Testbed-12 Vector Tiling Engineering Report

Table of Contents

1. Introduction . . . 7
1.1. Scope . . . 7
1.2. Document contributor contact points . . . 7
1.3. Future Work . . . 7
1.4. Foreword . . . 7
2. References . . . 8
3. Terms and definitions . . . 9
3.1. vector tile|tiled vector|vectile . . . 9
3.2. slippy map . . . 9
4. Conventions . . . 10
4.1. Abbreviated terms . . . 10
5. Overview . . . 11
6. Status Quo & New Requirements Statement . . . 12
6.1. Status Quo . . . 12
6.2. Requirements Statement . . . 12
7. Solutions . . . 13
7.1. Targeted Solutions . . . 13
7.2. Recommendations . . . 13
8. Vector Tiling Approaches . . . 14
8.1. Challenges . . . 14
8.2. Approaches . . . 15
8.2.1. Render-based tiling . . . 15
8.2.2. Feature-based tiling . . . 16
8.3. Industry implementations . . . 16
8.3.1. Mapbox Vector Tile Specification . . . 16
8.3.2. Cesium 3D Tiles . . . 16
8.3.3. Esri I3S . . . 16
8.3.4. OGC CDB . . . 17
8.3.5. Proprietary Implementations . . . 17
9. Data Coherence . . . 18
9.1. Use cases . . . 18
9.1.1. Styling . . . 18
9.1.2. Geometry operations . . . 20
9.2. Approaches . . . 20
9.2.1. Feature access on a data back-end / server . . . 20
9.2.2. Feature assembly information . . . 20
9.2.3. Alternative solutions . . . 20
10. Simplification . . . 21

10.1. Generalization of Lines . . . 21
10.1.1. Preservation of Topology . . . 22
10.1.2. Shared Segment Generalization . . . 22
10.2. Generalization of Areas . . . 23
10.3. Generalization by Transformation . . . 23
10.4. Feature Filtering . . . 23
10.5. Attribute Filtering . . . 23
11. Tiling schemes . . . 24
11.1. WMTS raster tiling schemes . . . 24
11.1.1. GoogleMapsCompatible . . . 24
11.1.2. GoogleCRS84Quad . . . 24
11.1.3. GlobalCRS84Scale . . . 24
11.1.4. World Mercator . . . 24
11.1.5. WGS 84 Geodetic . . . 24
11.2. Vector tiling schemes . . . 24
11.2.1. Choice of projection . . . 25
11.2.2. Maximum level of detail . . . 26
11.2.3. Alternate tiling schemes . . . 26
12. Feature Tiling Strategies . . . 27
12.1. Geometry Handling . . . 27
12.1.1. Coordinates or pixels . . . 27
12.1.2. Geometry types and tiling strategies . . . 28
12.1.3. Assembly . . . 28
12.2. Attribute handling . . . 29
12.2.1. Associate attributes with tiles . . . 29
12.2.2. Provide access to attributes through a database or service . . . 29
12.2.3. Hybrid approach . . . 30
13. Styling . . . 31
13.1. Styling vector tiles . . . 31
13.2. Feature filtering . . . 31
13.3. Tiling and line rendering . . . 32
13.3.1. Feature assembly approach . . . 33
13.3.2. Clipping bounds approach . . . 33
13.4. Tiling and area rendering . . . 35
13.5. Tiling and labeling . . . 35
14. Performance and memory usage . . . 36
14.1. Memory usage . . . 36
14.1.1. Anchor Tiles . . . 36
14.2. Shared Edge Detection . . . 36
14.3. Performance . . . 36
14.3.1. Render-based Tiling . . . 36

14.3.2. Reference Transformation . . . 36

Appendix A: Revision History . . . 37

Appendix B: Bibliography . . . 38

Publication Date: 2017-06-16

Approval Date: 2017-02-20

Posted Date: 2016-12-29

Reference number of this document: OGC 16-068r4

Reference URL for this document: http://www.opengis.net/doc/PER/t12-A008

Category: Public Engineering Report

Editor: Daniel Balog, Robin Houtmeyers

Title: Testbed-12 Vector Tiling Engineering Report

OGC Engineering Report

destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Abstract

This OGC Testbed 12 Engineering Report discusses the topic of vector tiling.

While tiling and the use of multiple levels of details are a proven technique for raster data, it is relatively new for vector data. This is due to the increased complexity for tiling vector data compared to raster tiling. Further, there is a lack of standardization on the topic. Yet vector tiles can provide the same benefits as for raster tiles:

- Services can easily cache tiles and return them upon request, without the need for any additional pre/post processing (assuming no geometry construction is needed in the server). Consequently, clients can request and receive tiles quickly, ensuring better user experience.
- Due to tiled, multileveled data representations, clients can better access the data most suitable for their current map location and scale. This avoids the need to load too much data, which can cause both excessive memory usage and network traffic resulting in reduced overall performance.

An example of vector tiling that illustrates the impact of these benefits is the OpenStreetMap (OSM) data store, which includes over 30 GB of data with worldwide coverage consisting of millions of vector features. Loading and visualizing all the OSM data into an application would either result in a memory shortage or unacceptable performance. By means of vector tiling and the generation of multiple levels of detail, apps using OSM data can load such data sets very efficiently into applications.

This Engineering Report (ER) focuses on the general aspects of vector tiling. One of the main goals is to characterize what vector tiling is and how it can be approached. Highlighted topics include tiling approaches and strategies, tiling schemes, data coherence, simplification, scalability and styling. With respect to tiling schemes, existing standards material related to raster tiling schemes is incorporated to align both topics and to maximize interoperability. This includes the Defence Geospatial Information Working Group (DGIWG) Web Map Tiling Standard (WMTS) profile and the National System for Geospatial-Intelligence (NSG) WMTS profile as defined by the U.S. National Geospatial-Intelligence Agency (NGA).

The topic of implementing vector tiles using a tile encoding / storage format is not covered. A study of implementing vector tiles in OGC GeoPackage is part of a separate Engineering Report, OGC 16-067, that builds on the results of this ER.

Business Value

With an increasing focus on Big Data for geospatial analytics and visualization, having interoperable solutions that are able to efficiently access large datasets in applications is very important.

A proven technique for raster datasets is raster tiling and the use of multiple levels of detail. OGC played an important role in creating and standardizing interoperable formats and services that apply this technique, including OGC WMTS and OGC GeoPackage.

For vector datasets, similar techniques also exist, yet implementation of vector tiling is much less common. However, the OGC membership recently approved CDB as an OGC standard. CDB defines a conceptual model and file system implementation instance for a vector tile and LoD structured data store. CDB has been in operational use for over 10 years. Standardizing this at the OGC level could give a boost to the adoption of vector tiles, leading to interoperable solutions capable of accessing large vector datasets.

What does this ER mean for the Working Group and the OGC?

As one of the leading organizations with respect to geospatial standardization, OGC has successfully released a number of standards related to raster tiling:

- OGC WMTS, to exchange raster tiles in networked environments;
- OGC GeoPackage, to persist raster tiles;
- OGC CDB, to persist a tiled vector data store.

These standards have been embraced and widely adopted by the industry and community. A logical next step is to define a general approach for the vector tiling use case. By means of standardization, the OGC can play an important role into increasing the adoption of vector tiling beyond the modeling and simulation community and improve the interoperability between industry and community solutions.

How does this ER relate to the work of the Working Group?

This ER relates to a number of Working Groups:

- WMS 1.4 SWG (includes WMTS): WMTS is OGC’s tiling web service for raster data and will be looked at in the context of researching a consistent tiling scheme for raster and vector data.

GeoPackage SWG: The topics discussed in this ER define the foundation of the Engineering Report on a vector tiling implementation in GeoPackage, OGC 16-067.

- CDB SWG: The CDB SWG is focusing on the evolution and extension of the CDB standard to meet requirements beyond its original focus on the DoD modeling and simulation community.
- WFS/FES SWG: WFS is OGC’s web service for exchanging vector data. This ER discusses the same topic but using a tiled approach.

Keywords

ogcdocs, testbed-12, vector, tiling, GeoPackage, WMTS, CDB

Proposed OGC Working Group for Review and Approval

GeoPackage SWG, WFS SWG.

-

Chapter 1. Introduction

1.1. Scope

This OGC Testbed 12 Engineering Report discusses the topic of vector tiling. One of the main goals of this report is to characterize what vector tiling is and how it can be approached. Highlighted topics include tiling approaches and strategies, tiling schemes, data coherence, simplification, scalability and styling. With respect to tiling schemes, existing material on the topic related to raster tiling schemes is incorporated to align both topics and to maximize interoperability; this includes the DGIWG WMTS profile and the NSG WMTS profile (developed by NGA).

The topic of implementing vector tiles using a tile encoding / storage format is not covered. A study of implementing vector tiles in OGC GeoPackage is part of a separate ER, the OGC Testbed 12 Vector Tiling Implementation Engineering Report, OGC 16-067, that builds on the results of this ER.

