ARCADA

The Server-side of Tiled Raster Web Maps

Benchmarking and Evaluating Static Raster Tile Map Generation Tools for Geospatial Data Processing and Visualization

Alexander Nylund

Degree Thesis

Bachelor of Engineering, Information Technology

2023

Degree Thesis

(Author) Alexander Nylund

The Server-side of Tiled Raster Web Maps. Benchmarking and Evaluating Static Raster Tile Map Generation Tools for Geospatial Data Processing and Visualization

Arcada University of Applied Sciences. Information Technology, 2023.

Identification number:

9190

Abstract:

This study compared the capabilities and performance of available open-source raster tile map generation software alternatives capable of prerendering static raster tiles. Another factor considered was the size effectiveness of the different file formats used on static raster tile servers. This study also briefly compares these static alternatives to dynamically tiled solutions. Using the hyperfine command-line benchmarking tool, the programs gdal2tiles, ctb-tile, gdal2tiles_parallel, and gdal2tiles_mp were tested in Docker containers using the most official docker images for the respective programs. Their capabilities and limitations were discussed based on the available documentation of the different programs. The benchmarking demonstrated that depending on the input and output file formats, the software execution times varied greatly. No program was consistently better than all the others taking into account both the benchmarks and features making the results mostly inconclusive. The key outcome of this study is the foundation it places for further testing, the development of the benchmarking environment and the specific commands that allow someone else to quickly reproduce the results on their own machine.

Keywords:

Gdal2tiles, TMS, XYZ, Web map service, Cesium terrain builder, Geospatial Data Processing

Lärdomsprov

(Författare) Alexander Nylund

Server-sidan av Tiled Raster Web Maps. Jämförelse och utvärdering av verktyg för generering av statiska rasterkartor för geospatial datahantering och visualisering.

Yrkeshögskolan Arcada: Informationsteknik, 2018.

Identifikationsnummer:

9190

Sammandrag:

Denna studie jämför förmågorna och prestandan hos tillgänglig mjukvara med öppen källkod för raster tile kart-generering, som är kapabla att rendera statiska raster tiles. En annan faktor som undersöks är storleken av olika filformat som används på statiska raster tile-servrar. Arbetet jämför också dessa statiska alternativ med dynamiska lösningar som skapar filerna först då de behövs. Med hjälp av hyperfine benchmarking-verktyget testades programmen gdal2tiles, ctb-tile, gdal2tiles_parallel och gdal2tiles_mp i Docker containrar med de mest officiella Docker avbildningarna för respektive program. Deras förmågor och begränsningar diskuterades baserat på den tillgängliga dokumentationen för de olika programmen. Benchmarkingen visade att programmen hade betydligt olika exekveringstider beroende på filformat. Inget program var ständigt bättre än alla andra, med hänsyn till både benchmarkingen och jämförelsen av funktionerna, vilket gjorde resultaten diskutabla. Det viktigaste resultatet av detta arbete är grunden det lägger för ytterligare tester, utvecklingen av benchmarking-miljön och de specifika kommandon som gör det möjligt för någon annan att snabbt reproducera resultaten på sin egen maskin.

Nyckelord:

Gdal2tiles, TMS, XYZ, Web map service, Cesium terrain builder, Geospatial Data Processing

Opinnäyte

(Tekijä) Alexander Nylund

Tiled Raster Web Mapsin palvelinpuoli. Staattisten rasteritiilikarttatyökalujen suorituskyvyn arviointi ja vertailu geospatiaalisen datan käsittelyä ja visualisointia varten.

Ammattikorkeakoulu Arcada: Informaatiotekniikka, 2023

Tunnistenumero:

9190

Tiivistelmä:

Tässä tutkimuksessa verrataan saatavilla olevien avoimen lähdekoodin rasterilaattojen open-source vaihtoehtojen ominaisuuksia ja suorituskykyä, jotka pystyvät ennakkorenderöimään staattisia rasterilaattoja. Toinen tutkittu tekijä oli eri tiedostomuotojen koon tehokkuus staattisilla rasteritiilipalvelimilla. Tutkimus vertaa myös lyhyesti näitä staattisia vaihtoehtoja dynaamisesti luotuihin vaihtoehtoihin. Käyttäen hyperfine-benchmarkingtyökalua ohjelmia gdal2tiles, ctb-tile, gdal2tiles_parallel ja gdal2tiles_mp testattiin Docker-konteinereissa käyttäen vastaavien ohjelmien virallisimpia Docker-kuvia. Niiden kyvyt ja rajoitukset keskusteltiin perustuen eri ohjelmien saatavilla olevaan dokumentaatioon. Vertailu osoitti, että ohjelmiston suoritusajat vaihtelivat suuresti input- ja output-tiedostomuodoista riippuen. Yksikään ohjelma ei ollut johdonmukaisesti parempi kuin toiset ottaen huomioon sekä testitulokset että ominaisuudet joten tulokset olivat suurelta osin kiistanalaisia. Tämän tutkimuksen keskeinen tulos on sen pohja jatkotestaukselle, benchmarking-ympäristön kehitykselle sekä komennot, joiden avulla joku muu voi nopeasti toistaa tulokset omalla koneellaan.

Contents

1 Introduction

Cartographic web services and maps hosted on the web have become increasingly common in today's digital age. These services provide users with access to a wide variety of maps and geographic information that can be used for research, navigation, planning, and many other purposes. These maps are built using prerendered image tiles arranged into a grid. The user can pan and zoom on the map, and the service dynamically fetches the relevant tiles from the server. Although there is ample documentation available for creating the front end of such services, the same cannot be said for the server side, especially the process of generating tiles. The focus of this thesis is to generate these raster tiles. Raster tiles can be used to visualise satellite imagery, aerial photography, infrared satellite imagery, digital elevation models (height maps), and other very large raster images.

This chapter presents the historical background of web maps, the motivations behind this thesis, the research questions, and a brief overview of how raster tile serves work along with the tiled web map standards that are commonly used on the web.

1.1 Background

Web maps have historical roots in GIS applications that, in turn, have older roots. The first GIS systems were developed in the early 60s. The earliest probably being the *Canada Geography Information System*, which was released in 1963 created by Roger Tomlinson and commissioned by the Canadian government. Others soon followed, such as Howard Fishers *SYMAP*, which was created in 1964. One of today's largest GIS software companies, ESRI, was founded in 1969 as a consulting firm that used computer mapping to help land use planers make informed decisions. They also developed ARC/INFO the first commercial GIS product in 1981. (*History of GIS*, 2023) In these early days GIS data was not easily accessible on the web but were often located on in-house mainframes or central file servers that could not be easily shared between organisations. (Sterling & John A., n.d.)

The map usually attributed as the first web map was *Xerox Parc Viewer* Released in 1993. Before long *MapQuest* was released in 1996. It was the first web-based tool that featured zooming and panning of the map, albeit rather slowly. *OpenStreetMap* released its Web map in 2004. In the same year, Google acquired *Keyhole*, *Where2 Technologies* and *Zipdash*.

(Forrest, 2021) These early web maps generated raster maps dynamically on request based on vector data. This meant that they often had to choose between speed and aesthetics because they could only afford to work with minimal amounts of vector data and layers at a time. (Sterling & John A., n.d.)

The following year, 2005, Google Map was released. It featured the *Slippy map* (the map slips around when you drag the mouse (OpenStreetMap Wiki Contributors, 2022)), also called the tiled web map, that all web services would be using in a few years in one form or another (Forrest, 2021). This innovation uses prerendered image tiles arranged in a grid. When the user pans and zooms the map, the service automatically fetches the relevant tiles from the server using AJAX (Asynchronous JavaScript and XML) (Sterling & John A., n.d.).

Some of the most widely used front-end libraries were also soon released, *OpenLayers* in 2006 (Osgeo, 2021), *Mapbox* in 2010 and *Leaflet.js* in 2011 (Forrest, 2021) .

Motivations

Currently, there are several different ways to create a Web map service. The front-end is commonly based on technologies such as Leaflet, OpenLayers, Cesium, or Mapbox.js. While the backend is based on one of the following standards WMS, TMS, WMTS, or XYZ. Some frontend libraries only support certain standards and image formats, while the backend map server or raster tile generation software can only create or serve maps in certain formats and standards.

There seems to be a lack of documentation and literature that lists and compares front-end web map libraries, tile servers, dynamic tile servers, tiled web map standards, and raster image file formats in the context of tiled raster web maps. The few academic resources that could contain good software lists, such as the Springer Handbook of Geographic Information (Masó, 2022) or the (‘Encyclopedia of GIS, 2017), are regrettably behind hefty paywalls and are not easily accessible to a developer.

This makes setting up the backend of a web map service a problem that is difficult to navigate. The reasons why someone would want to set up their own backend could include having their own data, the map tiles they need are very stylised, and require a lot of pre-processing or colour editing not available in services such as Google Maps, OpenStreetMaps, and Maptiler. Other

reasons could be that their data is confidential and only to be used internally. Generating raster tiles using open-source programs and self-hosting the tile server would also to some degree let you avoid being locked to any specific vendor.

1.2 Research questions

This work aims to compare the different tile generation alternatives that exist in the context of creating a self-hosted Web map service.

Research questions include:

- What are the capabilities and limitations of the available open-source raster tile map generation software alternatives?
    - How size effective are different file formats?
    - How do the performances of tile generation programs compare under varying circumstances?
- How does a static alternative compare with a dynamically tiled solution?

1.3 An overview of how a raster web tile server works.

To explain the place of raster tile generation software in the context of a raster tile generation backend service, here is an overview of the processing pipeline for a static raster web tile server.

The first step in the process is to acquire source material, which can be satellite images or maps in any format supported by GDAL.

The second step is optional preprocessing. Here, sharpening and colour edits can be made, as well as stylised modifications to make the map fit the application you are building better or blurring military areas. Other modifications could include cutting out the shape of a municipality, city, or country if the service is intended for a small region.

The third step is the tile generation. Converting the large source file into hundreds of smaller tiled images with overviews for the different zoom levels. The benchmarks in this thesis primarily focus on this step.

The fourth step is to serve the tiles. This can be done with a dedicated tile map server or an ordinary web server, such as Apache or Nginx. It is worth noting that there also exist dynamic tile serving solutions that combine the third and fourth steps as well as dedicated tile server software. More on these in 1.4.

The final step is to consume the service using a JavaScript library at the front end. Some libraries I found while working on this project were Leaflet, Open Layers, Cesium, Mapbox.js, Vts-browser-js, Polymaps, Modest maps, DeckGL, and Titiler Viewer.

