Jawg Maps

Whitepaper

The White Rabbit and Map Services

Let's face it, this is the best whitepaper name you've seen in a while

Version 1.2

Date	Rev	Editor	Changelog
01/08/2016	0.1	L. Ortola	Initial structure
03/08/2016	0.2	L. Ortola	Redaction of section 1.
11/08/2016	0.3	L. Ortola	Redaction of section 2.
12/08/2016	0.4	L. Ortola	Redaction of section 3.
16/08/2016	0.5a	L. Ortola	Prepare an early Alpha Release.
26/08/2016	0.6	L. Ortola	Added Differential database import content.
07/09/2016	0.7	C. Gayvallet J. Magloire I. Caraiman	Adjustments, Glossary
13/09/2016	1.0 -RC1	L. Ortola	Release candidate
22/09/2016	1.0 -RC2	T. Brown PH. Pillet	Review, corrections
23/09/2016	1.0	L. Ortola	First General-Availability Release
30/08/2017	1.1a	L. Ortola	Following the SOTM Japan, changing so many things to make it cooler and more accessible!
05/09/2017	1.1	T. Brown	Review, corrections
07/11/2019	1.2	J. Magloire	Logo update

Table of Contents

1. Why we're doing this

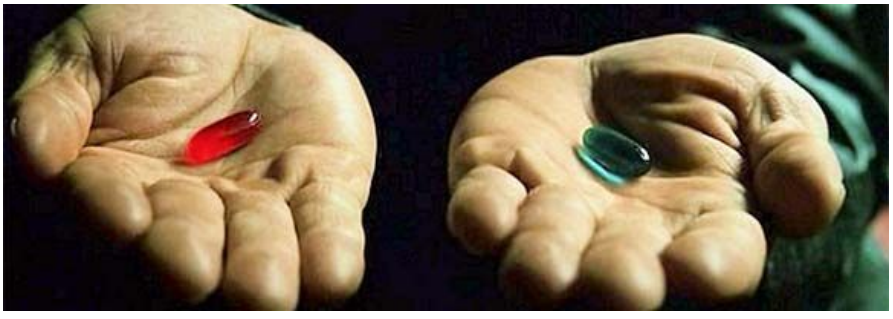
If you are looking to know more about the reproductive cycle of reptiles, this document is probably **NOT** what you are looking for. Today, we are here to talk about **maps**. But before we start, there are a few things we want to confess.

In early 2015, colleagues and I started giving some of our free time to help develop a mobile crowdsourcing app. It was aimed at helping Essonne (France) firemen to collect local AEDs (Automated External Defibrillators) in OpenStreetMap.

Soon after, we started to realize the power of OpenStreetMap and imagined how cool it would be to serve our own maps. We were quite confident geeks, great developers, but knew pretty much nothing about map services. Still, we were bold enough to take on the challenge of setting up our own map server... in a few hours. "How hard can it be, right?"

Those few hours turned into a few days of research, weeks of map server experiments, and lots of lows and highs. At the end, the story turned out great as it lead to the creation of our cool company, Jawg Maps, but let me assure you, the journey was not a fairy tale.
Coffee was spilled, pizzas were ripped apart, it was ugly, and frankly, quite a struggle for us map-noobs to get to the bottom of things.

Years later, looking back at all the awesome things the team has achieved, we wanted to give back what we had been missing at our start: A beginner's guide to map services. A reference document to help get you through the first step. Now, you have a choice:



Source: The Matrix (1999)

You can take the blue pill, and the story ends. You wake up in your bed and believe whatever you want to believe about maps services. Maps will stay this wonderful black-box with cool pins that you can click on.

Or you can take the red pill and we will show you what is behind the curtains. Even better, we will make understanding map services **simple**!

Made your choice yet? Let's go.

Loïc Ortola, CEO

2. Introduction - Acknowledgments

This paper is dedicated to the whole OpenStreetMap community.

Imagine if you had the ambition to create a map of Paris, from scratch.
It would probably look like this:

Your first map of Paris

Although we love contemporary art, it doesn't look like your wonderful map is going to be useful to anyone anytime soon. To draw something on your map, we first need data!
That means taking years to declare every shop, every street, and every stop sign that exists in Paris. A map service cannot exist without such data. This information would be stored in a **GIS** Database, which can contain hundreds of gigabytes of information.

To this day, 4 entities have somehow created such databases. Three of them are closed and non-free (TomTom, Google, Here), while one of them is… open-source.