No Change Requests have been revealed or defined in the context of this document.

1.2. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Table 1. Contacts

Name	Organization
Robin Houtmeyers	Luciad
Daniel Balog	Luciad
Frederic Houbie	Luciad
Peter De Maeyer	Luciad

1.3. Future Work

This Engineering Report provides a general overview on the topic of vector tiling. The authors expect that the resulting information is incorporated in future vector tiling standardization work within OGC, possibly in the context of an implementation in OGC GeoPackage.

1.4. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 2. References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- OGC 06-121r9, OGC® Web Services Common Standard

NOTE: This OWS Common Standard contains a list of normative references that are also applicable to this Implementation Standard.

- OGC 15-113, Volume 1: OGC® CDB Core Standard: Model and Physical Data Store Structure
- OGC 16-070, Volume 4: OGC® CDB Best Practice use of Shapefiles for Vector Data Storage
- OGC 07-057r7, OGC® Web Map Tile Service Implementation Standard
- OGC 06-042, OGC® Web Map Service Implementation Standard
- OGC 09-025r2, OGC® Web Feature Service 2.0 Interface Standard
- NGA Standardization Document NGA.IP.00 2016-02-24, National System for Geospatial-Intelligence (NSG) Web Map Tile Service 1.x.x Implementation Interoperability Profile (2016-02-24)
- Mapbox Vector Tile Specification
- Cesium 3D Tiles Specification
- Esri Indexed 3d Scene Layer (I3S) specification
- AIXM 5.1 - XML Schema (XSD)

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard [OGC 06-121r9] shall apply. In addition, the following terms and definitions apply.

3.1. vector tile|tiled vector|vectile

packet of geographic data, packaged into a pre-defined roughly-square shaped "tile" for transfer over the web.

3.2. slippy map

term referring to modern web maps which let you smoothly zoom and pan around (the map slips around when you drag the mouse)

Chapter 4. Conventions

4.1. Abbreviated terms

- COTSCommercial Off The Shelf
- NSG(NGA’s) National System for Geospatial Intelligence
- WFSWeb Feature Service
- WMSWeb Map Service
- WMTSWeb Map Tiling Service

Chapter 5. Overview

This Engineering Report discusses the general aspects of vector tiling, unrelated to a particular implementation. A candidate vector tiling implementation is within the scope of the related Testbed 12 Vector Tiling Implementation Engineering Report OGC 16-067, which discusses an implementation of vector tiles for the OGC GeoPackage.

Chapter 6 starts with a definition of the status quo and the requirements related to a general vector tiling study. Chapters 7 and 8 continue with a high-level overview of the targeted solutions and possible approaches to address vector tiling. The remainder of the document delves deeper into specific aspects of vector tiling, such as data coherence (Chapter 9), geometry simplification (Chapter 10), tiling schemes (Chapter 11), tiling strategies (Chapter 12), styling (Chapter 13) and performance and memory guidelines (Chapter 14).

Chapter 6. Status Quo & New Requirements Statement

6.1. Status Quo

Various tiling techniques have been applied to raster(ized) data for over a decade and have been widely adopted by the geospatial world, both at a standardization and implementation level. Examples within the OGC include the Web Map Tiling Service (WMTS) and GeoPackage standards. More recently, the candidate I3S and 3d-tiles submissions are specifications for implementing vector tiles for the streaming of 3d content for visualization engines.

Tiling applied to vector data is relatively new compared to raster data, but the topic has received increasing attention over the past several years. Perhaps best known to GIS developers is the public specification for vector tiles developed by Mapbox, with several implementations in both COTS and open-source components.

6.2. Requirements Statement

When discussing raster tiling, it is fairly clear how this technique is approached. Essentially, the gridded raster data values are subdivided into roughly square-like tiles and multiple levels of detail are generated by means of sampling and interpolation. The main topics to be considered are the tiling scheme and the format in which the tiles are stored.

For vector tiling, any technique is more complex. Compared to a raster data value, there is no single way to associate a vector feature with a tile. Similarly, creating multiple levels of detail for a vector feature is more complex compared to raster data sampling & interpolation.

Although vector tiling specification and implementation work is ongoing in the geospatial world, there is no general overview of vector tiling aspects and approaches, independent from a storage format. Therefore, the main requirement to be addressed by this ER is characterizing what vector tiling is and how any solution can be approached.

Chapter 7. Solutions

7.1. Targeted Solutions

Multiple solutions can be defined to address vector tiling, each with its own characteristics, advantages and disadvantages. Within this Engineering Report, we primarily focus on approaches and solutions that are based on the use of vector tile pyramids, consisting of multiple levels of detail and the use of tiles. Chapter 8 identifies the main envisioned approaches.

When dealing with a general study on the handling of large vector datasets, other solutions can be identified in addition to vector tile pyramids, such as the use of spatial indexes (to accelerate access to a vector dataset) and the use of multi-resolution vector features (i.e., providing a specific geometry for a particular level of detail). These solutions are considered out of scope for the Testbed 12 vector tiling study.

7.2. Recommendations

This Engineering Report documents that there is a wide range of aspects and possible approaches related to implementing vector tiling. Clearly there is no single, one-size-fits-all solution. Instead, any solution depends on the envisioned use case(s). Regardless of the implementation approach, vector tiling provides the same benefits as raster tiling.

With an increased focus on handling large datasets and the need for interoperability, the authors recommend further investigating a path of standardization for a possible OGC vector tiling model. The Testbed 12 Vector Tiling Implementation Engineering Report, OGC 16-067 discusses an initial implementation using OGC GeoPackage as data container. While vector tiling in a GeoPackage was stated as Testbed 12 requirement, it is recommended that compatibility with any of the existing OGC standards - OGC GeoPackage, OGC web services, OGC CDB, etc. - be taken into account. As such, a standardized vector tiling model should not be tied to a single data container or service. The GeoPackage-based implementation illustrates this goal by means of a successful integration attempt using an OGC WMTS instance as a service to serve and connect to the same vector tiles stored in OGC GeoPackage.

Chapter 8. Vector Tiling Approaches

For raster data, the tiling approach is pretty clear and consistent: the raster data values are split up according to a tile grid pattern, and multiple levels of detail are generated by combining and resampling detailed tiles into less detailed tiles. In case of vector data, the tiling approach is less straightforward. This chapter starts with a section on challenges that can be identified when discussing vector tiling. Although many solutions and variations can exist in practice to address vector tiling, we identify two major approaches:

- Render-based tiling: focused on visualization of the vector features.
- Feature-based tiling: focused on maintaining integrity of vector features for storage and analytics.

Both approaches are discussed in subsequent sections. This is followed by a high-level overview of industry implementations.

Before continuing, it is important to highlight that this Engineering Report explicitly focuses on turning vector features into tiles. In a broad sense, vector tiling could also be interpreted and implemented as a spatial index, in which a tiled, multileveled structure (e.g., an R-tree) is used to enhance rapid access to a vector feature dataset and/or data store. Although the topic of indexing may be mentioned, it is not discussed in detail and considered to be part of a future, dedicated spatial index study.

8.1. Challenges

Vector tiles share similarities with raster tiles. Consequently, it is useful to look at raster tiles when addressing vector tiles, as a number of problems and solutions are applicable to both topics. However, there are also a number of challenges unique to vector tiles:

- **Data coherence:** For some applications, there can be a need to assemble a feature that crosses multiple tiles back into its original form. To support this use case, a tile should contain all necessary information to assemble a feature. The need to assemble a feature depends on the targeted use cases.

Chapter 9 further discusses the topic of data coherence.

- **Defining multiple levels of detail:** For raster data, multiple levels of details can be generated by means of sampling and interpolation. Although various techniques and settings can be used in practice, the process is fairly easy and it can be done automatically without any input from a user. For vector data, the story is more challenging. In general, two techniques can be identified:
  - Feature filtering: Leaving out features on lower levels of detail, based on feature characteristics (e.g., local roads do not need to be present at a country level scale). The process of leaving out features is fairly simple. The difficulty lies in determining the business rules to decide when to filter. This information typically needs to be provided by a user who knows the dataset (i.e., its use cases, properties, …)
  - Feature generalization: reducing the amount of detail needed to represent a feature’s geometry, based on a map scale. Algorithms exist to support this, but a reoccurring difficulty

is preservation of the data’s topology.

Both topics are further discussed in Chapter 10.

- **Tile sectioning:** For raster data, a tile can be made by storing the raster data values contained within the tile boundaries. For vector data, there is no single way to associate a feature with a tile. As features inevitably cross tile boundaries, one needs to define an approach to associate features with tiles. An easy solution could be to associate a feature with a single tile with which it overlaps the most. However, this clearly reduces one of the primary benefits of tiling: to only load data currently needed. A real tiling solution uses tile sectioning, i.e. the splitting of features at tile boundaries. Multiple approaches can be defined for this, depending on targeted use cases and the type of features.