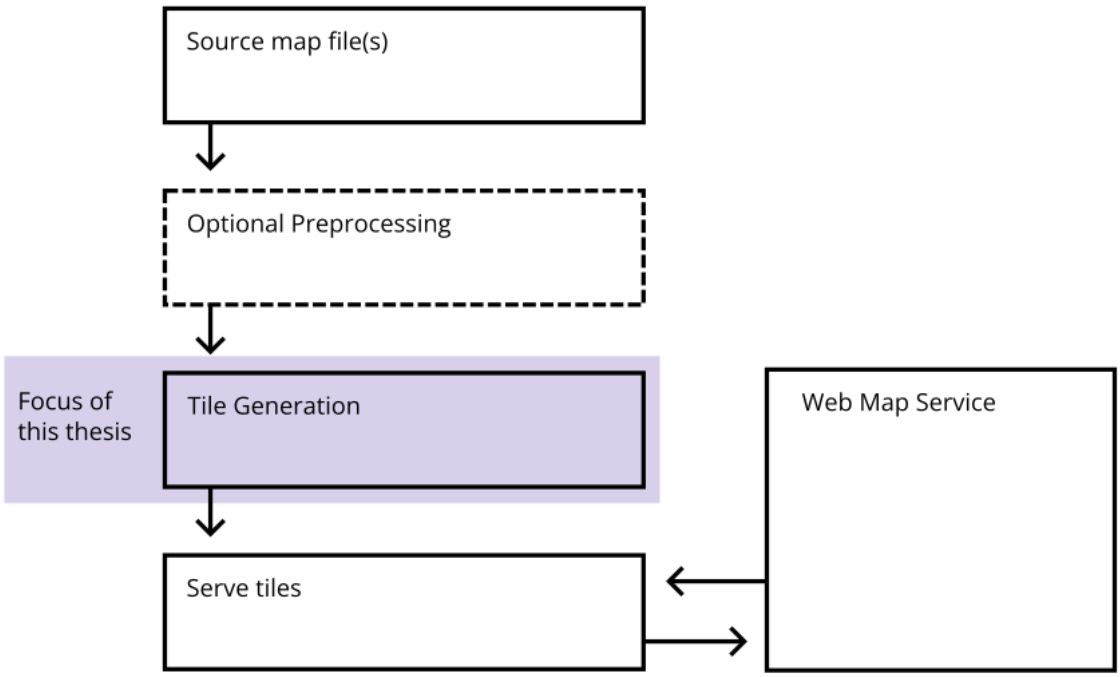

Figure 1. Flowchart of steps in a raster tile map service backend

1.4 Definitions

Slippy Map – is a tiled web map that "slips" around when the user pans the view with their mouse. It is composed of pre-rendered raster tiles arranged in a grid. The user can pan and zoom on the map, and the service dynamically fetches the relevant tiles from the server. (OpenStreetMap Wiki Contributors, 2022)

Raster Tile – an image file, commonly in the JPEG, WebP, or PNG file format usually with a tile size of 256x256 pixels. (But it can also be 64x64, 512x512 or 1024x1024 depending on the standard this raster tile is used in)

CRS – A Coordinate Reference System contains information about the coordinate system, units used, projection information, and datum that holds information about the origin used to place the coordinate system in space. (Wasser, 2019)

GDAL – Geospatial Data Abstraction Library. An open-source MIT licensed translator library for raster and vector geospatial data formats (GDAL/OGR contributors, 2023). (It is used in almost all projects mentioned in this thesis in one form or another)

OGC – Open Geospatial Consortium, an international organisation that maintains standards for geospatial data. (*About - Open Geospatial Consortium*, 2023)

OSGeo – Open Source Geospatial Foundation, a non-profit organisation whose mission is to foster the global adoption of open geospatial technology. (*About OSGeo - OSGeo*, 2023)

1.5 Tiled web map standards

A variety of tiled web map standards, specifications, protocols, and conventions exists for serving tiled web maps. I will refer to all of them as standards for the sake of simplicity, as they all serve the same purpose within the context of this thesis.

Not all frontend JavaScript libraries for interactive maps support the display of all tiled web map standards, and not all tile generation software can generate tiles in all standards. This makes the supported tile map standards an important consideration when deciding how to build a backend for an interactive web map.

Standard	Defined by	Suitable for web use
WMS	OGC	No
WMS-C	OSGeo	No
TMS	OSGeo	Yes
WMTS	OGC	Yes
XYZ	-	Yes
QuadKeys	Bing	Yes, but outdated
COG	Pending OGC standard?	Yes

Table 1. Tiled web map standards (COG Contributors, 2022; Open Geospatial Consortium, 2023b, 2023a; OSGeo, 2012b, 2012a; Schwartz, 2022)*)*

WMS (Web Map Service) and WMS-C (WMS Tile Caching) are mentioned here because they are official standards that could technically be able to serve tiled web maps. WMS-C has been superseded by TMS and WMTS, according to (OSGeo, 2012b). However, they are more suited for GIS Applications, and their APIs are orders of magnitude more complex and generally not as fast as other alternatives. The best resource comparing WMTS and TMS is a presentation from 2010 (Masó et al., 2010) that was presented at the FOSS4G (Free and Open-Source Software for Geospatial) event hosted by OSGeo.

The standards commonly used on the web (based on support in available JavaScript libraries) are TMS (Tile Map Service), WMTS (Web Map Tile Service), XYZ, and most recently COG (Cloud Optimized GeoTIFF). XYZ does not refer to a real standard, but is a common name used by various ad hoc schemes (smithkm & Taras, 2021). Tile servers claiming to use XYZ usually seem to serve tiles according to the WMTS standard but do not support any of the other features, such as GetCapabilities and GetFeatureInfo requests. Some well-known implementations, referred to as XYZ, are Google Maps, OpenStreetMap, ESRI, and newer versions of Bing Maps.

QuadKeys or Bing Maps Tile System, as referred to on the official webpage (Schwartz, 2022), is the standard Bing Maps used in the past and, to some degree, still use. It seems to be less commonly supported by JavaScript front-end libraries.

Cloud Optimized GeoTIFFs is a newer approach to hosting web maps that leverages the internal file structure and HTTP GET range requests to ask for just the parts of the file they need. This saves a lot of storage space on the backend, while sacrificing a bit of speed at the

front end. Most solutions that leverage COGs are backend servers that dynamically create tiles as needed from the COGs. Some examples of these are TiTiler and Terracotta which can be read more about in 2.7.1.

2 Methods

This chapter takes up the scope of the thesis, the benchmarking tools and environment used for the benchmarks, the geospatial data used for benchmarking, the criteria and selection of the tile generation software, some notes on dynamic tiling alternatives, and a detailed explanation of the actual benchmarking process and file size measurements.

2.1 Scope

Comparing all relevant server software solutions for raster tile generation and raster tile serving is too large of a scope for a bachelor's thesis. For this reason, I have focused on static solutions comparing the performance of solutions that generate *Slippy map tilenames*, XYZ raster tiles, WTMS, or TMS tiles.

Completely neglecting the dynamic tile servers and dedicated tile server software out there would paint a very narrow and misleading view of the state of raster tile servers and the available software. Therefore, even though I do not directly compare the performance of these other solutions, they help place static tile generation in perspective.

2.2 Performance testing

The scripts and command line tools being benchmarked take different input parameters, such as the input file to use as the source material, and the output folder for the tiles, as well as many other options. They generate folders of tiles in the specified formats and standards. In the benchmarks in this thesis, we focus on the TMS standard and the JPEG, WebP, and PNG file formats with a tile size of 256 × 256 pixels. In the TMS standard, the tiles are structured in folders according to zoom level, an X coordinate, and a Y coordinate like "/zoom/x/y.png". So, what we are going to benchmark is the time it takes for the selected programs to turn the entire source map into tiles in JPEG, WebP, and PNG formats according to the TMS standard with zoom levels from 0 to 18 (0 being 156543 meters per pixel and 18 being 0.597 meters per pixel (OpenStreetMap Wiki Contributors, 2023)).

The benchmarking comparisons were performed using the command line benchmarking tool Hyperfine (Peter, 2023). The tool launches the programs we want to benchmark with the correct parameters, measures the time it takes to generate the tiles, and then saves the result in a markdown table, CSV file, or JSON file with detailed information for each test run. The tool did not exist as an apt package to be installed in Ubuntu, so it had to be installed via other means, the most convenient being the programming language, Rusts package manager Cargo.

2.3 Testing tool selection process

There are a few different alternatives for benchmarking using different approaches. Benchmarking the script by measuring how long it takes for the program to execute or by profiling, a more detailed test measuring how often and for how long various parts of the program are used (Python Software Foundation, 2023). The Python standard library contains a profiler, cProfile, which can be used for this purpose. Because ctb-tile was not a Python program like the others, it made more sense to benchmark instead of profiling the program. The command line benchmarking alternatives that I could find were Hyperfine and Bench (Gabriella43, 2018). Of these, Hyperfine was selected based on more GitHub stars as well as features. The preparation feature that makes it possible to run commands between timed runs to remove old tiles was particularly helpful.

2.4 Environment

The testing environment for each program was a docker container with hyperfine added based on the most official docker container that existed for the program. The dockerfiles used are published on GitHub *(https://github.com/slightly-blue/raster-tile-map-generation-benchmarking)*. The docker image used to benchmark ctb-tile was based on the most official docker image for ctb-tile *homme/cesium-terrain-builder:latest,* and the docker image for the other three programs, gdal2tiles, gdal2tiles_parallel, and gdal2tiles_mp, was based on the official GDAL docker image *osgeo/gdal:ubuntu-full-latest*. The images were run using docker engine. The hosting system was Ubuntu 20.04.5 LTS which ran on a standard.large flavoured instance on Pouta, a service provided by the CSC – IT Center for Science, Finland, for computational resources.

The standard.large instance used has the following specs

Flavour	Cores	Memory	Root Disk	Ephemeral disk	Total disk	Memory / Core
standard.large	4	7.8 GiB	80 GB	0	80 GB	1.9 GiB

Table 2. Instance specs from CSC documentation https://docs.csc.fi/cloud/pouta/vm-flavors-and-billing/#standard-flavors_2

Benchmarking inside docker removes some of the external factors that can influence a benchmark. This approach is not perfect, and a few online articles report slower code and distorted benchmarks (Turner-Trauring, 2021). This should not matter because the most meaningful measurement is how the scripts compare with each other on the same system. The containerisation of the environment has the added benefit of making these tests quickly reproducible on another machine with the help of the Dockerfiles.