What? You mean that hundreds of thousand people are walking the streets and trails of the planet, contributing to a dataset and then access is just given away for free? That is right. As of today, thanks to the maturity of the **OpenStreetMap** dataset, this data is accessible out-of-the-box, and is continuously improved by both the community and many companies who were smart enough to understand that this is the future. Map data is becoming a commodity.

We really wanted to take the time to acknowledge the OpenStreetMap community. Neither this whitepaper nor our company would exist without you. We want to stay as close as we can, we promise to do our best to develop great tools to help you in your quest, while pushing the mapping industry a little further.

This paper was inspired by you, and we hope it helps new mappers, companies and developers to get a better introduction to this wonderful universe we are proud to be a part of. Because this document was made by humans for humans, we really tried not to take ourselves too seriously. Reader, beware: you may find silly jokes and weird sentences which prove that we geeks are way better at doing high-tech development than being funny.

The purpose of this document is to provide a basic understanding of the concepts involved for the creation of map services. You will learn the way maps are served and discover the most common vocabulary used to describe maps.
Want to save some trees? Don't print out this document, just enjoy the beautiful digital version.

2.1. Understanding map services

A map service is a system that provides maps. It can be the map of your GPS app, your favorite website, or even the basemap of Pokemon Go.

First things first, let us acknowledge what the definition of a map means in this document. A map is a support on top of which one can display geolocalized assets (Points of Interest, roads, shops, land, water, mountains, etc.). A map semantically represents the surface of the earth, at a certain position and scale.

With the OpenStreetMap database, we already have in our hands the data to draw on our map. The following step is mostly a designer's artwork:
Which color the lake is going to be rendered, what width will the main roads have, how layers are going to be ordered, are bridges supposed to be above or under roads, what about when the road is a 4+ lane highway, etc.
When you do it from scratch, this artwork is a profoundly technical task, as it requires understanding the database bottlenecks that a rendering choice may imply, and weight choices depending on the map scale. Those tricky rules (called **map style**) can then be used as an input to a map renderer, which will turn it into your final map (for instance, an image).

Database bottlenecks? Map renderers? If you're a bit lost don't worry, we'll explain all that in a minute, but for now just think of the challenges of drawing a digital map as pretty much the same as if you were a professional artist painting it on a canvas.
If I tell you "I want a map of the whole world, please draw every house and every shop on it", the work will probably go on for generations before it is done. Moreover, it hardly makes sense to draw a billion houses on a 1x1m canvas (but who are we to judge art?).

The same is true for the map renderer (the painter). To start drawing a map, you will tell him what to draw (I want the hills, major roads, water, city names, gardens, borders), how to draw it (I want the gardens green, the water ocean blue, the hills grey, the roads 4cm wide and orange, etc.), and which area (Paris please).

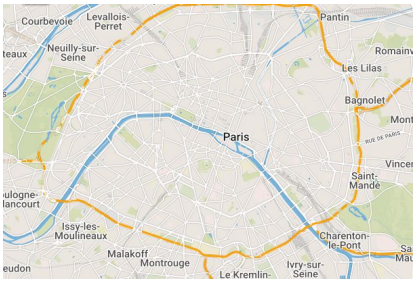
[In 2017, the Jawg team released the Lab, allowing anyone to make custom map-styles in just a couple clicks. We know most of you don't want to struggle fine-tuning your style. Check it out here https://www.jawg.io/lab]

In our case, "maps" can be defined as three things:

- A list of "things" to display: hills, roads, water, street numbers, direction, city names etc.
- A scale (or its digital equivalent, called zoom level)
- A region (referred to as 'bounding box')

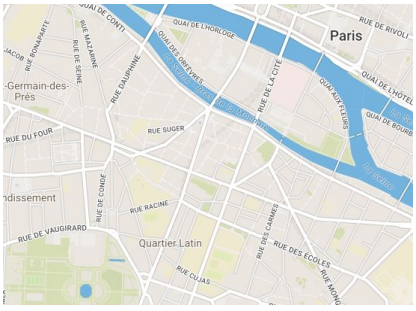
If you want to learn more about the different "things" (called map features) you can put on your map go to OpenStreetMap's Wiki (https://wiki.openstreetmap.org/wiki/Map_Features), but for now we are going to talk about scale.