Chapter 12 delves deeper into the topic of feature tiling strategies.

- **Unique feature identification:** During the tiling process, vector features might be split into multiple pieces across tiles and generalized versions of the feature might be introduced. To enable linking everything together, there is clearly a need to uniquely identify the source feature. Preferably, the source dataset includes a unique identifier property for each feature, but this might not always be the case.

8.2. Approaches

8.2.1. Render-based tiling

The render-based tiling approach is mainly focused on high-quality and high-performance rendering of vector data. With this approach, tiles are generated starting from a render-ready version of the vector features. This render-ready version can be seen as a tile with rendering instructions, ready to be submitted to the client device’s rendering engine. For instance, a tile might contain instructions to move to a pixel position (moveTo) or to render a line to another pixel position (lineTo). This approach is somewhat similar to SVG geometry primitives. The data is projected up front. Clients can read the tiles without the need to do any feature geometry assembly, reprojections or other intensive operations.

Compared to rasterizing the vector data and then using raster tiles, the above approach avoids rendering artifacts when rendering tiles at scales other than the available levels of detail. Additionally, it gives users the flexibility to customize the styling of the data client-side. Although the original geometry is no longer present, a tile can offer access to the feature’s properties. If a unique ID is present, it could be used to retrieve a feature’s geometry from a separate service (e.g., an OGC Web Feature Service (WFS)).

From an OGC web services perspective, this best relates to OGC WMTS. This web service is focused on serving tiles for rendering; each tile can be individually interpreted and rendered.

Typical use cases are rendering and slippy map.

The benefits are high-performance and high-quality.

8.2.2. Feature-based tiling

A feature-based tiling approach is mainly focused on giving users efficient access to the source geometry (coordinates) and feature properties in a vector data set, regardless of their physical size and amount of detail. In this approach, features are split into component pieces based upon their geometry and the tile grid pattern used. A resulting tile contains the geometry pieces of the feature that are geographically contained in that tile. Apart from the geometry, the tile offers access to the feature's properties, similar to the render-based tiling approach (either as a link or embedded in the tile). This will be discussed in the Performance and Memory Usage chapter. The tile also offers access to information necessary to reconstruct the feature's original complete geometry. If required, the client has responsibility for reconstructing the features and displaying them in the client's map projection. The tiled geometries may be defined in the data's native reference or reprojected to another reference.

Typical use cases are rendering and analysis.

The benefits are high-quality and the fact that the original feature data are no longer needed.

8.3. Industry implementations

This section gives a high-level overview of public industry and community specifications that support vector tiling. This section also briefly lists proprietary implementations used in the industry.

8.3.1. Mapbox Vector Tile Specification

A popular and widely adopted vector tiling specification is the Mapbox Vector Tile Specification [5]. The approach followed by Mapbox' specification can be classified as render-based tiling.

8.3.2. Cesium 3D Tiles

Cesium 3D Tiles is a public specification to stream 3D content, including buildings, trees, point clouds, and vector data to web applications [6]. Vector data tiles include WebGL calls, classifying the approach as render-based tiling. At the time of writing this ER, the specification has been submitted to the OGC as a candidate Community Standard [7].

8.3.3. Esri I3S

Indexed 3d Scene Layer (I3S) [8] originated from investigations into technologies for rapidly streaming and distributing large volumes of 3D content across enterprise systems that may consist of server components, cloud hosted components, and a variety of client software from desktop to web and mobile applications. A single I3S data set, referred to as a Scene Layer, is a container for arbitrarily large amounts of heterogeneously distributed 3D geographic data. A Scene Layer is structured as a set of hierarchically structured nodes, which are analogous to tiles, and Levels of Detail (LoD). A Scene Layer is characterized by a combination of layer type and profile to fully describe the behavior of the layer. At the time of writing this ER, the I3S specification has been submitted to the OGC as a candidate Community Standard [9].

8.3.4. OGC CDB

CDB is an OGC standard defining an open format and encoding for the storage, access and modification of a representation of the natural and built environment for simulation applications [10][11]. CDB makes use of several commercial and simulation data formats that are in widespread use within the simulation industry. For vector data storage, the Esri Shapefile format is used as a best practice [12]. One of the core CDB concepts is the use of a tiled representation of the Earth with multiple levels of detail. Each tile can link to data, such as a Shapefile dataset consisting of vector features. Consequently, CDB can be classified as a variant of feature-based tiling. CDB does not define a stand-alone vector tile format but it does define the means to use tiling and multiple levels of detail with existing vector formats that are unaware of these concepts.

8.3.5. Proprietary Implementations

Next to open specifications listed above, the industry also defined proprietary implementations on the topic. The following is a non-exhaustive list identifying several representative implementations and their classification with respect to the vector tiling approach:

- GenaMap: Developed in 1985, GenaMap's technology is probably one of the earliest solutions that implemented vector tiling. With support for the raw geometry, topology, and a strong focus on maintaining precision and accuracy, the used approach can be classified as feature-based tiling. The implementation used the concept of an "edge node" to identify where a feature geometry split across adjacent tiles.
- Google Maps & Apple Maps: the well-known Google Maps & Apple Maps use vector tiles for a few years to have high-quality graphics at arbitrary zoom levels. Their approaches can be classified as render-based tiling.
- Luciad: Since 2011, Luciad has provided a vector tiling solution in its LuciadFusion product. The solution uses a feature-based tiling approach, with support to automatically generate multiple levels of detail - using techniques such as geometry simplification and feature filtering.

Chapter 9. Data Coherence

Within a vector tiling context, data coherence refers to the ability to assemble and access a feature when reading vector tiles, similar to how the feature was represented before tiling and the possible generation of multiple levels of detail. To support this use case, a vector tile should contain the necessary information to perform the assembly. The actual need to access an assembled feature depends on the targeted use cases. This chapter gives an overview of use cases that benefit from feature assembly. Additionally, it discusses approaches to implementing feature assembly.

9.1. Use cases

Use cases that benefit from data coherence mostly relate to (complex) styling and operations that require the feature's native geometry.

9.1.1. Styling

- On-path labeling of lines crossing tiles. An example is street name labeling following the geometry of street features, as illustrated in the screenshot below.

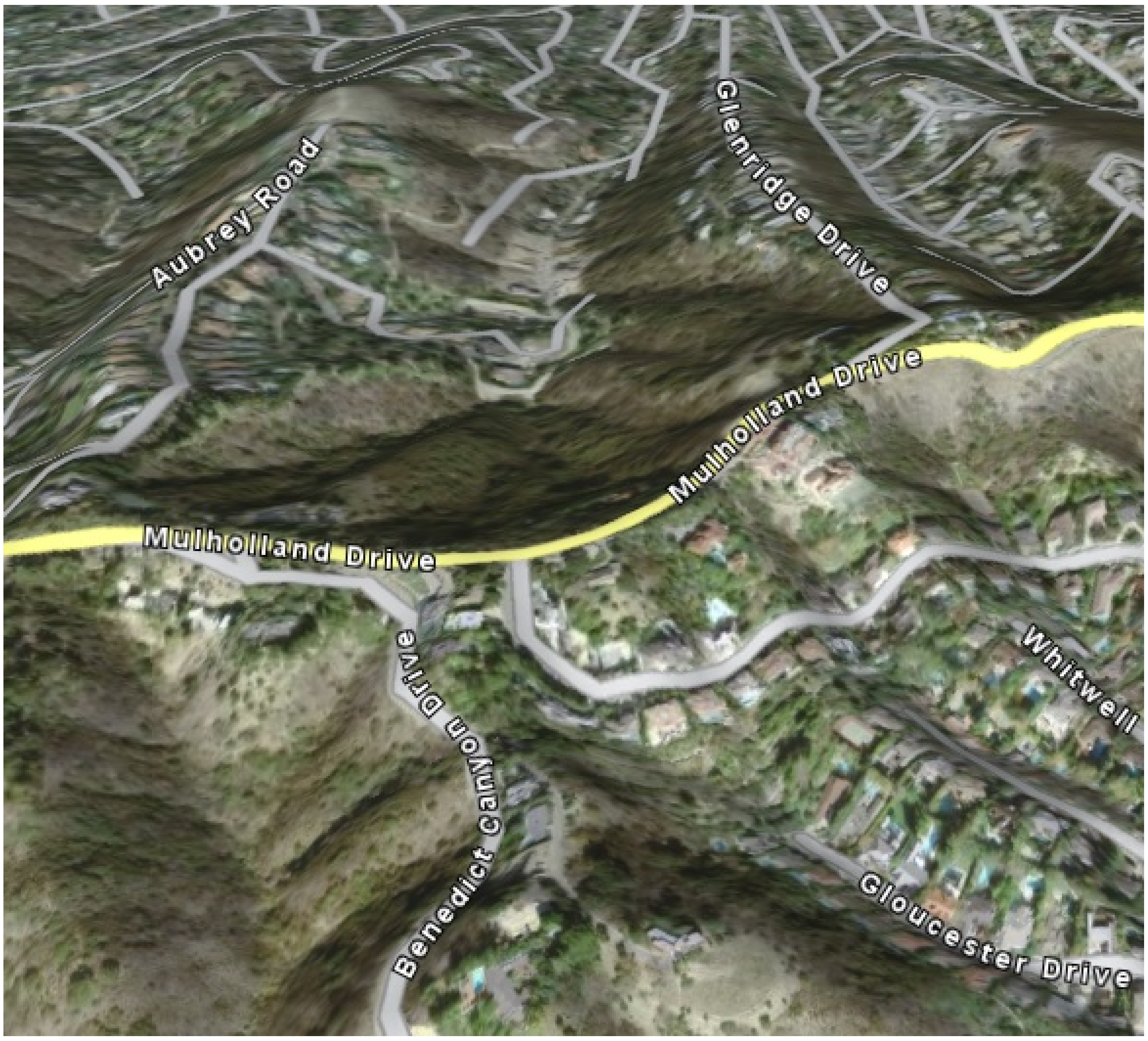

Figure 1. On-Path Labeling

.