2.5 Benchmark file

The file used to benchmark the different programs was mostly arbitrarily chosen from the National Land Survey of Finland's file service for open data (National Land Survey of Finland, 2023). The files available for download are in the format JPEG2000 in the coordinate reference system ETRS-TM35FIN. The size of the tile was 6 × 6 km and 12 000 × 12 000 px. With 3 bands. The tile chosen was L4321G, representing the southern area of the city of Porvoo, partly because it had a good mix of cityscape, sea, forest, and fields, where file compression has room to make a difference.

The JPEG2000 format is good because it is small, but it is not very friendly to the web. The file is also in the wrong CRS; therefore, additional conversions are required by the software generating the tiled web map.

The readme file in the repository for Cesium terrain builder recommends an input format that is tile-based (as opposed to scanline-based) with overviews and the same spatial reference system as the output (WGS 84) to speed up the processing. The coordinate reference system used on the web is WGS 84 / Pseudo-Mercator -- Spherical Mercator (EPSG:3857), not to be confused with WGS84 used by the global positioning system (EPSG:4326). For this reason, we will use a secondary file, the same 6 × 6 km area converted to Cloud Optimized GeoTIFF.

(with the target CRS baked in) to measure how well the programs handle the different input formats. This file was created with GDAL by first converting the CRS with *"gdalwarp -t_srs EPSG:3857 L4321G.jp2 out.tif"* and then converting that to COG with "*gdal_translate out.tif cog-source.tif -co TILED=YES -co COPY_SRC_OVERVIEWS=YES -co COMPRESS=DEFLATE"* using the official instructions for creating a COG (COG Contributors, 2022). After converting the CRS to *EPSG:3857* for the COG source file, the image size increased to 12 235 × 12 255 px.

2.6 Criteria for choosing software

The criteria for selecting the tile generation software to be benchmarked were that they had similar capabilities that could be compared to gdal2tiles. The capability to transform large rasters (in the formats JP2 or COG) to JPEG, PNG, or WEBP tiles in the TMS, WMTS, or XYZ standards. The secondary requirement was that the solution be free and open source. Although these are hard-set criteria for being able to time the performance of the scripts on the backend, comparing the resulting file size of these solutions does not have the same limitations, making it possible to compare the file size to COGs. While very different from the other static formats, consuming COGs directly on the front end and transforming it into a tiled web map is also possible.

Other solutions capable of generating tiled web maps do exist, but they are dedicated/dynamic tile servers that are not directly comparable to the scripts and command line utilities like gdal2tiles. Because they cannot be timed like scripts and the resulting dataset cannot be compared in terms of size.

2.7 Selecting the tile generation candidates

Many programs seem to have the capability to generate tiles in a way comparable to gdal2tiles, but many I looked at ended up being some sort of dynamic tile server [see section about dynamic tile servers, 2.7.1].

Finding these programs where a result of working with gdal2tiles and reading through different forums like *GIS Stack Overflow* and readme files on GitHub repositories as well as using

targeted Google searches with various gdal2tiles related keywords such as “generating tiles”, “XYZ tiles”, “TMS tiles” “Slippy map tilename generator”, “gdal2tiles alternative”

These are the primary ones I were able to find:

Cesium Terrain Builder (ctb-tile)

A C++ Library and associated command line tools to generate terrain tiles were built by (GeoData, 2018) and first released in 2014. Terrain tiles are used to represent digital elevation models. However, the program can also be used to work with other raster tiles. It is built on GDAL and works with all GDAL-accepted formats. For the benchmarks, the command line tool ctb-tile was used through the *homme/cesium-terrain-builder* docker image. The program is also mentioned in (Di Staso et al., 2016) who proposed a similar tool with the added capability to merge heterogenous-resolution DEM (digital elevation model) datasets.

Gdal2tiles

Gdal2tiles (GDAL/OGR contributors, 2022) create TMS or XYZ tiles and metadata in the KML format from any format accepted by GDAL and can bundle simple web viewers. The program seems to have been released in 2008, stemming from a Google summer of code project based on copyright information in the script.

Gdal2tiles_parallel

Released in 2014 as part of geopackage-python, a Python tool to create OGC GeoPackages by Reinventing Geospatial, Inc (Lander, 2014). This version of gdal2tiles added multiprocessing improvements to obtain better tile-generation performance. The latest release is from 2015 and the project is now archived.

Gdal2tiles_mp

A GitHub project inspired by gdal2tiles_parallel aiming to bring multiprocessing to gdal2tiles (Gaviria & Raigosa, 2019). There is no information about the first release date of the software, but the oldest commit in the repository was from 2018.

A few other software I looked at that I initially thought would be able to generate tiles are these. Gdal_translate which comes with GDAL and can convert between many file formats but does not seem to be capable of generating tiles in any of the standards I investigated. Another was

rio-tiler a Python library that essentially functioned as a wrapper for GDAL. The rest I looked at was either dedicated tile servers or dynamic tile servers.

2.7.1 On dynamic tiling alternatives and dedicated tile servers

While researching different tiling alternatives, quite a few ended up being unsuitable for direct comparison with gdal2tiles because of their different natures. This was due to them either being dedicated tile servers with no capability to generate tiles or dynamic tile servers generating tiles on the fly.

Initially, I thought about comparing the static pre-rendered solutions discussed so far with a modified variant of these scripts that, in theory, would dynamically generate tiles upon request, saving a lot of storage space. However, upon a superficial investigation, it did not make economic sense to approach such a solution. The computing cost of converting the tiles every time they are needed on a service such as EC2 would outweigh the cost of storing the tiles with a bit of increased storage on a bucket service such as S3. Furthermore, the front-end loading times would most likely suffer.

There are a few dynamic tiling solutions, but they do not work based on the GDAL-based programs that have been covered in this work thus far, as I am aware. They are however very different from one another, leveraging different techniques and serving slightly different purposes while all falling within the category of tile servers.

To better understand how they work, know that they use a combination of these techniques and features:

Core features

- Cloud Optimised Geo Tiffs and range requests
- Intelligent caching
- Rendering on the fly
- Pre-rendering

Other features

- Built-in webserver.
- API to specify rendering parameters (like tile size, format, and compression)
- Converting vector data to raster tiles
- Styling tiles (colour correction, sharpening, and contrast, in combination with the conversion from vector tiles to raster tiles defining the styles used)
- Sourcing from the GIS database

Mod_tile

Mod_tile is an Apache plugin using Mapnik as a rendering backend. Features efficient caching and on-the-fly rendering. Mapnik focuses on converting vector data to raster tiles while applying custom styling in the process. (Mod tile contributors, 2021)

TileSweep

Prerendering and serving tiles with a built-in web server. Mapnik backend. (Kallas, 2018)

Mb_tileserver

Server for map tiles in MBTiles format (Ward & Wickborn, 2022). The Mbtile format is a packaging format for raster tiled maps specified by Mapbox (Mapbox, 2018)

TiTiler

Based on FastAPI. Has an extensive API for requesting tiles in various formats. The source dataset can be COG (Cloud Optimized GeoTIFFs) or STAC (Spatio Temporal Asset Catalogs). Can serve tiles using the WMTS standard. (Sarago et al., 2023)

TileServer GL

A vector and raster tile server. Uses *MapLibre GL Native* to render raster tiles from vector data. (Maptiler, 2023)

Terracotta

A serverless Python deployed XYZ tile server, sourcing content from COGs (Terracotta contributors, 2021)

Cloud Optimized GeoTIFFs

Technically, a file format and not a tile server, but in practice, the frontend can use HTTP GET range requests to obtain only the parts of the file it needs. This is also mentioned among the tiled web map standards. However, it requires a separate web server. Most object storage options (Amazon, Google, Microsoft, OpenStack, etc) support get range requests) (COG Contributors, 2022)

To paint a better picture of the popularity, and as a probable extension quality, the number of GitHub stars is included in the following table.

Program	GitHub Stars
Mod_tile	255
TileSweep	18
Mb_tileserver	475
TiTiler	492
TileServer GL	1.7k
Terracotta	555

Table 3. GitHub stars for a few Dynamic and Dedicated Tile servers April 2023

2.8 Benchmark process

The following are the commands that were used to perform the benchmarks in the respective docker containers. The -d flag allows running the container in detached mode in the background, but immediately stops unless the commands are not running in the foreground, as documented in this article (Hirokuni, 2014). This is perfect for our use case because we want the instance to perform the benchmarks and then promptly shut down. The -v flag is used to mount the data folder from the Ubuntu instance to the same data folder in the Docker instance. This is to persist the results from Hyperfine that export the benchmark results in the form of markdown tables and JSON files. The benchmark was performed five times for each operation to obtain an average result to further remove external factors that could influence the benchmark, and the –prepare flag in hyperfine was leveraged to delete the generated tiles between each run.

To make the benchmarks as fair as possible, the tiles were all generated in the TMS standard, with zoom levels from 0 to 18, in the Mercator EPSG:3857 projection. All other options were kept as their internal default values.

For further information about the parameters, refer to the GitHub repositories for the respective programs being benchmarked as well as the GitHub repository for Hyperfine and the docker documentation.

The benchmarking was performed using the following commands.

CTB-TILE from JP2

```
sudo docker run -v /data:/data -d -e GDAL_SKIP='JP2ECW' ctb-tile-benchmark:latest  \
hyperfine \
--runs 5 \
--export-json /data/result/ctb-tile-5runs-jp2.json \
--export-markdown /data/result/ctb-tile-5runs-jp2.md \
--prepare 'rm -r /data/tiles/* || true' \
'ctb-tile --start-zoom 18 --end-zoom 0 --output-format PNG --profile mercator -o /data/tiles /data/L4321G.jp2' \
'ctb-tile --start-zoom 18 --end-zoom 0 --output-format JPEG --profile mercator -o /data/tiles /data/L4321G.jp2' \
'ctb-tile --start-zoom 18 --end-zoom 0 --output-format WEBP --profile mercator -o /data/tiles /data/L4321G.jp2'
```

ctb-tile crash after creating tiles for levels 18-12 without the *GDAL_SKIP='JP2ECW'* parameter. The available JP2 drivers in this docker image were the following, collected with "*gdalinfo --formats | grep 'JP2'*".

