichtbare Kacheln geladen werden
- Funktioniert gut für Rasterdaten (XZY, WMTS)
- Nachteile: Hoher Speicherplatzbedarf, aufwändige Cache-Generierung, Labelling erschwert



Lösung #2: Dateiinhalt sortieren

- Optimierte Ordnung des Dateiinhaltes zum Lesen von “Chunks”
- HTTP Range Requests für partielles Lesen
- Format ist “cloud optimized”

- HTTP Range Request:

```
curl http://example.com/image.tif -H "Range: bytes=0-1023"
```

```
GET /image.tif HTTP/1.1
Host: example.com
Range: bytes=0-1023
```

Rasterdaten

GeoTIFF:

- Unterstützt Kacheln
- Unterstützt Overviews

Cloud Optimized GeoTIFF (COG):

- Metadaten sortiert
- Bilddaten sortiert
- https://www.cogeo.org/

Erzeugung Cloud Optimized GeoTIFFs

- **GeoTIFF zu COG:**

```
gdal_translate in.tif cog.tif -co TILED=YES
-co COPY_SRC_OVERVIEWS=YES -co COMPRESS=DEFLATE
```

- **Seit GDAL 3.1:**

```
gdal_translate in.tif cog.tif -of COG -co COMPRESS=DEFLATE
```

Lesen von Cloud Optimized GeoTIFFs

- **GDAL datasource (QGIS, Mapserver, etc.):**
- `/vsicurl/https://s3-us-west-2.amazonaws.com/planet-disaster-data/hurricane-harvey/SkySat_Freeport_s03_20170831T162740Z3.tif`

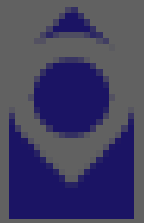
- **Im Browser mit geotiff.js**
  - https://geotiffjs.github.io/
- **Leaflet: georaster-layer-for-leaflet**
  - https://github.com/geotiff/georaster-layer-for-leaflet
  - Beispiel: https://share.streamlit.io/mykolakozyr/cogviewer/main/app.py

Vektordaten

Bestehende Vektorformate:

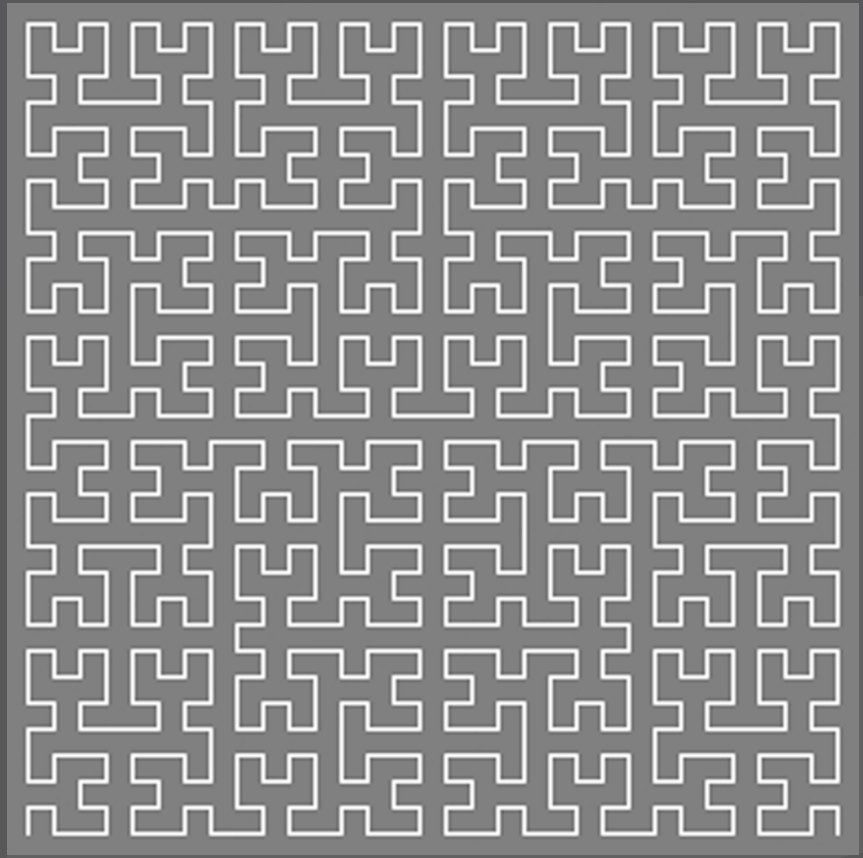
- GeoJSON/GML: Ungeeignet
- GPKG: Optimiert für Disk Block I/O
- Shapfiles: Zu limitiert und zu viele Sidecars