- Complex strokes and/or fills. The screenshot below shows an example of pattern-based strokes commonly used by the military symbology standard MIL-STD 2525c.

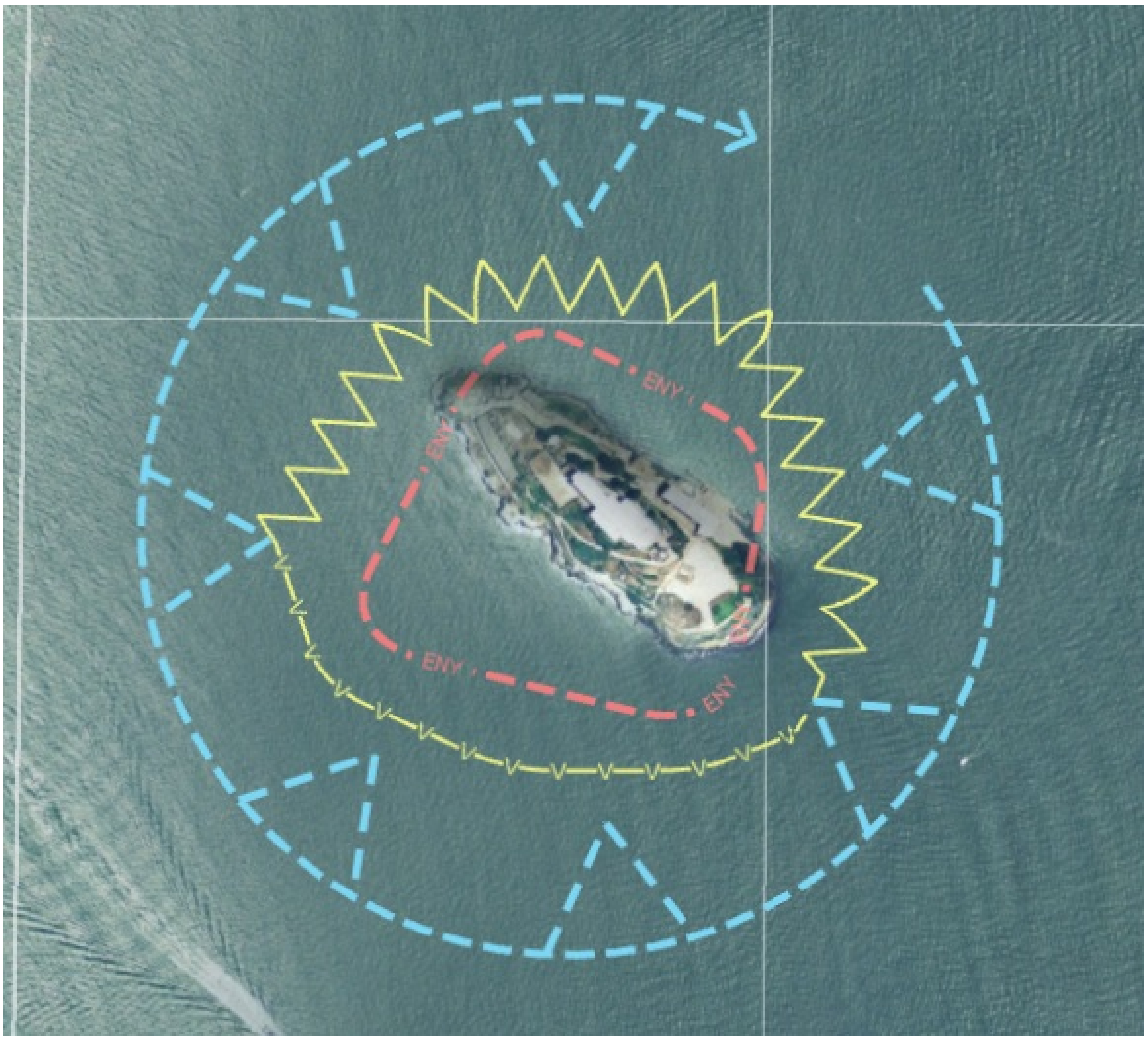

Figure 2. Complex Stroking

- Consistent selection of a feature while panning / zooming. For instance, a selected feature should remain selected when more detailed geometries are loaded for the feature while zooming in; the screenshot below shows this by means of a selected road (orange line) at multiple zoom levels.

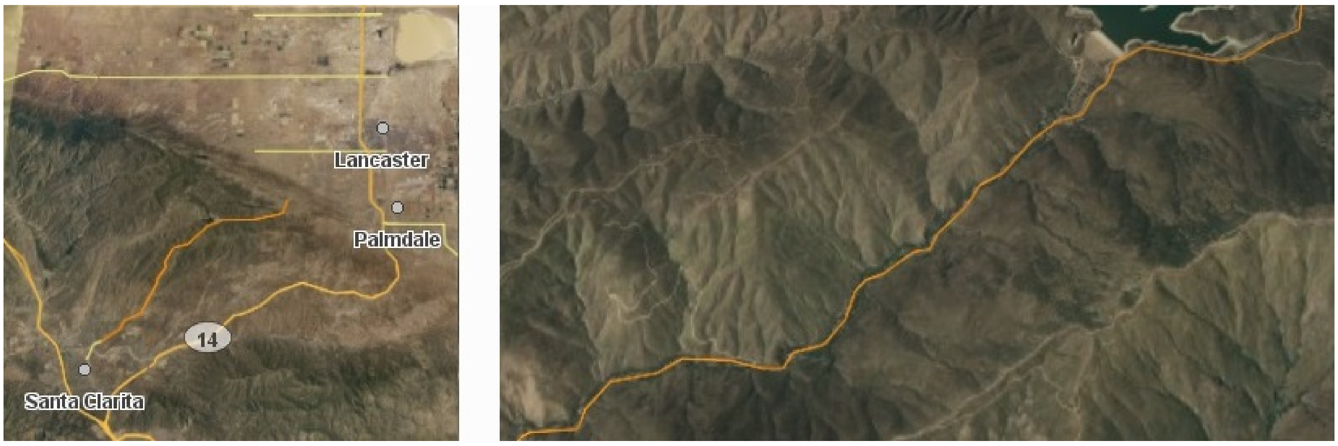

Figure 3. Consistent Selection

9.1.2. Geometry operations

- Routing: Compute the shortest path between two locations in a network, such as a road dataset.
- Intersections: Compute intersections between the geometries of two features.
- Topological operations: Compute the topological relationship between two features (e.g., disjoint, contained in, etc.).

9.2. Approaches

Multiple approaches can be defined to support data coherence in combination with vector tiling. The following sections briefly introduce a few of them.

9.2.1. Feature access on a data back-end / server

An easy approach to provide feature access is to store a link or id in the vector tile, which can be used by an application to request the native feature from a data back-end / server. This approach is ideally suited for a client/server environment in combination with render-based tiling: the render-based tiles are sufficient for standard visualization purposes, while the link / id can still be used to retrieve the underlying feature, if needed. From an OGC perspective, this corresponds to the goal of the GetFeatureInfo request available in the OGC WMS and WMTS standards: give the user the ability to retrieve additional information about a selected location on a map / tile.

9.2.2. Feature assembly information

For the case of feature-based tiling, data coherence can be supported by storing the necessary information in the tile to assemble the feature in an application. This also avoids the need for an external connection to retrieve the feature. The information to be stored should enable an application to differentiate between geometry natively available in the feature (possibly simplified) and geometry introduced by the tiling.