- JP2ECW -raster,vector- (rw+v): ERDAS JPEG2000 (SDK 3.x)
- JP2OpenJPEG -raster,vector- (rwv): JPEG-2000 driver based on OpenJPEG library
- JP2MrSID -raster- (rov): MrSID JPEG2000

CTB-TILE from COG

```
sudo docker run -v /data:/data -d ctb-tile-benchmark:latest  \
hyperfine --runs 5 \
```

*--export-json /data/result/ctb-tile-5runs-cog.json *
*--export-markdown /data/result/ctb-tile-5runs-cog.md *
--prepare 'rm -r /data/tiles/ || true' *
*'ctb-tile --start-zoom 18 --end-zoom 0 --output-format PNG --profile mercator -o /data/tiles /data/cog-source.tif' *
*'ctb-tile --start-zoom 18 --end-zoom 0 --output-format JPEG --profile mercator -o /data/tiles /data/cog-source.tif' *
'ctb-tile --start-zoom 18 --end-zoom 0 --output-format WEBP --profile mercator -o /data/tiles /data/cog-source.tif'

GDAL2TILES

*sudo docker run -v /data:/data -d gdal-parallel-benchmark:latest *
*hyperfine *
*--runs 5 *
*--export-json /data/result/gdal2tiles-cog-and-jp2.json *
*--export-markdown /data/result/gdal2tiles-cog-and-jp2.md *
*--export-csv /data/result/gdal2tiles-cog-and-jp2.csv *
--prepare 'rm -r /data/tiles/ || true' *
*'gdal2tiles.py -p mercator -s EPSG:3857 -z 0-18 -w none --tilesize=256 --tiledriver=PNG /data/cog-source.tif /data/tiles' *
*'gdal2tiles.py -p mercator -s EPSG:3857 -z 0-18 -w none --tilesize=256 --tiledriver=WEBP /data/cog-source.tif /data/tiles' *
*'gdal2tiles.py -p mercator -s EPSG:3857 -z 0-18 -w none --tilesize=256 --tiledriver=PNG /data/L4321G.jp2 /data/tiles' *
'gdal2tiles.py -p mercator -s EPSG:3857 -z 0-18 -w none --tilesize=256 --tiledriver=WEBP /data/L4321G.jp2 /data/tiles'

GDAL2TILES_PARALLEL

*sudo docker run -v /data:/data -d gdal-parallel-benchmark:latest *
*hyperfine *
*--runs 5 *
*--export-json /data/result/gdal2tiles_parallel_cog_and_jp2.json *
*--export-markdown /data/result/gdal2tiles_parallel_cog_and_jp2.md *
--prepare 'rm -r /data/tiles/ || true' *

*'python gdal2tiles_parallel.py -p mercator -e -z 0-18 -f JPEG /data/cog-source.tif /data/tiles' *
*'python gdal2tiles_parallel.py -p mercator -e -z 0-18 -f PNG /data/cog-source.tif /data/tiles' *
*'python gdal2tiles_parallel.py -p mercator -e -z 0-18 -f JPEG /data/L4321G.jp2 /data/tiles' *
'python gdal2tiles_parallel.py -p mercator -e -z 0-18 -f PNG /data/L4321G.jp2 /data/tiles'

GDAL2TILES_MP

*sudo docker run -v /data:/data -d gdal-parallel-benchmark:latest *
*hyperfine *
*--runs 5 *
*--export-json /data/result/gdal2tiles_mp_cog_and_jp2.json *
*--export-markdown /data/result/gdal2tiles_mp_cog_and_jp2.md *
--prepare 'rm -r /data/tiles/ || true' *
*'python gdal2tiles_mp.py -p mercator -z 0-18 /data/L4321G.jp2 /data/tiles' *
'python gdal2tiles_mp.py -p mercator -z 0-18 /data/cog-source.tif /data/tiles'

2.9 File size measurements

I chose to measure the file size when generating from both COG and JP2 due to them reporting different values while they should be very close to the same, more on that later.

The same pair of source tiles used to benchmark the processing speed was used to measure the resulting size of the created tiles. The area represented is 6 × 6 km and 12 000 × 12 000 px. With 3 bands. After converting the CRS to EPSG:3857 for the COG source file, the image size increased to 12 235 × 12 255 px. ctb-tile through the docker image on my Windows 10 workstation was used to generate XYZ tiles with the zoom levels 0-18. To read the filesize of the directory the command "*dir /s tiles*" was used and the command "*del /S *.xml*" to delete the additional XML files. Ctb-tile automatically creates *`{index}.jpg.tmp.aux.xml`* files along with the *`{index}.jpg`* files. The content of the XML file indicates a GDAL PAM Dataset. For this reason, I measured the total file size with and without auxiliary metadata.

After generating the files and measuring the file sizes, I did the same thing with COG as the source tile, and, to my surprise, it generated a different result. I investigated zoom layer 18 with the X folder 149777. The Unix diff command utility (through Cygwin) reported that all binary

files differed between the /18/149777/ folder generated from JP2 and the corresponding folder generated from the converted COG file.

Figure 2. The file /18/149777/186592.png. generated from COG (left), generated from JP2 (middle), and the difference between the images.

The difference is barely noticeable having the images right next to each other. To better show the difference, I have provided a third image created in gimp by taking the difference between the two images, imported as layers, and exporting the result; however, overlaying and toggling between the images show the difference much better. By toggling between them, one can also see the warping taking place, looking like the COG generation warped ca 70px strips at the same time; for example, the road on the left-hand side of the image is slightly disjointed and the car in the middle is sliced in two, whereas JP2 handles it on a per-pixel basis.

It seems to be an issue not related to ctb-tile or gdal_translate that was used to create the Cloud Optimized Geo TIFF but rather a problem with the Gdalwarp utility that changes the CRS. Running *gdalwarp -t_srs EPSG:3857 L4321G.jp2 out.tif* shows the same artefacts as the output. I controlled with the GDAL_SKIP=JP2ECW parameter to determine if it was the JP2 driver that was acting up again, but it returned the same result. This bug report seems to be about the same issue (https://github.com/OSGeo/gdal/issues/1620)

While regrettable, some pixels in slightly incorrect positions should not noticeably influence the benchmarks.

3 Literature Review

This is a short literature review of what previous work has been done on the speed of raster tile generation with programs such as gdal2tiles, file size comparisons of PNG, JPEG, and WebP, in addition to literature mentioning gdal2tiles and literature comparing static and dynamic tile servers.

Comparing the speed of tile generation

A few studies have compared raster and vector tiles (Agosto, 2013; Fábry & Feciskanin, 2019; Netek et al., 2020), but there is not much on tile generation. An article on the GitHub wiki for gdal2tiles_parallel (Cochran, 2015) measured the speed of generating tiles to and from different coordinate reference systems in the TMS standard. However, it does not mention what file format the generation is performed in or makes comparisons to other file formats.

In a message on a gdal-dev mailing list Cesium terrain builder is presented as a gdal2tiles alternative with notes on performance (Zwaagstra, 2014). The message mentions that ctb-tile is multithreaded which would improve performance over gdal2tiles by the number of cores the machine has, noting that disk i/o starts to become a limiting factor.

Comparing file size between formats

Regarding the performance of the file formats, one of the articles comparing raster and vector tiles (Netek et al., 2020) noted that the WebP format uses approximately three times less data than the PNG format. Scientific articles on the encoding and decoding aspect of the file formats like (Ginesu et al., 2012; Öztürk & Mesut, 2021) are easily found but those that consider the file size is not listed on google scholar or springer link. Some webpages, such as Google developers, have comparisons related to the WebP format. One such example is (Alakuijala & Vincent, 2017) which shows WebP as a more efficient replacement for PNG, with faster decoding speeds and 23% denser compression. With lossy compression (*WebP Compression Study | Google Developers*, n.d.) shows that WebP files are 25%-34% smaller compared to JPEG files of equivalent SSIM (structural similarity index measure) index.

Some backend load benchmarking exists for Cloud Optimized GeoTIFFs on https://trac.osgeo.org/ (*CloudOptimizedGeoTIFF – GDAL*, 2021); however, it mostly measures read times when ingesting the content through GDAL compared to normal GeoTiffs.

Literature mentioning gdal2tiles

Literature mentioning gdal2tiles ranges from the field of virtual microscopy (Alfaro et al., 2013) to Mars lander data display (Abarca et al., 2019) and even as a tool for converting old maps and documents to the web (Přidal & Žabička, 2008). However, nothing is related to the performance.

Static vs Dynamic tile servers

While there exists a multitude of blog posts comparing dynamic and static raster tile servers, there is little scientific literature directly dealing with it. (Netek et al., 2020) indirectly deals with it when in addition to comparing vector against raster tile servers it also compares tiles served via TileServer GL in raster format on request. This study shows the dynamically generated raster files in both PNG and WebP formats performing much worse than both static raster and vector tiles in downloading time and map loading time.

The documentation for TiTiler has a section on dynamic tiling (*Dynamic Tiling - TiTiler*, 2023) that mention that static tiling will always be faster than dynamic tiling, but that a cache layer can be set up in front of the dynamic tiler. The section also notes that having a dynamic tiler usually means that the user can set multiple options which often means that the same tiles will not be served twice.

One interesting approach to Cloud optimized GeoTIFFs is to ingest them directly on the frontend, and a journal article by (Pau & Fernández, 2022) mentioned that visualising COG files in web browsers using the geotiff.js library is surprisingly fast. The article also predicts that several new cloud-optimized formats will appear in the next few years.

A medium article showcasing the use of COGs directly on the frontend (Rennie, 2021) gave the impression that COGs would be the better choice for raster datasets with a resolution higher than TMS zoom level 25, based on the processing involved in generating all the tiles. There does not seem to be a maximum zoom level set in the TMS standard, but most front-end applications appear to stop at around the zoom level 20. Google Maps ending at 18 (Google Maps Platform, 2023), and the OpenStreetMap documentation not going beyond 20 (OpenStreetMap Wiki Contributors, 2023) An issue in the *Mapbox GL JS* repository (Mapbox GL, 2017) mention problems with zoom levels over 25, 20 being the default value in *Mapbox*

GL JS. No other resource I could find mentioned the resolution aspect of COGs as a perk over the TMS standard, but it does not appear to have been the object of study either.

4 Results

In this chapter, we review the results of the benchmarks, starting with the performance associated with generating tiles in different formats and continuing to the features available in each program. Finally, we look at the file sizes of the formats themselves.