FlatGeobuf:

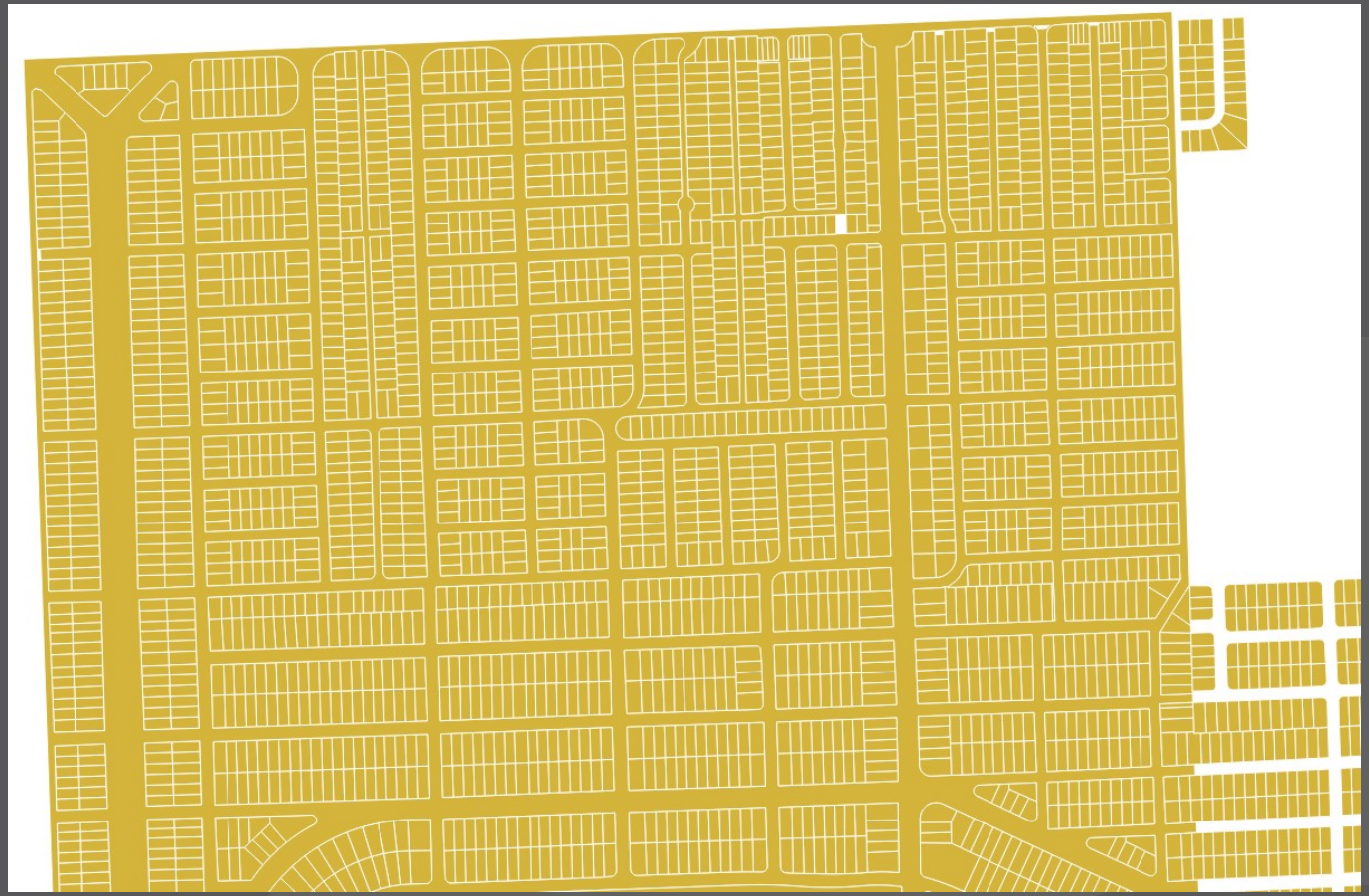


- Metadaten mit Index (packed Hilbert R-Tree)
- Sortierte Vektordaten
- Basiert auf FlatBuffers (Schema, Portabel, Verifikation)
- https://flatgeobuf.org/