This is a map of the main streets and roads of Paris, at a scale 1:150.000
(<=> zoom level 12)



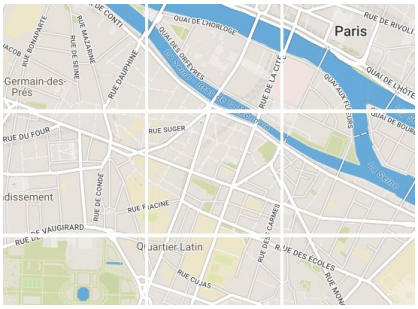
At this scale, giving you a map of the whole world would represent a picture of about 1 Trillion pixels.

Let us get a little closer. At scale 1:15.000 (<=> zoom level 15), the same map of Paris looks like this:



As one image, a map of the world at this scale would now be about 70.4 Trillion pixels, and would probably take tens of days to render.

Therefore, to ease-up both the rendering process AND the data transfer time, maps are rendered and cut into small chunks (called tiles). You may have noticed already that this is what any map actually looks like:



At zoom level 15, we will have over 1 billion tiles of 256x256 pixels instead of 1 tile of 70 trillion pixels.

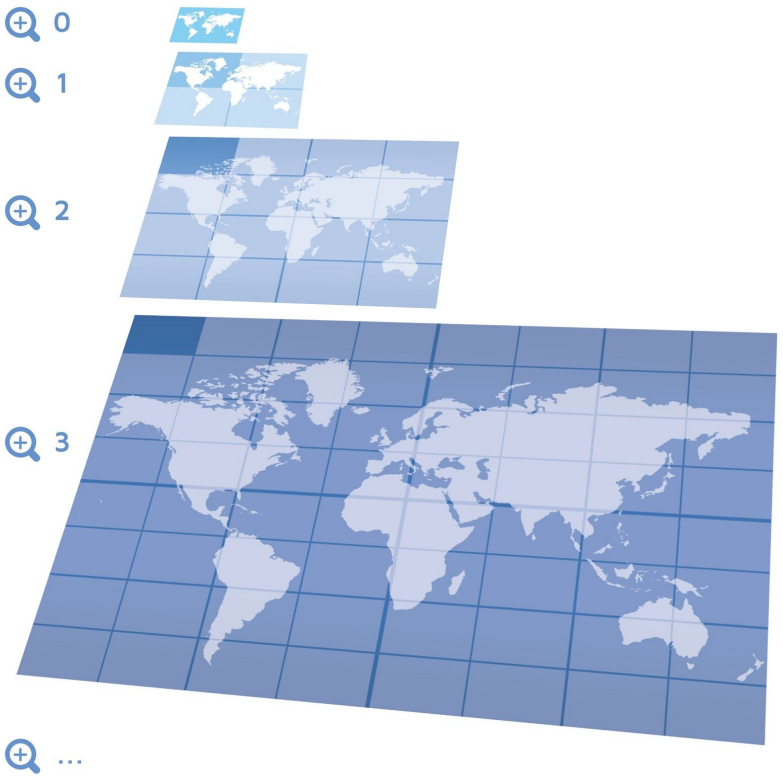
Before understanding how tiles are divided, the concept of scale factor and zoom level must be explained further.
Historically, we have drawn maps at one scale at a time. Your atlas book may represent a map of the world, then multiple maps of the continents, countries… Each at its own scale.

Since digital maps have appeared, the user has the power to change scale (aka: zoom in or out) and display areas with different precisions.
What this means, is that maps propose different projections of a place in the world, with different precisions, and with different amounts of information displayed. Indeed, users are moving their digital maps not over 2, but 3 dimensions, the third one being scale.

Zoom level 0 consists of one image of the world at a scale of 1:500 million.

Zoom level 1 proposes a scale of 1:250 million.
To keep the same level of precision, each “tile” would have to represent half of the region, on each dimension (x and y). This results in the creation of 4 tiles, each representing a quarter of the world.

This goes on and on for every zoom level.



Two points are worth being mentioned:

- A tile will hold a varying amount of information depending on its zoom level. The tile at zoom 0 won't display the roads or the name of your street, because it wouldn't make sense at this scale. This makes the creation of a map-style much harder, although making it right is absolutely critical for performance aspects.
- Many would argue a map server could simply consist of static files, rendered again from time to time. Although this seems appealing, rendering the whole world to the zoom **19** means creating more than **366 billion** tiles. This also poses many challenges regarding data-indexing, storage efficiency, and cache expiry.

Therefore, map services are framed by the following constraints:

- Map renderers use dense GIS databases with complex queries, which are CPU, I/O and memory bound.
- Map styling must be done carefully to limit the complexity of such queries.
- Map rendering is time consuming and CPU-bound. Maps services cannot dynamically render all tiles.
- Map services require the use of caching, which is IO-bound.
- Map services cannot serve 100% cache of the whole world, as this would take too much space, and would be profoundly inefficient on cache-invalidation.
- Cache invalidation strategy should be handled dynamically, not within a global batch.