4.1 All benchmarks

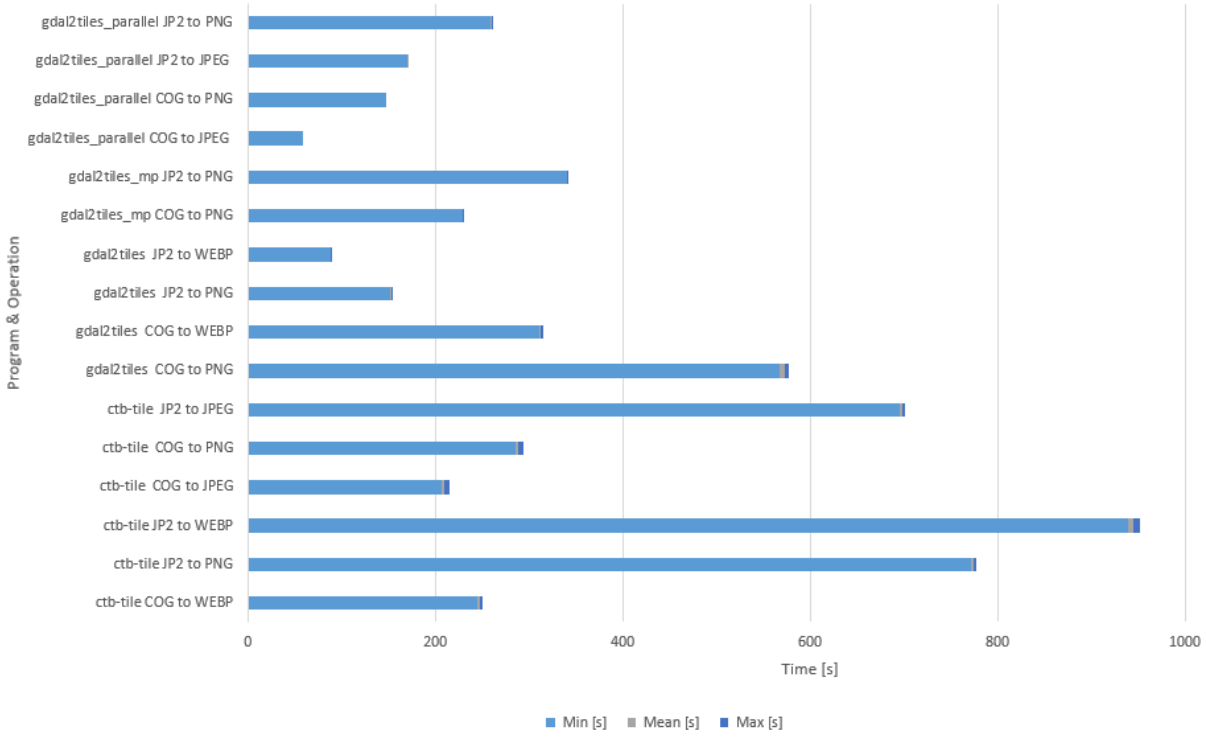

Figure 3. Benchmarks of all runs in order of program.

4.2 JP2 to WebP

JP2 to WEBP	Mean [s]	Standard deviation [s]
ctb-tile JP2 to WEBP	944.983	± 6.055
gdal2tiles JP2 to WEBP	88.837	± 0.183

Table 4. Benchmarks of tile-generating programs capable of turning JP2 source maps into WebP formatted map tiles.

Gdal2tiles_mp and gdal2tiles_parallel could not generate WebP tiles. Presumably, they were based on older versions of gdal2tiles that did not support it.

Furthermore, the documentation for gdal2tiles_mp does not mention any method of setting the output format. Looking through the code for clarification a comment seems to indicate the only available output is PNG tiles (gdal2tiles_mp.py, Line 1390)

4.3 JP2 to PNG

JP2 to PNG	Mean [s]	Standard deviation [s]
ctb-tile JP2 to PNG	775.139	± 2.018
gdal2tiles_mp JP2 to PNG	340.669	± 0.983
gdal2tiles_parallel JP2 to PNG	260.293	± 1.095
gdal2tiles JP2 to PNG	152.518	± 1.044

Table 5. Benchmarks of tile-generating programs capable of turning JP2 source maps into PNG formatted map tiles.

Tile generation from the JP2 format to PNG was the most performant on the standard gdal2tiles script, while being the slowest on ctb-tile.

4.4 JP2 to JPEG

JP2 to JPEG	Mean [s]	Standard deviation [s]
ctb-tile JP2 to JPEG	697.819	± 2.037
gdal2tiles_parallel JP2 to JPEG	171.121	± 0.699

Table 6. Benchmarks of tile-generating programs capable of turning JP2 source maps into JPEG-formatted map tiles

Gdal2tiles only support PNG tiles and WebP tiles in *GDAL 3.7.0* version released 2023/02/22.

4.5 COG to WebP

COG to WEBP	Mean [s]	Standard deviation [s]
gdal2tiles COG to WEBP	312.205	± 1.605
ctb-tile COG to WEBP	247.566	± 2.175

Table 7. Benchmarks of tile-generating programs capable of turning COG source maps into WebP formatted map tiles.

4.6 COG to PNG

COG to PNG	Mean [s]	Standard deviation [s]
gdal2tiles COG to PNG	573.103	± 3.460
ctb-tile COG to PNG	288.706	± 3.526
gdal2tiles_mp COG to PNG	229.855	± 0.897
gdal2tiles_parallel COG to PNG	147.538	± 0.071

Table 8. Benchmarks of tile-generating programs capable of turning COG source maps into PNG formatted map tiles.

4.7 COG to JPEG

COG to JPEG	Mean [s]	Standard deviation [s]
ctb-tile COG to JPEG	209.358	± 3.879
gdal2tiles_parallel COG to JPEG	58.605	± 0.188

Table 9. Benchmarks of tile-generating programs capable of turning COG source maps into JPEG-formatted map tiles

4.8 Source and output file format handling

The programs handle the JP2 and COG input file formats with very different performance. In the following table, it can be seen how gdal2tiles struggle with the COG source format, whereas ctb-tile performs much better with that format compared to the JP2 file format. All software except gdal2tiles handled COG much better than the JP2 source format. Looking at the performance associated with generating the output tile formats, it can be seen that JPEG is more expensive to generate than PNG. The performance related to creating WEBP tiles depends more on the software, gdal2tiles handling that better than PNG, whereas the performance of ctb-tile varies depending on the input format.

	COG to JPEG	COG to PNG	COG to WEBP	JP2 to JPEG	JP2 to PNG	JP2 to WEBP
ctb-tile	209.358	288.706	247.566	697.819	775.139	944.983
gdal2tiles	N/A	573.102	312.204	N/A	152.518	88.837
gdal2tiles_parallel	58.605	147.538	N/A	171.121	260.293	N/A
gdal2tiles_mp	N/A	229.855	N/A	N/A	340.669	N/A

Table 10. Overview matrix of all benchmarked software over operation. Values given as a mean [s] of all runs if the operation was possible.

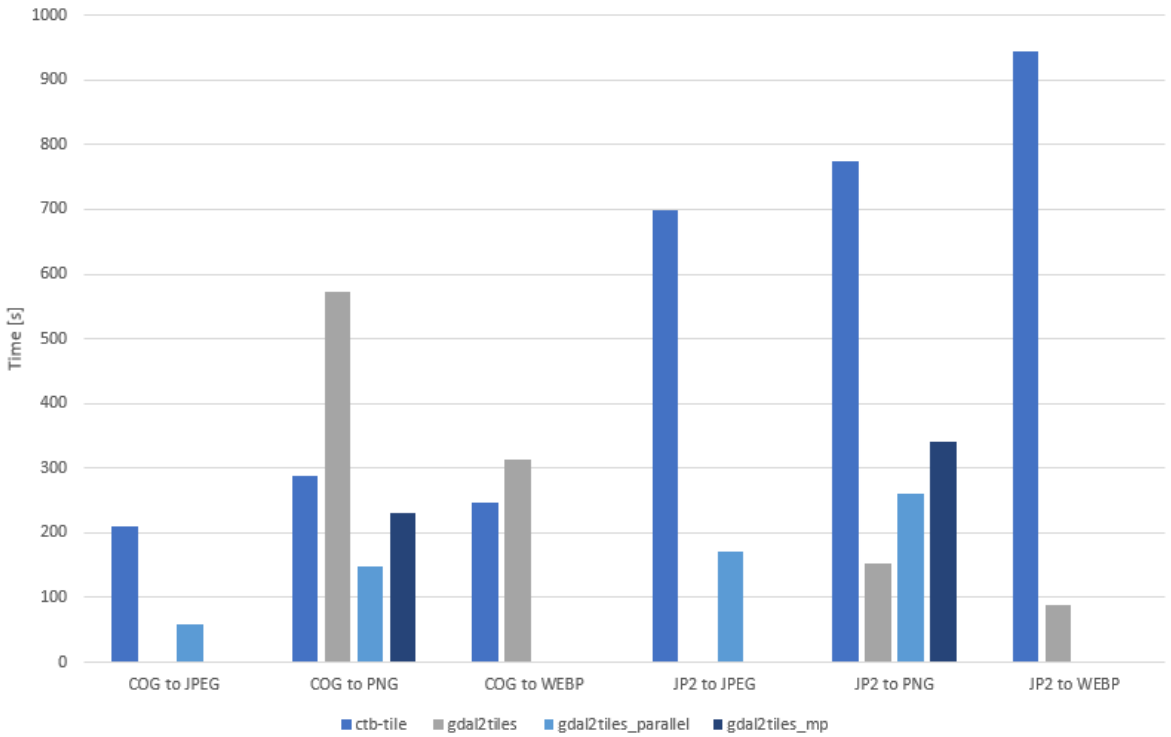

Figure 4. Tile generation software benchmark in mean[s] by operation.

4.9 Feature Map

To compare the features of the programs, their command line options were taken from their documentation and compiled in Tables 11 and 12. gdal2tiles_mp did not have any documentation, so I have only filled in the values for the areas I have tested and left the rest as N/A.