9.2.3. Alternative solutions

While the approaches listed above provide a solution to achieve data coherence, they either require accessing the feature on a server or the ability of an application to assemble a feature (which can be computationally intensive for large datasets). If complex styling is the primary use case (on-path labeling, complex stroking, etc.), an alternative solution consists of generating vector tiles with bounds slightly larger than the tile bounds and rendering them with a clip on the tile bounds. Chapter 13 discusses this approach in practice.

Chapter 10. Simplification

Vector tiling inherently includes the aspect of having multiple levels of detail - either physical or logical, depending on how data are organized in the physical data store and what metadata is available. Having multiple levels of detail makes sense in case of features that only need to be made available at certain map scales and / or in case of features with very detailed geometries that are too complex to be shown at small map scales. For example:

- A local road does not need to be visible at small map scales; apart from the risk of loading too much data, it would also introduce visual clutter.
- A lake boundary with 100000 points does not need to be represented with that degree of detail at small map scales. Transferring and attempting to use such complex features on a lightweight client would cause excessive memory usage and a reduction of overall performance.

One easy approach to address this is to leave out a feature class (layer), such as leaving out local roads at lower levels of detail. Another approach is generalization: i.e. the reduction of a feature's geometry by removing coordinates - for instance, by reducing the complexity of the geometry of the lake boundary consisting of thousands of coordinates into a simple polygon consisting of only a few tens of coordinates at lower levels of detail. Again, this is a reduction of data complexity and volume based on some set of user specified rules.

The following sections further discuss generalization techniques and aspects.

10.1. Generalization of Lines

Line generalization is achieved by reducing the number of coordinates needed to represent a line geometry. One of the most well-known algorithms to do this is the Ramer-Douglas-Peucker algorithm. Its purpose is, given a curve composed of line segments, to find a similar curve with fewer coordinates. The algorithm defines 'dissimilar' based on the maximum distance between the original curve and the generalized curve (i.e., the Hausdorff distance between the curves). The generalized curve consists of a subset of the points that defined the original curve.

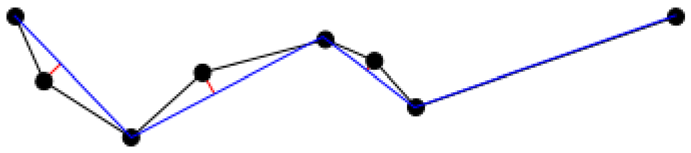

Figure 4. RDP Algorithm

This principle is illustrated in the image above. The black dots represent the points of the original line geometry. The blue line shows the result after applying the Ramer-Douglas-Peucker algorithm with a relatively small Hausdorff distance. This distance is visualized by means of the red lines, resulting in the removal of three points in this example (the 2nd, 4th and 6th counting from the left).

When generating multiple levels of detail, line generalization algorithms such as Ramer-Douglas-Peucker can be an effective way to reduce the geometry complexity at small map scales. By varying the Hausdorff distance at each map scale, one can adjust the complexity to match the scale.

The following paragraphs discuss a few additional complexities that might arise during a line generalization process.

10.1.1. Preservation of Topology

While generalizing the geometry of a line can be relatively easy for an isolated feature, special attention needs to be paid to the topological relationship between geometries in case of multiple features. Even within a single feature, topological relationships between the individual line segments exist that can make the line generalization process more complex. The images below illustrate this with an example.

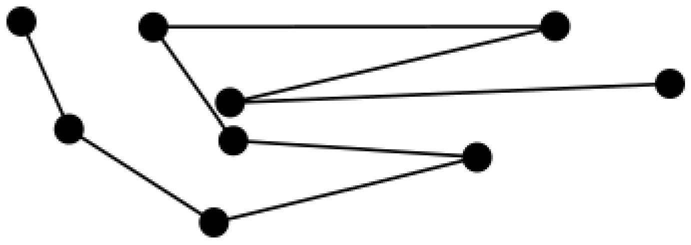

Figure 5. Topology Image 1

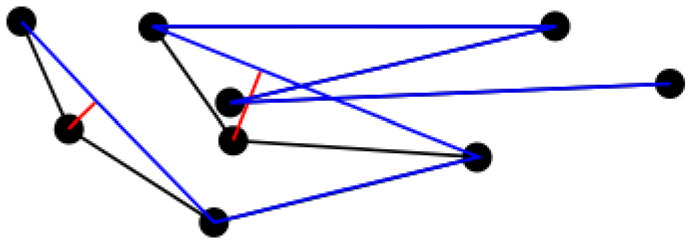

Figure 6. Topology Image 2

The first image shows a variation of the previous feature, represented by a black line. The second image shows a generalized version, represented by the blue line. The generalized version includes an intersection between two line segments of the feature which was not present before, hence a change in the topological relationship. The same can happen between segments that belong to different features. A common real-world example is an island located close to a coast; regardless of the generalization, both features should remain distinct at all scales.

Extensions to Ramer-Douglas-Peucker and alternative algorithms are available to also take into account topological relationships between line segments within a single feature or across multiple features.

10.1.2. Shared Segment Generalization

A special case of topology preservation is shared segment generalization. Consider two features with adjacent line segments - for instance, two country border features. To make sure that the adjacent line segments remain adjacent, they need to be generalized in the exact same way. Although topology preserving line generalization techniques could be used to address this, another approach is to represent the adjacent segments as a single set of segments. Apart from solving the topology preservation issue, it also reduces the amount of data needed to represent a data set.

10.2. Generalization of Areas

The generalization of areas relies on the same techniques discussed for the generalization of lines. Additionally, areas with holes can benefit from leaving out small holes at lower levels of detail, such as a small lake that disappears at a country level map scale.

10.3. Generalization by Transformation

Next to line and area generalization, it is also possible to approach generalization by transforming a geometry into a more simple geometry. One easy but effective example is the reduction of a line or polygon geometry to a point. This can be useful when a feature's presence still needs to be visible at small map scales but when it is too small to be visualized with any detail.

10.4. Feature Filtering

While the generalization of geometries is an effective solution to keep a feature visible at lower levels of detail with an appropriate amount of detail, it can also be appropriate to leave out features entirely at a certain scale. A common example is omitting local roads at country level scales. Chapter 13 further discusses this topic in the context of styling.

10.5. Attribute Filtering

For features with an extensive list of attributes, it can be worthwhile to filter out unnecessary attributes - again possibly related to a scale range. An example from the aeronautical domain can be found in AIXM, an aeronautical data exchange format. A feature in AIXM represents an aeronautical entity, such as an airport, a runway, an airspace, etc. In addition to aviation domain-specific attributes, an AIXM feature can also include an extensive set of metadata attributes modeled using the ISO 19115 metadata standard, typically used to describe the data origin and quality. Depending on the use case, having the ISO 19115 metadata in the resulting tiled feature may not be required. Next to attribute filtering, there are also other techniques to optimize the handling of the attributes of features. For more discussion on this topic, please refer to Chapter 12.

Chapter 11. Tiling schemes

Essential to tiling is the choice of a tiling scheme. This topic has been thoroughly researched for raster tiling, with many tiling schemes available in practice. Therefore, when investigating vector tiling, consideration of raster tiling schemes and review of their applicability to vector tiling makes sense. This chapter discusses a number of raster tiling schemes and alignment with vector tiling schemes. From an OGC interoperability perspective, we focus on the OGC WMTS standard and a number of standardized WMTS tiling schemes.

11.1. WMTS raster tiling schemes

11.1.1. GoogleMapsCompatible

Probably the most widely used and one of WMTS' well-known tiling schemes is the GoogleMapsCompatible tiling scheme [13][14]. This approach specifies a Web Mercator map projection (EPSG:3857), a square layout, a fixed tile size of 256 x 256 pixels and 18 zoom levels.

11.1.2. GoogleCRS84Quad

The GoogleCRS84Quad is defined by WMTS as a well-known tiling scheme. Apart from the projection this approach is equivalent to the GoogleMapsCompatible tiling scheme. It uses a geodetic reference instead of a Spherical Mercator projection.

11.1.3. GlobalCRS84Scale

The GlobalCRS84Scale tiling scheme is defined by WMTS as a well-known tiling scheme. This approach specifies a geodetic tile pyramid (CRS:84), a square layout, a fixed tile size of 256 x 256 pixels and 21 zoom levels.

11.1.4. World Mercator

The World Mercator tiling scheme is defined by NGA's NSG WMTS 1.0 Implementation Profile [15]. That profile specifies a Mercator map projection (EPSG:3395), a square layout, a fixed tile size of 256 x 256 pixels and 25 zoom levels.

11.1.5. WGS 84 Geodetic

The WGS 84 Geodetic tiling scheme is defined by NGA's NSG WMTS 1.0 Implementation Profile. This approach specifies a geodetic tile pyramid (EPSG:4326), an equirectangular layout, a fixed tile size of 256 x 256 pixels and 24 zoom levels.

11.2. Vector tiling schemes

The raster tiling schemes listed in the previous section all use a quad-tree structure, in which each tile is the parent of 4 other tiles.