These constraints demonstrate the obvious need for a distributed software allowing to efficiently serve maps: a map server.

N.B.: Map servers do much more than this, and handle some aspects which were not discussed such as: multi-layer, multi-style, scale factor, resolution factor, sub-regions, raster vs vector tile-rendering,

storage meta-tiling efficiency, memory-caching, map-service distribution and clustering, load-balancing, diff-repository, distributed api checks, stats, metrics, etc…

2.2. Common terms and concepts

2.2.1. Raster vs Vector maps

You may have heard these terms multiple times when talking about maps. But what do they mean really?

When rendering a map we produce “tiles”. Tiles represent a portion of a map, because querying a map globally takes too much time, is too big and cannot be cached. No one told you that a portion of a map was always a picture though (yeah, all this time, we were hiding things from you. Can you believe it?). So far, we have showed you tiles as pictures, but there are other kinds of tiles. Maps can either be:

- An image, or a set of images (called tiles). Those are called **raster-maps**.
- A set of vectors (lines, polygons, dots, text) displayed dynamically on top of a canvas (OpenGl, WebGl, …). Those are called **vector-maps**.

We hope your world didn’t collapse. Here are more details on the pros and cons of each:

Raster-maps are images, generated and rendered by the server (which means client-rendering is not necessary). It is the most common-used technology. They can also be pictures (drone imagery, satellite imagery) which were not generated by GIS data.
The downsides of this technology include the inability to interact with the elements of the map (it won’t be possible to create an interaction with the “Metro station” rendered on the image) and any client-side operation (color-change, layer filtering, etc.). Those interactions are usually provided with vector data displayed as an overlay.
It also gives a pretty “static” navigation. There is one set of tiles per integer zoom, you cannot browse to zoom 16.4 or 17.8. The global experience is usually not as smooth as vector maps.

Vector-maps are also lightweight, generated by the server, although rendered by the client. It is a more recent technology, which allows for more styling abilities and uses the client resources (CPU or GPU) for rendering. The key here is that vector maps are “small databases”. They do not fix the look of your map. It is up to you to choose the style and what you want to display directly in your app. How awesome!
The downside of this technology is that the tools don’t have the same maturity as raster technology and require recent clients (Web browsers with WebGl compatibility, recent Smartphones, etc.). Sometimes, you just can’t use vector tiles (satellite imagery is an image, whether you like it or not).

2.2.2. Pre-rendering

Some tiles (especially between zoom levels 10 and 14) can take tens of minutes to be generated, especially if they are of very dense areas (remember, the painter needs time to find what he needs to paint and to actually paint it). Therefore, it is a common practice to manually trigger the rendering of certain areas to populate the caches automatically. This procedure is called **pre-rendering**. This is great practice although one cannot render the whole world as it would take years.
Therefore, map services are a smart balance between live-rendering and caching.

2.2.3. Meta-tile

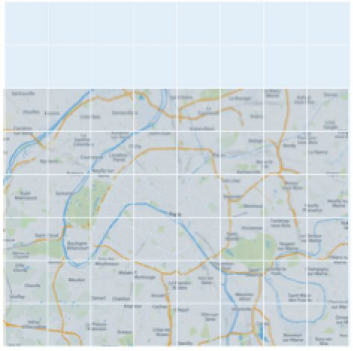
The meta-tile is a common optimization happening on different levels of the map server.

2.2.3.1. Rendering

When a client requests a map view, it usually consists of multiple contiguous tiles (a modern full-screen desktop will typically require a matrix of between 7x4 = 28 and 8x6 = 48 tiles).
If meta-tiling did not exist, 48 cache or render requests would be triggered, which would lead to 48 I/O reads or worse, 48 renders (each one of them using a pool of database connections).
Imagine if your boss started to print 48 pages and told you to go get each page from the printer and bring it to him one at a time. It makes no sense from a human point of view, and the perspective is not different in the digital world.