	Profile			**Output formats**		
	Mercator	**Geodetic**	**Raster**	**PNG**	**JPEG**	**WEBP**
gdal2tiles parallel	yes	yes	no	yes	yes	no
gdal2tiles	yes	yes	yes	yes	no	yes
gdal2tiles_mp	yes	n/a	n/a	yes	no	no
ctb-tile	yes	yes	no	yes	yes	yes

Table 11. Overview matrix of the profile and output formats available in the programs

	XYZ	Exclude	Processes /Threads	Tile Size	Resume	No data	CRS Setting	Resampling
gdal2tiles parallel	no	no	No, built-in	no	yes	yes	yes	no
gdal2tiles	yes	yes	yes	yes	yes	yes	yes	yes
gdal2tiles_mp	n/a	n/a	n/a, built-in	n/a	n/a	n/a	n/a	n/a
ctb-tile	no	no	yes	yes	yes	no	no	yes

Table 12. Overview matrix of various features available in the programs

An explanation for these features is in order. XYZ represents the ability to generate tiles in the XYZ unofficial standard with a flipped Y coordinate. The exclusion feature excludes transparent tiles from the output folder which, depending on the application, can save considerable space. The processes or threads feature is an option that specifies how many parallel processes or threads is allowed to be used to speed up the computation. Gdal2tiles additionally have a (Message Passing Interface) feature that enables MPI parallelism, which could be used to scale the performance further. The tile size option allows custom tile sizes to be specified, and the resume option makes it possible to pick up a half-completed run. The nodata option allows specifying what values should be considered “nodata” This is particularly useful if you want to exclude black tiles and not just transparent tiles. The CRS setting makes it possible to define the coordinate reference system used by the source input data. The resampling option tells the program the resampling algorithm to use, presumably for generating tiles beyond the most detailed zoom level in the source dataset or in the process of converting the CRS. The documentation is unclear.

Looking at these tables, gdal2tiles seem overwhelmingly superior in features to the other programs, while Ctb-tile can generate tiles in the largest number of formats. It is important to note that gdal2tiles have multiprocessing, which is the primary purpose of gdal2tiles_mp and gdal2tiles_parallel. This probably means that multiprocessing was added to gdal2tiles after gdal2tiles_mp and parallel were released. However, considering the rapid development of some open-source projects, these tables may also not be accurate or relevant for a long time.

4.10 File Format Comparison

Format	File(s)	Bytes
COG	1	421389190
JP2	1	105316770

Table 13. The file size of the source rasters provided for reference.

The file folder sizes were measured with and without auxiliary metadata XML files. See chapter 2.9 on why the measurement was done for both the tiles generated from COG and JP2 and why they differ.

COG Source	Including xml		Excluding xml	
Format	**File(s)**	**Bytes**	**Files(s)**	**Bytes**
JPEG	18148	91608216	9074	78015364
PNG	18148	800022389	9074	791057277
WEBP	18148	55256064	9074	47107612

Table 14. File size comparison of total generated tiles in the file formats jpeg, png, and webp, generated from the COG source file.

JP2 Source	Including xml		Excluding xml	
Format	**File(s)**	**Bytes**	**Files(s)**	**Bytes**
JPEG	18148	91669923	9074	78077071
PNG	18148	809277596	9074	800312484
WEBP	18148	55511196	9074	47362744

Table 15. File size comparison of total generated tiles in the file formats jpeg, png, and webp, generated from the JP2 source file

I will use the values generated from the COG to discuss the performance of the respective file formats. So, on average the size of each tile was 87.18kb for PNG, 8.60kb for JPEG, and 5.19kb for WebP tiles, respectively. These measurements show that PNG tiles take up 10.1 times more space than JPEG tiles and 17.8 times more space than WebP tiles. JPEG tiles, on the other hand take up 1.66 times more space than WebP tiles. JPEG and WebP use the default compression levels obtained from the internal settings of GDAL which should be 75 for both. A study mentioned earlier (*WebP Compression Study | Google Developers*, n.d.) indicates different SSIM (structural similarity index measure) for the same compression value between the formats, but does not mention how large the difference is. Even so, it still shows that WebP is more performant, even when taking this into account.

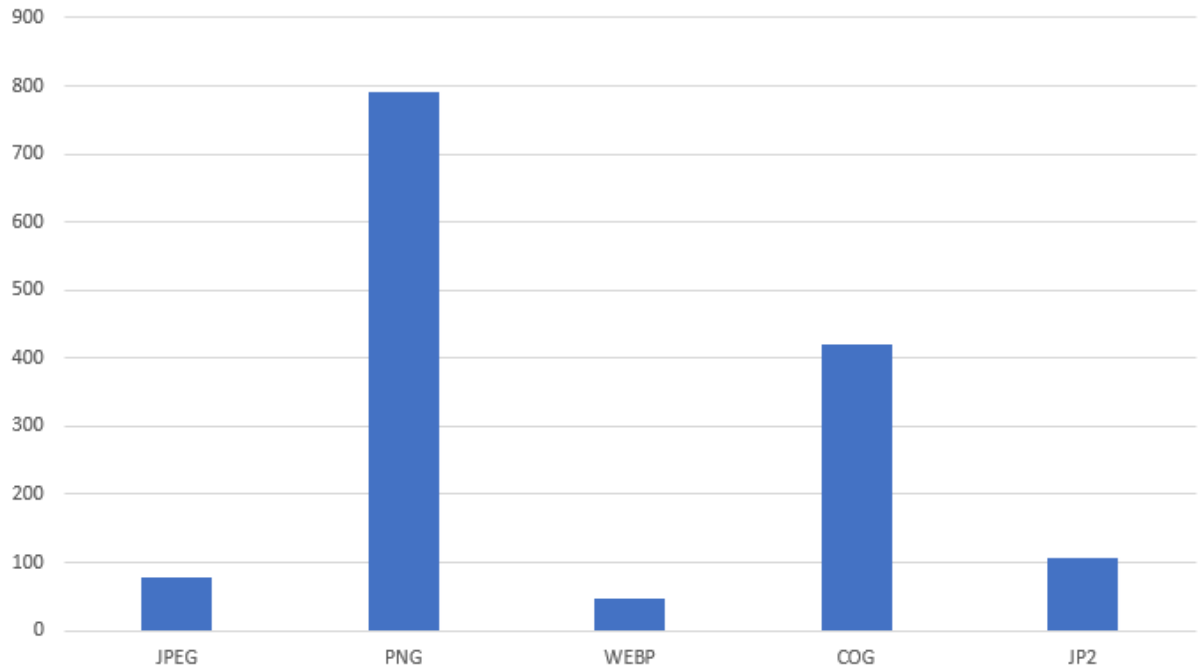

Figure 5. Bar chart of the file size for the respective formats generated from COG in megabytes, excluding xml files. The size of the source files COG and JP2 are provided for reference.

JPEG and PNG files are supported in most if not all browsers and WebP is supported in all web browsers listed at *caniuse.com/webp* except for internet explorer that is no longer supported. It is still at the end up to the individual front-end JavaScript libraries what formats are supported. In particular, the files contain elevation data that must be decoded and transformed into a 3D model.

An interesting observation here is that COGs use less space than PNG while also being a lossless format, in addition to having overviews and internal tiling. This could make COG seem more attractive if you want to ingest it directly on the front end.

5 Discussion and conclusions

This part is a summary of the work providing answers to the research questions. Acknowledging the limitations of this approach of benchmarking and comparing these software, as well as discussing interesting avenues for further research.

The capabilities and limitations of the available open-source raster tile map generation software alternatives discussed.

gdal2tiles have many more features, whereas ctb-tiles can generate tiles in the largest number of formats. Gdal2tiles not supporting JPEG as an output format in the new versions is surprising, but considering that WebP files take up less space, this is inconsequential. A comment on Line 1391-1392 in gdal2tiles.py *"TODO: fill the null value in case a tile without alpha is produced (now only png tiles are supported)"* indicates that the missing support for jpeg might be temporary. Gdal2tiles and ctb-tiles also have the largest amount of documentation, and based on GitHub stars, they also have the largest user base. This alone make them the most logical choice for developing a mapping application. Gdal2tiles have the optional feature of automatically creating a frontend web map that can be hosted along with the tiles. It also featured options for MPI parallelism that could increase the processing speed quite a bit. Features none of the other solutions had. Ctb-tile, on the other hand, was designed for terrain data and can create terrain tiles in the heightmap-1.0 terrain format.

The performance of the tile generation programs.

Retrospectively I think gdal2tiles_mp and gdal2tiles_parallel should have been skipped from this study based on the lack of updates over the last few years to their respective GitHub repositories and the general lack of documentation. These factors make them unsuitable for use in a real environment where documentation and maintenance of the program are relevant for futureproofing. This is despite having the best performance on COG files and okay performance on JP2 files. It might seem drastic to suggest that the programs should not be used when they had the best results, but it is important to note that the benchmarks were done without tweaking any extra parameters, doing any optimisations, or tweaking GDALs internal settings. This leaves room for further optimisation which could improve the performance of gdal2tiles and ctb-tiles. Because these are better documented and in wider usage, they serve as a better ground for exploring optimizations. A deeper dive in just optimising gdal2tiles could prove an interesting topic for further research.

Another interesting observation from the benchmarks is that gdal2tiles handled JP2 better than COG, whereas the rest of the programs had the opposite behaviour. I could not find a good explanation as to why this was.

How does a static alternative compare with a dynamically tiled solution?

One of the research questions was how static tile servers compared to dynamic ones. This was dealt with through the file size comparisons between COGs and the other static formats, as well as the literature review that dealt with the more traditional dynamic tile servers. The results from the literature review were that a dynamic tile server will always be slower than a static tile server, but it was a bit inconclusive when it came to serving COGs directly on the front end. The results from the file size study in this thesis show that COGs can save space compared to PNG tiles, but not when compared to WebP and JPEG tiles, at least in their lossy variants.

How size effective are different file formats?

While the file size tests showed how much space can be saved with compression (PNG tiles being 17.8 times larger on average compared to WebP), it was not exactly a ground-breaking revelation to anyone. An interesting aspect was that Cloud Optimized GeoTIFFs also took up less space than the PNG tiles, which begs the question of how much it would differ from WebP in its lossless configuration, which was not explored in this study.

Avenues for further research

This study had too large a scope to work with all the aspects and considerations around file formats in web map service applications but could serve as a foundation for further research.

Aspects that would improve this study would be to consider different tile sizes, and different compression levels while considering the SSIM index and the PSNR (Peak signal to noise ratio) to make the study comparable with other studies on file formats, such as the WebP format. It would also be helpful to differentiate between lossless and lossy formats more clearly.

Another aspect that was not covered here is the loading times associated with decoding the different formats on the front end, which have varying impacts depending on the performance of the device separate from the speed of the Internet connection. This was covered in (Öztürk & Mesut, 2021) but not in the context of web map servers.

The inclusion of more formats would also be an improvement. One more common format that was not included in this study was TIF files as tiles, more used for elevation data than satellite imagery, but nevertheless used. Investigating terrain tile formats further and the unique aspects of working with elevation data in web map services would also be interesting. Compression could be a problem if applied on formats such as terrain tiles, for example, Terrarium tiles, where raw elevation data are encoded in meters and split into red, green, and blue channels. The problem here is that one unit of change in the red value corresponds to 256 meters in elevation, while a unit of change in the blue channel only corresponds to 1 / 256 meters or ca 0.4cm. Compression algorithms, as far as I know, are not tuned for these edge cases which could potentially lead to some interesting bugs.