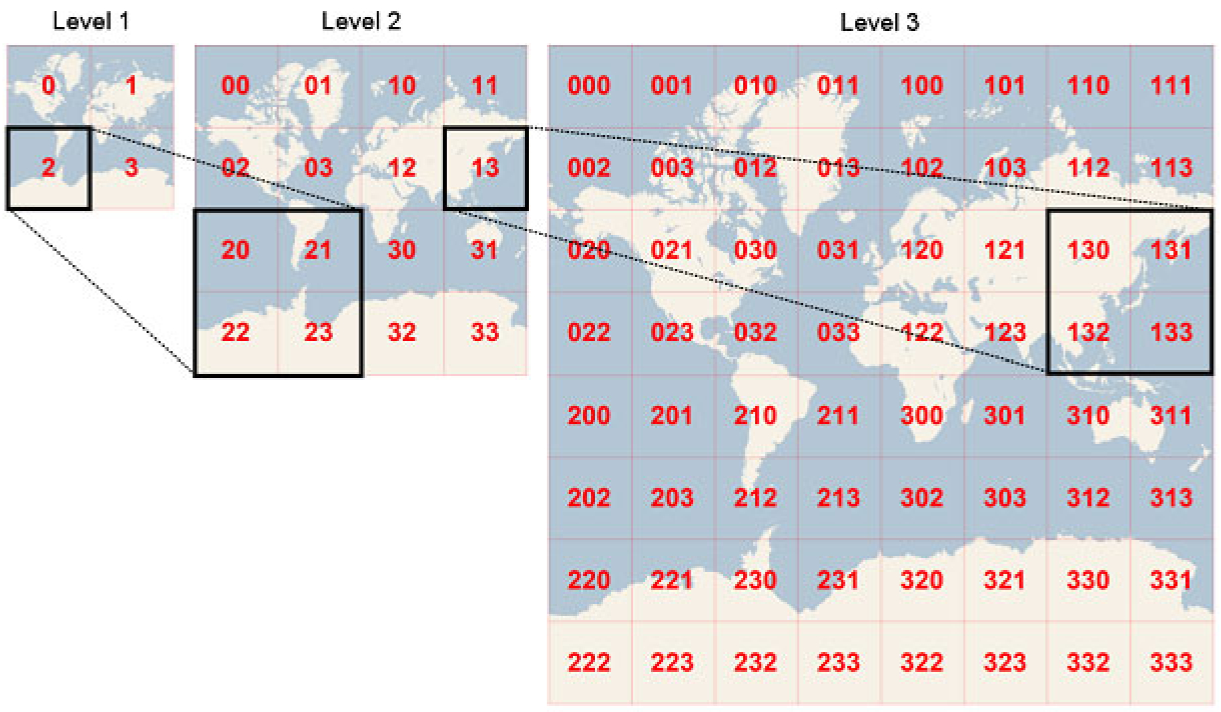

Figure 7. Quad Tree

This is no coincidence, given its simplicity and ease to work with in applications. In general, the same tiling structure can be used for vector tiling. The main difference is the format of the data contained in a tile, which will no longer be a bitmap image as used in raster tiling schemes. The following sections discuss a number of aspects that influence the decision for a vector tiling scheme.

11.2.1. Choice of projection

The choice of a coordinate reference system (CRS) is an important question and is impacted by the intended use of the vector tiles. If visualization in a fixed projection is the main use case, then choosing a tiling scheme built around that projection makes sense. An example is the Mapbox Vector Tile Specification which specifies the Web Mercator projection. In use cases where there is no single projection specified and reprojection is needed, and/or if feature processing is required, then choosing a tiling scheme built around a geodetic reference (in case of geodetic source data), not specifically tied to a projection, would be preferred.

Relation to the tiling approach

The choice of CRS (projection), and more specifically the ability to reproject data afterwards, is also related to the tiling approach. In case of render-based tiling, the features are turned into pixel-based rendering instructions tied to a given projection, making it difficult to support or use a different projection. This is in contrast to feature-based tiling, which keeps the feature's coordinates, and is therefore represented in the source CRS. Note that in case of feature-based tiling, it can also be interesting to transform the data into another reference during the vector tile production - not a view (pixel-based) reference but a geographical reference. For instance, if you know upfront that your data will always be visualized using a Mercator projection, then transformation to Mercator upfront would make sense.

11.2.2. Maximum level of detail

By definition, a tile pyramid has a set of levels or tile matrices, each representing a set of tiles at a given scale. In case of raster data, the deepest level of detail typically corresponds to the scale needed to be able to represent the original raster data. For example, in the case of raster data with 10 cm/pixel and NGA’s World Mercator projection, you need 22 levels to represent the detail down to the original source. This is based on the quad-tree tiling scheme’s zoom level scale set table ([15], Table 8). With less zoom levels, you will not reach a 10 cm/pixel resolution.

The choice for a maximum level of detail is different for vector data, as there is no concept of resolution or pixel density. Additional levels of detail are mainly needed if something changes to the state of a feature. Examples of changes:

- If a feature is added (e.g., local roads added at a high level of detail)
- If a feature’s representation is changed (e.g., country border gets more accurate at a high level of detail)

If there are no further changes beyond a given level of detail, there is no enhanced value in adding more levels of detail, since vector data can be rendered with infinite precision - in contrast with raster data. An application can simply use the latest level of detail and use the associated features, regardless whether the current map scale is at a higher level of detail.

11.2.3. Alternate tiling schemes

The tiling schemes discussed above are all built-around a similar quad-tree structure. Although a quad-tree tiling scheme structure is the most common one in practice because of its efficiency and simplicity, other tiling scheme structures exist that focus on specific use cases, such as:

- K-d trees: In a K-d tree, a tile has 2 children, typically not equal in size. The split axis can be chosen based on data statistics (e.g., median). This structure focuses on non-uniformly distributed data sets.
- Loose quadtrees: This is a variant of a quadtree, in which the 4 child tiles of a parent tile can have overlap. This structure can be useful to avoid feature splitting for features with a relatively small spatial extent, such as buildings.
- Irregular trees: Irregular trees step away from the quad-tree concepts of having square tiles and a fixed relationship between parent and child tiles. Similar to K-d trees, this structure focuses on non-uniformly distributed data sets.
- Octrees: In an octree, a tile has 8 children. While quadtrees are ideal for 2D data, octrees can be useful for 3D data, such as 3D buildings.

As an example, the I3S specification supports the use of multiple indexing schemes. I3S is agnostic with respect to the model used to index objects/features in 3D space. Both regular partitions of space (e.g. “Quadtrees” and “Octrees”) as well as density dependent partitioning of space (e.g. R-Trees) are supported by I3S.

Chapter 12. Feature Tiling Strategies

When a tiling scheme has been chosen, the next logical step is to define an approach to assign features to tiles. In this chapter, we give an overview of possibilities and points of attention. In general, a vector feature consists of two items:

- geometry
- attributes

The two sections below respectively discuss both topics in more detail.

12.1. Geometry Handling

The handling of a vector feature's geometry is an important part of a vector tiling approach. The approach used in an implementation should take into account all aspects of a solution:

- Targeted use case: Are the vector tiles primarily intended for visualization? Is reprojection support needed? Do consumers need to be able to change the styling client-side? Is the geometry needed to support analysis / computational tasks?
- Complexity for the producer: How difficult is it to produce vector tiles? Is feature assembly information required?
- Complexity for the consumer: How difficult is it to consume vector tiles? Do clients need to assemble features?

12.1.1. Coordinates or pixels

A major decision is the choice between tiling on a pixel level or at a coordinate level. This directly relates to the two vector tiling approaches discussed at the beginning of this Engineering Report, render-based tiling and feature-based tiling.

When tiling on a pixel level, vector features are rendered during the vector tile production process. This is done to determine the instructions needed to render the data in an application. These render instructions should be directly usable by a rendering engine in a consumer (client or (rendering) service). Consequently, this is the easiest approach from a consumer's point of view.

The SVG format is a good example of how such rendering instructions appear. The instructions are typically in the form of *moveTo pixel position A*, *lineTo pixel position B*, etc. The resulting rendering instructions are clipped according to the tile boundary and stored in a tile.

When tiling at the geometry and coordinate level, vector features are tiled using their original geometry (transformations can be optionally used, as indicated in the previous chapter). Consumers need to transform them to pixel coordinates, using a projection of choice. Consequently, this increases the complexity in the consumer (client or (rendering) service), compared to pixel level tiling. It does have the advantage of being projection-independent, which can be important for certain use cases.

12.1.2. Geometry types and tiling strategies

Depending on the geometry type, multiple tiling strategies can be defined.