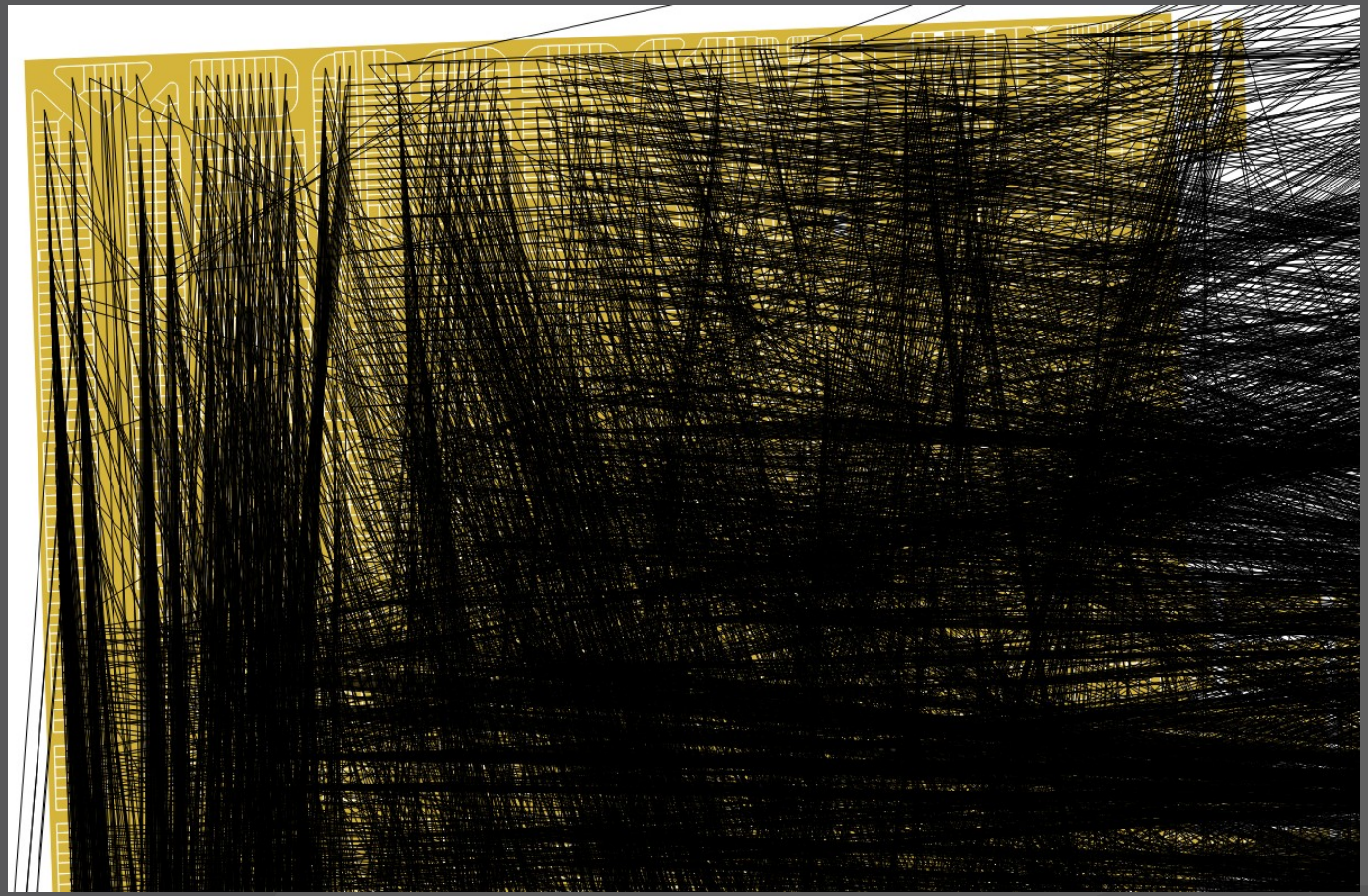
Hilbertkurve



Hilbert Sortierung 1/3



Hilbert Sortierung 2/3



Hilbert Sortierung 3/3

Bonus: Hilbert Sortierung in PostGIS

```
CREATE INDEX poly_hilbert_idx
ON poly_table (ST_Centroid(wkb_geometry));


CLUSTER poly_table
USING poly_hilbert_idx;
```