Ending notes

The greatest limitation of these benchmarks was how different the programs were which made it difficult to perform fair comparisons. This work, in the end, can be seen as a foundation for further benchmarking of raster tile applications. The benchmarking pipeline is probably the single most useful thing this thesis has provided, along with the collection of information that could help a developer obtain a rudimentary understanding of the state of raster tile servers.

References

Abarca, H., Deen, R., Hollins, G., Zamani, P., Maki, J., Tinio, A., Pariser, O., Ayoub, F., Toole, N., Algermissen, S., Soliman, T., Lu, Y., Golombek, M., Calef, F., Grimes, K., De Cesare, C., & Sorice, C. (2019). Image and Data Processing for InSight Lander Operations and Science. *Space Science Reviews*, *215*(2), 1–53. https://doi.org/10.1007/S11214-019-0587-9/FIGURES/39

About - Open Geospatial Consortium. (2023). https://www.ogc.org/about-ogc/

About OSGeo - OSGeo. (2023). https://www.osgeo.org/about/

Agosto, E. (2013). Vector-raster server-side analysis: A PostGIS benchmark. *Applied Geomatics*, *5*(2), 177–184. https://doi.org/10.1007/s12518-013-0104-x

Alakuijala, J., & Vincent, R. (2017). *Lossless and Transparency Encoding in WebP | Google Developers*. Google, Inc. https://developers.google.com/speed/webp/docs/webp_lossless_alpha_study#results

Alfaro, L., Roca, M. J., & Catala, P. (2013). Virtual microscopy with Google-Earth: a step in the way for compatibility. *Diagnostic Pathology 2013 8:1*, *8*(1), 1–4. https://doi.org/10.1186/1746-1596-8-S1-S11

CloudOptimizedGeoTIFF – GDAL. (2021). https://trac.osgeo.org/gdal/wiki/CloudOptimizedGeoTIFF#Performancetesting

Cochran, J. (2015). *Profiling Results of Tiling and Packaging Features in GeoPackage Python · GitHubRGI/geopackage-python Wiki*. https://github.com/GitHubRGI/geopackage-python/wiki/Profiling-Results-of-Tiling-and-Packaging-Features-in-GeoPackage-Python

COG Contributors. (2022). *Cloud Optimized GeoTIFF in depth*. https://www.cogeo.org/in-depth.html

Di Staso, U., Soave, M., Giori, A., Prandi, F., & De Amicis, R. (2016). Heterogeneous-Resolution and multi-source terrain builder for CesiumJS WebGL virtual globe. *International Journal of Civil and Architectural Engineering*, *10*(1), 129–135.

Dynamic Tiling - TiTiler. (2023). https://developmentseed.org/titiler/dynamic_tiling/

Encyclopedia of GIS. (2017). *Encyclopedia of GIS*. https://doi.org/10.1007/978-3-319-17885-1

Fábry, J., & Feciskanin, R. (2019). Publishing geodata in the form of vector tiles and comparing performance with raster tiles. In *Geograficky Casopis* (Vol. 71, Number 3, pp. 283–293). Institute of Geography of the Slovak Academy of Science. https://doi.org/10.31577/geogrcas.2019.71.3.15

Forrest, M. (2021, June 12). *A Brief History of Web Maps - Matt Forrest - Modern GIS and Geospatial Ideas and Guides*. https://forrest.nyc/a-brief-history-of-web-maps/

Gabriella43. (2018). *bench*. GitHub. https://github.com/Gabriella439/bench

Gaviria, S., & Raigosa, D. (2019). *parallelo-ai/gdal2tiles_mp: Adding multiprocessing to gdal2tiles*. https://github.com/parallelo-ai/gdal2tiles_mp

GDAL/OGR contributors. (2022). *GDAL/OGR Geospatial Data Abstraction software Library*. https://doi.org/10.5281/zenodo.5884351

GDAL/OGR contributors. (2023). *OSGeo/gdal: GDAL is an open source MIT licensed translator library for raster and vector geospatial data formats.* https://github.com/OSGeo/gdal

GeoData. (2018). geo-data/cesium-terrain-builder: A C++ library and associated command line tools designed to create terrain tiles for use in the Cesium JavaScript library. In *GeoData* (0.4.1). GitHub. https://github.com/geo-data/cesium-terrain-builder

Ginesu, G., Pintus, M., & Giusto, D. D. (2012). Objective assessment of the WebP image coding algorithm. *Signal Processing: Image Communication*, *27*(8), 867–874. https://doi.org/10.1016/J.IMAGE.2012.01.011

Google Maps Platform. (2023). *Maximum Zoom Imagery Service | Maps JavaScript API | Google Developers*. https://developers.google.com/maps/documentation/javascript/maxzoom

Hirokuni, K. (2014). *Gotchas in Writing Dockerfile | Program Is Made At Night*. https://kimh.github.io/blog/en/docker/gotchas-in-writing-dockerfile-en/#hack_to_run_container_in_the_background

History of GIS. (2023). https://www.esri.com/en-us/what-is-gis/history-of-gis

Kallas, S. (2018). *seemk/TileSweep: OpenStreetMap tile server* (0.1.0). GitHub. https://github.com/seemk/TileSweep

Lander, S. D. (2014). *Usage Instructions for gdal2tiles_parallel.py · GitHubRGI/geopackage-python Wiki · GitHub*. Reinventing Geospatial®, Inc. (RGi®). https://github.com/GitHubRGI/geopackage-python/wiki/Usage-Instructions-for-gdal2tiles_parallel.py

Mapbox. (2018). *mapbox/mbtiles-spec: specification documents for the MBTiles tileset format*. https://github.com/mapbox/mbtiles-spec

Mapbox GL. (2017). *Rasterlayer MaxZoom above 22 not loading · Issue #4067 · mapbox/mapbox-gl-js*. https://github.com/mapbox/mapbox-gl-js/issues/4067

Maptiler. (2023). *maptiler/tileserver-gl: Vector and raster maps with GL styles. Server side rendering by MapLibre GL Native. Map tile server for MapLibre GL JS, Android, iOS, Leaflet, OpenLayers, GIS via WMTS, etc.* GitHub. https://github.com/maptiler/tileserver-gl

Masó, J. (2022). Geospatial Web Services. In W. Kresse & D. Danko (Eds.), *Springer Handbook of Geographic Information* (pp. 493–530). Springer International Publishing. https://doi.org/10.1007/978-3-030-53125-6_16

Masó, J., Pons, X., & Singh, R. (2010). *OGC WMTS and OSGeo TMS standards: motivations, history and differences*.

Mod tile contributors. (2021). *openstreetmap/mod_tile: Renders map tiles with mapnik and serves them using apache* (0.6.1). https://github.com/openstreetmap/mod_tile

National Land Survey of Finland. (2023). *Open data file download service | National Land Survey of Finland*. https://www.maanmittauslaitos.fi/en/e-services/open-data-file-download-service

Netek, R., Masopust, J., Pavlicek, F., & Pechanec, V. (2020). Performance testing on vector vs. Raster map tiles - Comparative study on load metrics. *ISPRS International Journal of Geo-Information*, *9*(2). https://doi.org/10.3390/ijgi9020101

Open Geospatial Consortium. (2023a). *OpenGIS Web Map Tile Service Implementation Standard - Open Geospatial Consortium*. https://www.ogc.org/standard/wmts/

Open Geospatial Consortium. (2023b). *Web Map Service - Open Geospatial Consortium*. https://www.ogc.org/standard/wms/

OpenStreetMap Wiki Contributors. (2022). *Slippy map - OpenStreetMap Wiki*. https://wiki.openstreetmap.org/wiki/Slippy_map

OpenStreetMap Wiki Contributors. (2023). *Zoom levels - OpenStreetMap Wiki*. https://wiki.openstreetmap.org/wiki/Zoom_levels

OSGeo. (2012a). *Tile Map Service Specification - OSGeo*. https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification

OSGeo. (2012b). *WMS Tile Caching - OSGeo*. https://wiki.osgeo.org/wiki/WMS_Tile_Caching

Osgeo. (2021). *Open Source GIS History - OSGeo*. https://wiki.osgeo.org/wiki/Open_Source_GIS_History

Öztürk, E., & Mesut, A. (2021). Performance evaluation of JPEG standards, WEbP and PNG in terms of compression ratio and time for lossless encoding. *Proceedings - 6th International Conference on Computer Science and Engineering, UBMK 2021*, 15–20. https://doi.org/10.1109/UBMK52708.2021.9558922

Pau, J. M., & Fernández, X. P. (2022). After years convincing people about geospatial services, we are going back to files, but this time we call them" Cloud optimized"/Después de años convenciendo a la gente sobre los servicios geoespaciales, volvemos a los archivos, pero esta vez los llamamos" Optimizados para la nube". *GeoFocus. International Review of Geographical Information Science and Technology*, *30*, 1–4.

Peter, D. (2023). *hyperfine*. https://github.com/sharkdp/hyperfine

Přidal, M. P., & Žabička, P. (2008). Tiles as an approach to on-line publishing of scanned old maps, vedute and other historical documents. *E-Perimetron*, *3*(1), 10–21.