- **Point**: A point is the most trivial case, as it doesn’t need any geometry splitting and it can be easily assigned to one tile. A special case is a point falling precisely on a tile boundary, requiring a decision to assign the point to one of the adjacent tiles.
- **Lines**: A line can cross tile boundaries so all intersecting tiles need to know (at least) about the contained part of the line.
- **Areas**: Very similar to lines, with the added complexity that tiles might not only intersect with the area, they can also be completely contained in the area.
- **Parametric curve-based shapes (circle, arc, ellipse, …)**: Can be treated similar to areas, but the tile sectioning can be done more efficiently: in general, a curve-based shape has a pretty compact representation (e.g., a point and a radius for a circle), so it makes sense to keep the geometry as a whole without any tile sectioning.
- **Point clouds**: A variation of the point case, without any additional complexity with respect to tiling. Since a point cloud is 3D, an octree tiling schema could be of use.
- **3D objects**: 3D objects can be handled atomically, in which they are assigned to one particular tile, or as a mesh consisting of vertices (points), edges (lines) and faces (polygons), in which they can also be split across tiles. For objects with a relatively small geographic extent (e.g., a building), it will be easier for a consumer to use one tile. Similar to point clouds, an octree tiling schema could be of use.

Topology Preservation

When performing tile sectioning, it is generally important to preserve a feature’s topology. For instance, to ensure that an area-shaped feature converted to vector tiles and subsequently read by a consumer remains an area-shaped feature - and not a feature composed of geometry pieces introduced by the tile sectioning algorithm.

Points on the edge

Special cases can arise for geometries that completely overlap with a tile edge - e.g., for lines and points. The chosen vector tiling approach should define how to handle such cases: store the features across the adjacent tiles or assign the features to one single tile.

12.1.3. Assembly

For the render-based approach, assembly of features is generally not required. The targeted use case is high-performance & high-quality visualization. To maximize the performance, consumers should be able to directly read and visualize a tile’s contents.

For the feature-based approach, assembly of features may be required to support operations that require the representation of the original feature - aligned with the targeted use cases of feature-focused vector tiling. The ability to assemble a feature contained in multiple tiles does make it considerably more complex for a producer and consumer:

- Producers need to store all necessary information in a tile to enable consumers to assemble a feature
- Consumers need to read all tiles with which a feature overlaps and consequently assemble the feature.

When visualization is the primary use case, special clipping techniques can be used by both the raster and feature-based tiling approaches to avoid tile-related rendering artifacts with common styling, without the need to have the full geometry. The Styling chapter will further discuss this.

12.2. Attribute handling

When dealing with vector features, one is inherently also dealing with vector features' attributes. Depending on the targeted use case, multiple approaches can be defined to handle a feature’s attributes in a vector tiling context.

12.2.1. Associate attributes with tiles

One solution is to include attributes in the vector feature tiles. This enables direct use of the attributes in applications, to support a variety of use cases:

- Attribute-based styling (e.g., fill color depending on an attribute value)
- Labeling (e.g., street names)
- Filtering (e.g., filter airspace reservation data based on time)
- Information consulting

An attention point for this approach is the amount of storage needed to store the attributes. A simple but effective solution is the introduction of an anchor tile.

Anchor tile

An anchor tile represents a tile that can be used to store unique information related to a feature. Each tile that intersects a feature contains a link to the anchor tile. This enables users to find it and access its information. As such, the anchor tile can be used to store the attributes of a feature. Although theoretically not required, an anchor tile is typically the tile with which the feature overlaps the most. This ensures that the tile is typically already available for the user that is visualizing the feature.

12.2.2. Provide access to attributes through a database or service

Depending on the vector feature’s data type, the number of attributes can be significant. For instance, in case of the AIXM 5.1 aeronautical data format, a vast number of attributes can be defined for a feature, including an entire ISO 19115 metadata description. Depending on the targeted use case and available transfer bandwidth, there might be too much information to store in a tile. A tile with too much information can reduce the performance benefits related to vector tiling. An alternative solution is to keep the attributes in a separate data store and make them available upon request - potentially through a web service. A tile (possibly only the anchor tile) could include a link to this store, so that users can access it. Conceptually, this approach is comparable to the OGC WMS GetFeatureInfo request, which enables users to request more information for a given location on a map retrieved via the WMS.

12.2.3. Hybrid approach

Hybrid approaches can be defined that use a combination of attribute storage in the tile and on the server. This enables users to make the most important attribute information readily available in the tile, while less frequently used information can be requested from a server.

Chapter 13. Styling

Rendering vector data requires applying styling to the vector features' geometries: line strokes, area strokes and fills, icons, labels, and so forth. In this chapter, we look at the relation between styling and vector tiles: the possibilities to style vector tiles, the relation with filtering and the impact of vector tiling on the styling process.

13.1. Styling vector tiles

In the context of vector tiling, styling can be coupled or decoupled from the vector tile data, depending on the tiling approach.

With a render-based tiling approach, the styling description is an integral part of the vector tiles - either in an embedded form or through a reference. The focus is on high-performance, high-quality vector data rendering, so applications consuming the data need to be able to immediately render vector data contained in a tile. Additionally, the approach gives data providers the possibility to associate the preferred style with the data, thus ensuring consistent maps on different platforms and applications.

With a feature-based tiling approach, styling can be completely decoupled from the vector tile. The vector tile essentially consists of the feature’s geometry and its attributes (or a way to access them). Although styling information can be included, it can equally be left out and replaced by a stand-alone feature styling definition unaware of the vector tile structures. An example of the latter is an OGC Symbology Encoding file.

13.2. Feature filtering

To optimize a vector tile’s content, filtering of vector features is preferably done during the tiling production process. However, since the underlying tile pyramid has a discrete set of levels of detail, data might be filtered at a slightly different scale than intended.

For instance, assume two levels of detail in a random quadtree tile pyramid, one at 1:50000 and the other at 1:25000. If feature data needs be filtered out at a scale between both map scales, it will only be effective at 1:50000. This is illustrated in the image below, in which a filter for local roads is associated with scale 1:40000.

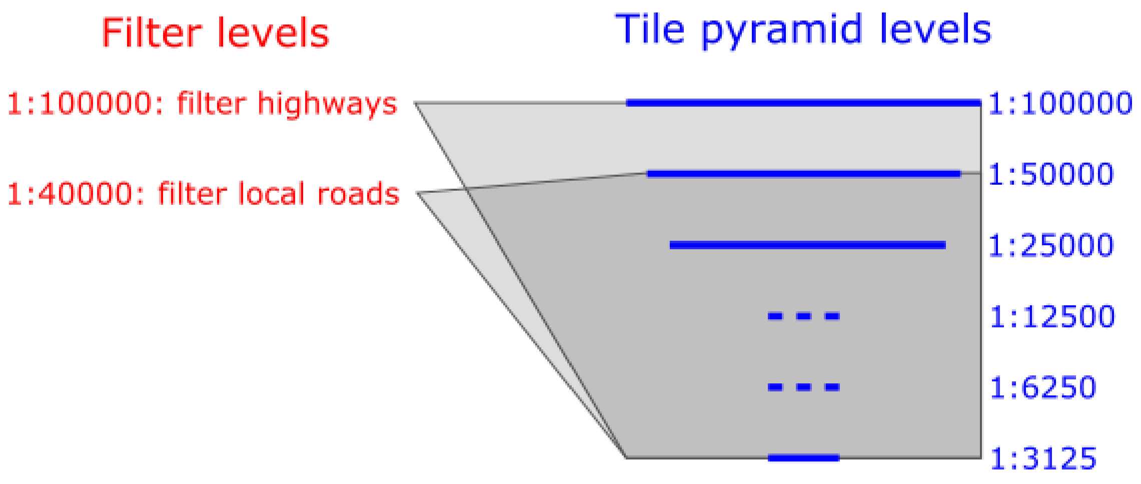

Figure 8. Scale Filtering

This can be solved by applying the same filtering rule when consuming the vector tiles. Applied to the example, the application will read the feature data at 1:50000 and additionally filter out features based on the map scale defined by the filtering rule.