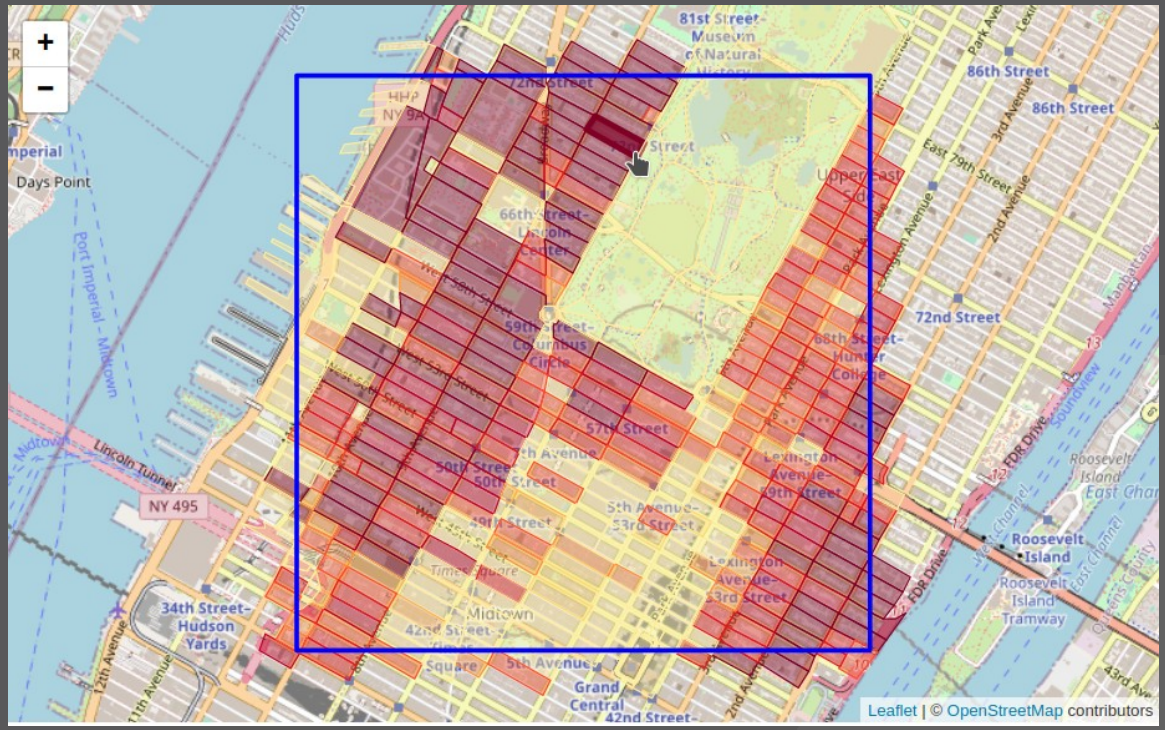
Erzeugung und Lesen von FlatGeobuf

- **GDAL 3.1:**

  ```
  ogr2ogr -f FlatGeobuf countries.fgb countries.json
  ```

- **Anwendungen:**
  - OpenLayers, Leaflet, GeoServer (WFS output format), QGIS
- **Programmiersprachen:**
  - JavaScript, TypeScript, C++, C#, Java, Rust

FlatGeobuf Demo



- 11GB Census-daten
- https://flatgeobuf.org/examples/leaflet/large.html

Spaltenorientierte Daten / Arrays / GPU

Spaltenorientierte Speicherformate:

- **Parquet**, Orc, RCFile
- Diskussionen:
  - geopandas/geo-arrow-spec
  - opengeospatial/cdw-geo

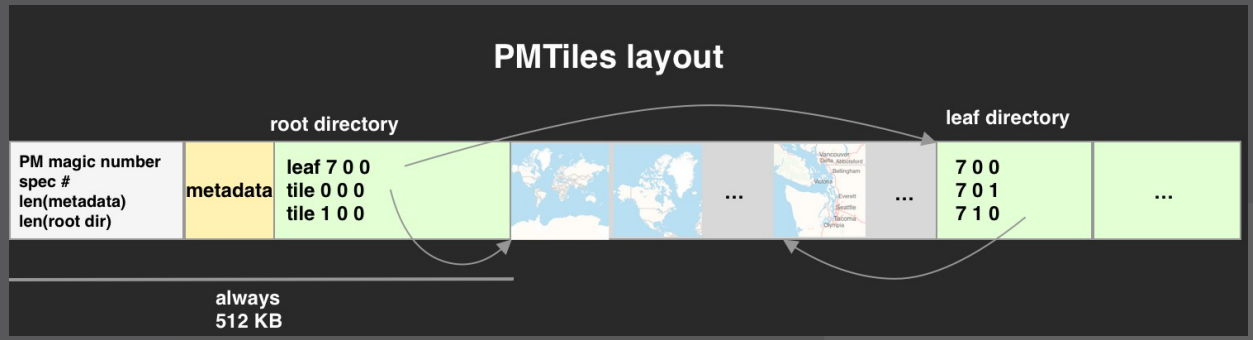
Zarr: Chunked, compressed, N-dimensional Arrays

- **https://zarr.readthedocs.io/**

GPU-Accelerated Spatial Data: cuSpatial

Kachel-Archive

- **PMTiles: Single-File Archivformat für Kachel-Pyramiden**



- **Vorteile:**
  - Raster- und Vektorkacheln
  - Optimiert für HTTP Range Requests
  - Performanter Filezugriff ähnlich MBTiles
- **https://github.com/protomaps/PMTiles**
- **Alternativen: TileBase, Cloud optimized tar**

Point Cloud Daten

Entwine Point Tile (EPT)

- Octree-basierte Speicherung von LAZ Files
- JSON Metadaten

COPC – Cloud Optimized Point Cloud:

- Metadaten in LAZ Header integriert
- Single File oder externe Octree-Speicherung
- https://copc.io/
- Demo: viewer.copc.io

Zusammenfassung

- Raster → COG
- Vektordaten → FlatGeobuf
- Spaltenorientierte Daten → (Parquet)
- N-Array-Daten → Zarr
- Kacheln → z.B. PMTiles
- Point Clouds → COPC

Danke!

Pirmin Kalberer
@implgeo