Python Software Foundation. (2023). *The Python Profilers — Python 3.11.3 documentation*. https://docs.python.org/3/library/profile.html

Rennie, S. (2021). *COGs in production. Leveraging Cloud Optimised GeoTIFFs… | by Sean Rennie | Medium*. https://sean-rennie.medium.com/cogs-in-production-e9a42c7f54e4

Sarago, V., Barron, K., & Albrecht, J. (2023). *TiTiler* (0.11.5). GitHub. https://github.com/developmentseed/titiler

Schwartz, J. (2022). *Bing Maps Tile System - Bing Maps | Microsoft Learn*. https://learn.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system

smithkm, & Taras. (2021). *What are the Differences Between TMS, XYZ & WMTS? - Geographic Information Systems Stack Exchange*. https://gis.stackexchange.com/a/242978

Sterling, Q., & John A. (n.d.). *The history and importance of web mapping | GEOG 585: Web Mapping*. Dutton E-Education Institute, College of Earth and Mineral Sciences, The Pennsylvania State University. Retrieved 4 May 2023, from https://www.e-education.psu.edu/geog585/node/643

Terracotta contributors. (2021). *Welcome to Terracotta — Terracotta 0.7.6.dev127+g17d4789 documentation*. https://terracotta-python.readthedocs.io/en/latest/

Turner-Trauring, I. (2021). *Docker can slow down your code and distort your benchmarks*. https://pythonspeed.com/articles/docker-performance-overhead/

Ward, B., & Wickborn, F. (2022). consbio/mbtileserver: Basic Go server for mbtiles. In *Conservation Biology Institute* (0.9.0). Conservation Biology Institute. https://github.com/consbio/mbtileserver

Wasser, L. (2019). *Coordinate Reference System and Spatial Projection | Earth Data Science - Earth Lab*. https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/intro-to-coordinate-reference-systems/

WebP Compression Study | Google Developers. (n.d.). Retrieved 23 April 2023, from https://developers.google.com/speed/webp/docs/webp_study

Zwaagstra, H. (2014). *[gdal-dev] gdal2tiles.py alternative*. https://lists.osgeo.org/pipermail/gdal-dev/2014-July/039550.html

Appendices

Appendix 1: gdal-parallel-benchmark/Dockerfile

```
FROM osgeo/gdal:ubuntu-full-latest

RUN apt-get -qq update
RUN apt-get install -y -q \
    build-essential \
    curl

# Installing rust (required for hyperfine on ubuntu)
RUN curl https://sh.rustup.rs -sSf | sh -s -- -y

# Add .cargo/bin to PATH
ENV PATH="/root/.cargo/bin:${PATH}"

# installing hyperfine
RUN cargo install hyperfine

# gdal2tiles_parallel.py
RUN curl https://raw.githubusercontent.com/GitHubRGI/geopackage-
python/master/Tiling/gdal2tiles_parallel.py --output gdal2tiles_parallel.py

# gdal2tiles_mp.py
RUN curl https://raw.githubusercontent.com/parallelo-
ai/gdal2tiles_mp/master/gdal2tiles_mp.py --output gdal2tiles_mp.py

# command to create a docker image based on this file
# docker build -t gdal-parallel-test:latest .

# This image is used for benchmarking gdal2tiles, gdal2tiles_parallel and gdal2tiles_mp,
```

Appendix 2: ctb-tile-benchmark/Dockerfile

```
FROM homme/cesium-terrain-builder:latest

# Installing hyperfine

# ubuntu apt and apt-get does not have hyperfine so we will need
# to install it trough cargo
# Inspired by: https://stackoverflow.com/questions/49676490/when-installing-rust-toolchain-in-docker-bash-source-command-doesnt-work

RUN apt-get -qq update
RUN apt-get install -y -q \
    build-essential \
    curl

# Installing rust (required for hyperfine on ubuntu)
RUN curl https://sh.rustup.rs -sSf | sh -s -- -y

# Add .cargo/bin to PATH
ENV PATH="/root/.cargo/bin:${PATH}"

# installing hyperfine
RUN cargo install hyperfine

# command for creating a Dcker image based on this file.
# docker build -t ctb-tile-test:latest .

# This image is used for benchmarking ctb-tile
```

Appendix 3: Längre sammandrag på svenska

Det här är ett längre sammandrag på svenska som krävs av Arcadas exmensstadga (som utgår ifrån yrkeshögskolelagen (932/2014) och förordningen om yrkeshögskolor (1129/2014)) eftersom lärdomsprovet är skrivet på ett annat språk än examens språket.

Denna studie jämför förmågorna och prestandan på verktyg för raster tile kart-generering. En annan faktor som undersöks är storleken av olika filformat som används på statiska raster tile-servrar. Arbetet jämför också dessa statiska alternativ med dynamiska lösningar som skapar filerna först då de behövs.

Motivation

Motivationen bakom det här arbetet var bristen på dokumentation och litteratur som jämför frontend web map bibliotek, tile servrar, dynamiska tile servrar och tiled-web-map standarder och hur bra dessa fungerar med varandra. Det var också svårt att hitta information om vilka raster-filformat som lönar sig att använda inom kontexten av tiled-web-map tjänster. De här faktorerna gör skapandet av en web map service till ett problem som är onödigt svårt att navigera.

Begränsningar

Arbetet har begränsats till jämförelse av funktioner och prestanda mellan statiska lösningar för raster tile generering med öppen källkod som klarar av att generera tiles i TMS (Tile map service) standarden. Men att helt försumma dynamiska lösningar skulle måla en snäv och missvisande bild av tillståndet vad gäller raster tile servrar. Därför tas de upp i samband med valet av de statiska alternativen även om jag inte jämför prestandan på dessa dynamiska lösningar direkt.

Metodik

Med hjälp av *hyperfine* benchmarking-verktyget testades programmen gdal2tiles, ctb-tile, gdal2tiles_parallel och gdal2tiles_mp i Docker containrar med de mest officiella Docker avbildningarna för respektive program. Benchmarkingen gjordes via docker engine på operativsystemet *Ubuntu 20.04.5 LTS* som i sin tur kördes på en *standard.large* instans på *CSCs (IT Center for Science, Finland, for computational resources)* tjänst *Pouta*. Att göra benchmarkingen via docker tar bort en del externa faktorer som kan påverka resultatet. Att

miljön är containeriserad gör att dessa test enkelt kan reproduceras på en annan maskin med hjälp av dockerfilerna.

Filen som används som datakälla för alla benchmarks är tagen från Finska Lantmäteriverkets fil-tjänst för öppna data. Ortofoto-filer från tjänsten är i formatet *JPEG2000* i koordinat referenssystemet *ETRS-TM35FIN*. Dessa kartblad representerar en *6x6 km* region av storleken *12000x12000 px*. Filen som används i testen här var *L4321G* som representerar den södra delen av staden Borgå i Finland, vald för att den hade en bra blandning av stadsmiljö, hav, skog och åkrar. Variationen tjänar som en utmaning för kompressionsalgoritmerna när storleken på de olika filformaten testas.

För att göra testet mer realistiskt gjordes benchmarking också med en variant av samma fil konverterad till *Cloud Optimized GeoTIFF (COG)*. COG-filen är färdigt konverterad till samma koordinatreferenssystem som slutresultatet och har också inbyggda överblickar (overviews) som i teorin borde försnabba genereringsprocessen.

För att göra benchmarkingen så rättvis som möjligt genererades alla tiles i TMS standarden med zoomnivåer från 0 till 18 i *Mercator EPSG:3857* projektionen. Alla andra inställningar hölls som programmens interna standardinställningar.

Tidigare verk

Arbetet innehåller en kort litteraturöversikt som behandlar tidigare arbete vad gäller hastigheten på raster tile generation, filformatjämförelser mellan PNG, JPEG och WebP, samt litteratur som jämför statiska och dynamiska tile servrar. Litteraturen och dokumentationen som fanns behandlade inte direkt någon av dessa problem, och kom oftast inte från akademiska källor. Några källor behandlade jämförelser mellan raster och vektor tiles, och andra bara enskilda filformat. Ingen källa var direkt jämförbar med det här arbetet.

Resultat

Resultatet av benchmarkingen visade väldigt varierande exekveringstider. Skillnaden mellan exekveringstiderna för indata formaten JP2 och COG är ett bra exempel. Gdal2tiles hanterade COG formatet väldigt långsamt och JP2 formatet relativt snabbt medan ett av de andra programmen ctb-tile, gdal2tiles_mp och gdal2tiles_parallel hade motsatt prestanda.

Vad gäller prestandan för utdata-filformaten tog JPEG konstant längre att generera än PNG medan genereringen av WEBP formatet berodde mer på programmet. Gdal2tiles hanterade WebP bättre än PNG medan ctb-tile hade olika prestanda beroende på indata formatet.

Programmens förmågor och begränsningar diskuterades baserat på den tillgängliga dokumentationen för de olika programmen. Gdal2tiles hade överlägset mest funktioner medan ctb-tile kunde generera tiles i största antalet format. En intressant observation var att gdal2tiles har multiprocessering in den nyaste versionen vilket är det primära syftet bakom gdal2tiles_mp och gdal2tiles_parallel. Gdal2tiles och ctb-tile hade också mest dokumentation och baserat på mängden GitHub stjärnor också den största användarbasen. Gdal2tiles hade också en funktion för att skapa kod för en frontend webbkarta som kan serveras direkt med raster tile filerna. En funktion som ingen av de andra hade.

Storleken på filformaten som används för raster tiles skilde sig ganska mycket. Storleken för raster tiles var i genomsnitt s 87.18kb för PNG, 8.60kb för JPEG, och 5.19kb för WebP tiles. Förhållandemässigt innebär det här att PNG tiles tar upp 10.1 gånger mera utrymme än JPEG tiles och 17.8 gånger mera än WebP tiles.

En annan intressant observation var att COG filen tog mindre utrymme än samma område som PNG tiles. Det går nämligen att ta in COG-filer direkt på front-enden via till exempel geotiff.js biblioteket istället för att använda en traditionell tile-server arkitektur. Båda formaten är förlustfria, har intern tiling och överblickar (overviews). Det här kan få COG att verka som ett mer attraktivt val om man vill undvika extra genererings steg och spara utrymme. Det har inte testats hur mycket långsammare den här arkitekturen är.

Diskussion och slutsatser

Inget program var ständigt bättre än alla andra, med hänsyn till både benchmarkingen och jämförelsen av funktionerna, vilket gjorde resultaten diskutabla. I efterhand anser jag att programmen gdal2tiles_mp och gdal2tiles_parallel inte skulle behövt tas upp i det här arbetet eftersom ingen av dem har blivit uppdaterade på flera år och mängden stjärnor på github indikerar att få använder dem. Dokumentationen som fanns för de här två programmen var också väldigt bristfällig. Deras exekveringstider var i flera aspekter bäst men med tanke på att inga extra finjusteringar eller optimeringen var gjorda till programmen, eller GDALs interna inställningar, finns det ännu utrymme att förbättra resultaten för gdal2tiles och ctb-tile.

Eftersom de är bättre dokumenterade och i bredare användning utgör de en bättre grund för att utforska optimeringar.

Den största begränsningen till dessa jämförelser var hur olika programmen var vilket gjorde det svårt att göra rättvisa jämförelser. Detta arbete kan till sist ses som en grund för ytterligare benchmarking av verktyg som genererar raster tiles för web map tjänster. Benchmarking arbetsflödet är förmodligen det mest användbara det här arbetet har åstadkommit tillsammans med samlingen information som kan hjälpa en utvecklare få en rudimentär förståelse för raster tile servrar.