One way to implement this in a standardized way is to rely on OGC's Symbology Encoding (SE) specification. The following OGC SE snippet defines a styling rule to render local roads at or below 1:40000, corresponding to the previous example. During the vector tile production process, this style definition could be used to filter out the data at the next scale in the tile pyramid, 1:50000. When the data is accessed in the client, the client can use the style both to find the right stroke width and color and to hide it between 1:50000 and 1:40000 (it will automatically be hidden above 1:50000, since it is no longer part of the data at or above that scale.)

```
<FeatureTypeStyle xmlns="http://www.opengis.net/se" version="1.1.0">
  <Rule>
    <Filter xmlns="http://www.opengis.net/ogc">
      <PropertyIsEqualTo>
        <PropertyName>type</PropertyName>
        <Literal>local</Literal>
      </PropertyIsEqualTo>
    </Filter>
    <MaxScaleDenominator>40000</MaxScaleDenominator>
    <LineSymbolizer>
      <Stroke>
        <SvgParameter name="stroke-width">0.5</SvgParameter>
        <SvgParameter name="stroke">#809bc0</SvgParameter>
      </Stroke>
    </LineSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

13.3. Tiling and line rendering

Ideally, the styling of data is not impacted by tile boundaries. However, without any measures, tile boundary artifacts will quickly show up.

As an example, assume a road feature dataset. To improve the visibility of the road on maps, a common road styling rule includes two strokes, one for the interior and one for the boundary. The figure below illustrates this styling for a single road feature. In practice, this is achieved by rendering the feature twice, once with a thick stroke, representing the color for the road boundary, and once with a thin stroke, representing the color for the road interior.

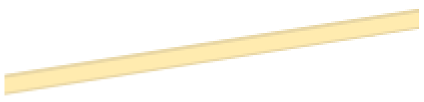

Figure 9. Road Style

When vector tiling is applied to this dataset, it can happen that the road is split into two or more fragments, each representing the part of the road that overlaps with a given tile. When the twofold styling is applied to the resulting tiled features, the result will include an unwanted line at the tile boundary, as shown in the figure below.

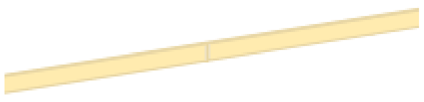

Figure 10. Road Style - Tiled

There are a number of approaches to solve this:

13.3.1. Feature assembly approach

In case of the feature-based tiling approach, by reassembling the geometry pieces across the different tiles, the feature's original geometry can be reconstructed. By applying the style on the resulting feature, there will be no impact anymore from the tile boundaries.

13.3.2. Clipping bounds approach

Another solution that works both with the feature-based and rendering-based tiling approaches is to define the vector tiles with a boundary slightly larger than the bounds of the tile. When the data gets rendered, the rendering engine needs to apply the tile bounds as clipping bounds, hence avoiding the boundary rendering artifact.

The figure below illustrates this by showing the rendering of the feature's segment that is located in the left tile. The grey line represents the tile boundary. Without any clipping bounds the result would look like the rendered road in this figure.

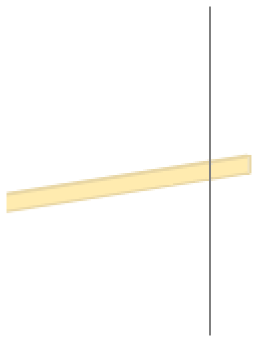
Figure 11. Road Style without Clipping Bounds

By applying the clipping bounds, the piece right from the grey line will not be rendered, as shown in the figure below.

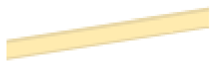
Figure 12. Road Style with Clipping Bounds (right)

By applying the same clipping technique to the other tile, as illustrated in the figures below, the end result will be as in the initial figure, without any tile boundary lines.

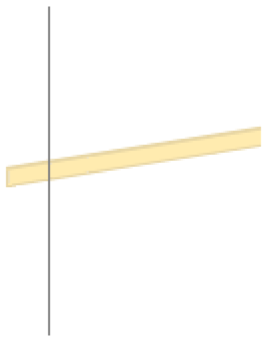
Figure 13. Road Style without Clipping Bounds

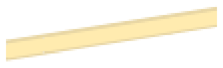
Figure 14. Road Style with Clipping Bounds (left)

13.4. Tiling and area rendering

Compared to line rendering, area rendering adds the concept of a fill. Fill operations on a polygon can result in similar rendering artifacts. In general, the same techniques for line rendering - feature assembly and clipping bounds - can be applied to area rendering. Special care is however needed in case the area fill is defined by a pattern (e.g., hatched style). By definition, the feature assembly approach will work as if there were not tiles. In case of the clipping bounds approach, special measures might be needed to make sure that the resulting, visually merged patterns are aligned with each other.

13.5. Tiling and labeling

Labels can exist in many forms - single-line, multi-line, rotated, curved (e.g, street name within a curved road feature) with an anchor point that defines its location. A potential issue in combination with vector tiling is the effect of having multiple labels in the result. This can happen if the split vector features are labeled separately. Although label decluttering techniques could mask this at large map scales, cluttering and overlap of labels will eventually becoming visible when zooming in and / or around tile boundaries.

Similar as with line and area rendering, the feature assembly approach solves the above issues automatically, since each feature will only be labeled once. Other solutions specific to labeling are:

- Track labeled feature segments: If each feature’s segment has a unique feature id, it is possible for a rendering engine to only label a feature once - e.g., the first encountered segment. Although effective, it has as drawbacks (1) the requirement of having additional labeling logic specific for vector tiling and (2) a potential unnatural label location, which could result in seeing the tile boundaries throughout the label placement.
- Define label positions upfront: In case of render-based tiling, it can make sense to define label positions upfront, embedded in the screen coordinates of a vector tile. With this approach, the vector tile production process can make sure that features get a predefined label position, which is then to be used by the rendering engine in the component that reads and renders the vector tiles.

Chapter 14. Performance and memory usage

This chapter discusses a few aspects and design considerations related to the performance and memory usage of vector tiles. Most concepts have been introduced in other chapters. This chapter revisits them in the light of performance and memory usage.

14.1. Memory usage

14.1.1. Anchor Tiles

When the requirement is to store the attributes of features directly into the tile (for fast access by an application), consider the concept of an anchor tile allowing the storage of the feature attributes only once. In case of large datasets with many attributes, this can result in a significant reduction of required memory, both in the application reading the tiles and in the data container storing the tiles. In the application, this typically outweighs the cost of one extra request to get the anchor tile.

14.2. Shared Edge Detection

In case of vector data with shared edges (e.g., country borders), it can be interesting to detect these shared edges and store their geometry only once. Besides a positive impact on memory consumption, it also benefits simplification and the preservation of topology.

14.3. Performance

14.3.1. Render-based Tiling

With render-based tiling, a tile is optimized for high-performance visualization: an application reading tile needs to apply the rendering instruction to have the tile contents displayed on the screen. Performance-wise, this typically outperforms a feature-based tiling approach, since there are no geometry assembly and/or reprojection steps involved.

14.3.2. Reference Transformation

In case of render-based tiling, a feature’s geometry is already projected to screen coordinates, avoiding the cost of any additional projection logic. In case of feature-based tiling, this is not the case. However, if there is a commonly used target reference, it does make sense to transform a feature’s geometry to this reference. This avoids the need for an application to do the transformation on-the-fly for each tile. For instance, in case of source data defined using one or multiple references and an application that always uses Mercator, it is recommended performance-wise to transform the data to Mercator during the vector tile production process.

Appendix A: Revision History

Table 2. Revision History

Date	Release	Editor	Primary clauses modified	Descriptions
April 14, 2016	.1	R. Houtmeyers	all	IER
September 30, 2016	.8	R. Houtmeyers	all	draft ER
October 31, 2016	.9	R. Houtmeyers	all	addressed comments from internal reviews

Appendix B: Bibliography

[1] Web: Robert Patrick Victor Nordan: An Investigation of Potential Methods for Topology Preservation in Interactive Vector Tile Map Applications, http://www.diva-portal.org/smash/get/diva2:566106/FULLTEXT01.pdf

[2] OGC 07-057r7: OGC Web Map Tile Service Implementation Standard

[3] OGC 06-042: OGC Web Map Service Implementation Standard

[4] OGC 09-025r2: OGC Web Feature Service 2.0 Interface Standard

[5] Web: Vladimir Agafonkin, John Firebaugh, Eric Fischer, Konstantin Käfer, Charlie Loyd, Tom MacWright, Artem Pavlenko, Dane Springmeyer, Blake Thompson: Mapbox Vector Tile Specification, https://github.com/mapbox/vector-tile-spec

[6] Web, Patrick Cozzi: Cesium 3D Tiles Specification, https://github.com/AnalyticalGraphicsInc/3d-tiles

[7] Web: OGC press release announcing 3D Tiles for consideration as a Community Standard, http://www.opengeospatial.org/pressroom/pressreleases/2466

[8] Web: Esri: Indexed 3d Scene Layer (I3S) specification, https://github.com/Esri/i3s-spec

[9] Web: OGC press release announcing I3S for consideration as a Community Standard, http://www.opengeospatial.org/pressroom/pressreleases/2499

[10] Web: OGC press release announcing OGC CDB as OGC Standard, http://www.opengeospatial.org/pressroom/pressreleases/2488

[11] OGC 15-113, Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure

[12] OGC 16-070, Volume 4: OGC CDB Best Practice use of Shapefiles for Vector Data Storage

[13] OGC 07-057r7 Web Map Tile Service Implementation Standard.

[14] Web: Joe Schwartz: Bing Maps Tile System, https://msdn.microsoft.com/en-us/library/bb259689.aspx

[15] NGA Standardization Document: National System for Geospatial-Intelligence (NSG), Web Map tile Service 1.x.x Implementation Interoperability Profile (2016-05-20)