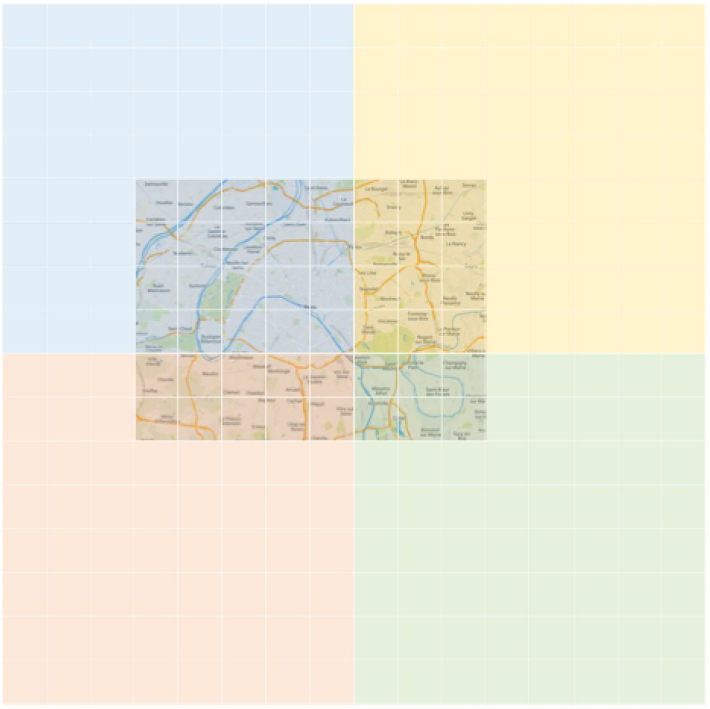
A meta-tile is a way of rendering X tiles at once (8x8 for instance). The upsides are that it saves a lot of unnecessary sockets from being open and optimizes computation.
The downsides are that the meta-tile will obviously take more time to generate than a single tile, and the efficiency (number of tiles used / number of tiles effectively asked) will be between 18% and 75%.

Explanation: When requesting a map view of 8x6 tiles, this is what the best case would look like:



The meta-tile (in blue) rendered 8x8 tiles and the view was luckily fitting perfectly in it.
The display efficiency in our case is 48 / 64 = 75% (which means that 48 tiles among the 64 generated were displayed).

In most cases, a map view has very few chances (less than 5% in a typical map view) of fitting in exactly one meta-tile.



To display a typical map-view, up to 4 meta-tiles can be rendered. This brings the efficiency to 48 / 256 = 18%.

When choosing to use meta-tiles, people will experiment on the meta-tile size (here, 8x8) to find the right balance between query optimization and the resulting efficiency.

Most of the time, this overhead is acceptable and advised, as users usually perform moves around the displayed map-view.

2.2.3.2. Storage

Spoiler alert: This paragraph may contain technical language. Viewer discretion is advised.
Let's do some math:
A map tile of the ocean is usually below 1Kb in size. In ext4 filesystems, the default inode size (aka: the key to retrieve a file on the filesystem) is 128 bytes. The default block size (minimum size occupied by a file. Non-used data will be filled with a sequence of zeros) is 4Kb.
That means that each tile will use at least 4.128 Kb.
For 64 tiles, it would take no less than 264Kb, which is 80% pure loss.
With meta-tiles, the overhead can be drastically reduced to almost no loss, for a small cost of CPU and memory operations. This implies creating a home-made storage system which stores together contiguous tiles within the same file, and allows for easy and fast retrieval.
The second optimization that it does on storage is that getting a meta-tile from cache will require less I/O read operations. Those operations are costly and are usually quite a bottleneck.
[The Jawg meta-tile specification (.mmt) is available on github.com/jawg/meta-tile-spec]

3. Conclusion

What? Are we done already?
*Your feedback matters greatly to us! Send us a Tweet **@jawgio** and let us know what you think*

This whitepaper laid out the main concepts surrounding map services.
We hope it gave you a basic understanding of how maps work, and a quick introduction to the main terms and concepts.

Our teams are working every day to transfer their expertise to others, share their love of maps, and learn more about this rich ecosystem. While this document did provide an overview of the how maps work, it did not attempt to get into all the specifics of map servers. Those specifics will be detailed in a separate paper, and released after this one.

Don’t forget to leave us some feedback on Twitter, and see you soon for another episode!

From the Jawg Team with love.

4. Glossary

GIS: Geographic Information System - is a system capable of incorporating geographical features.

Map style: represents a customized visual display of the elements of the map like text police, roads or buildings.

Meta-tile: Consists of an aggregation of tiles. A meta-tile is used for rendering and storage efficiency.

Pre-rendering: “Offline” rendering in order to pre-cache levels of zoom. Low zoom levels (0-14) are usually pre-rendered.

Raster-maps: Represents a map using arrays of pixel values.

Vector-maps: Represents a map using polygons.