DynoViz: Dynamic Visualization of Large Scale Satellite Data

Zhuocheng Shang
University of California, Riverside
USA
zshan011@ucr.edu

Suryaa Charan Shivakumar
University of California, Riverside
USA
sshiv012@ucr.edu

Ahmed Eldawy
University of California, Riverside
USA
eldawy@ucr.edu

Abstract

With the rapid increase in publicly available satellite data with high-resolution, so did the demand on interactive visualization of this data on a web map. This data is often high-resolution, with up-to daily three-meter resolution data, and multi-spectral with up-to 15 bands per datasets. Users in various fields need to interactively explore terabytes of this data via a web-based interface to choose the right data for their projects. Unfortunately, existing systems are either single-machine with limited scalability, or they do have limited support for dynamic visualization. Moreover, most systems pre-render visible bands only, i.e., RGB, and ignore other bands even though many scientific domains are more interested in other bands, e.g., infrared. This work introduces DynoViz, a novel system for dynamic web-based scalable visualization of satellite data. It visualize big satellite data on a regular web-based interface through multilevel dynamic-resolution visualization. The design consists of three main parts. First, a pre-generation process produces a limited set of select static tiles stored on disk. This process is controlled with a parameter to balance interactivity and disk usage depending on the application needs. Second, a dynamic on-the-fly generation technique uses a raster index to provide real-time visualization of high-resolution regions of the map. Third, a web-based interface provides client-side rendering of tiles according to the user requirements and can handle multi-spectral data with no additional overhead on the server. Experiments with terabyte-scale datasets show that DynoViz is up-to an order of magnitude faster than other distributed systems in the pre-generation phase and uses 60 times less disk storage without sacrificing the interactivity.

CCS Concepts

• **Human-centered computing** → **Geographic visualization**.

Keywords

Raster data, Spatial data, Satellite imagery, Big data, Visualization, multilevel

ACM Reference Format:
Zhuocheng Shang, Suryaa Charan Shivakumar, and Ahmed Eldawy. 2024. DynoViz: Dynamic Visualization of Large Scale Satellite Data. In *SIGSPATIAL '24: 12th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, October 29–November 1, 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3681763.3698475

SIGSPATIAL '24, October 29 - November 1, 2024, Atlanta, GA, USA

ACM ISBN 979-8-4007-1143-5/24/10
https://doi.org/10.1145/3681763.3698475

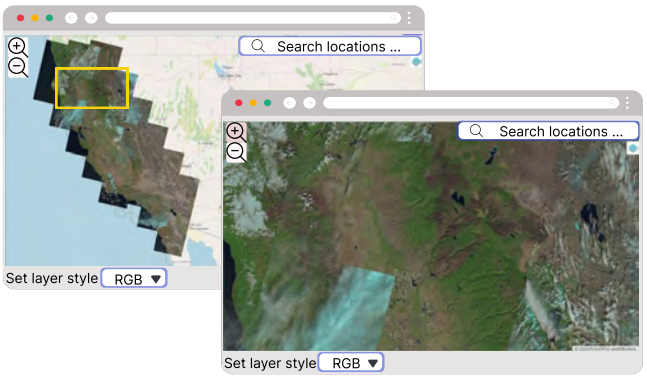


Figure 1: Browser view of the Landsat8 California imagery

1 Introduction

With the rapid growth of satellite imagery, there is an increasing demand for data exploration across various fields. According to NASA, its Earth Science data is expected to exceed 245 petabytes by 2025. Domain scientists in several fields, such as agriculture, climate forecasting, disaster monitoring, and marine research [3, 13, 16, 19], heavily rely on satellite data. One of the important features that data scientists demand is interactive exploration of big datasets.

Consider a repository that contains daily satellite data over several years. Data scientists would like to download subsets of this data that match their need. One possible solution is to directly download all data in given region and time and then visualize it using a GIS tool, e.g., QGIS or GDAL. However, this method is very time consuming due to the huge size of this data and the slow visualization process of GIS tools. Eventually, scientists will only keep a small subset that satisfies their application requirements, e.g., low cloud coverage and certain change patterns that can be visible in the data. Due to the variety of applications, each scientist could be interested in specific bands of the data, e.g., land cover or soil moisture, or computed indices, e.g., normalized difference vegetation index (NDVI). Junior scientists and students can waste months looking for the right dataset due to the slow process.

Figure 1 illustrates an alternative approach where users can browse the data in a web-based interface to quickly select the datasets of their interest. Users can easily navigate the map in real-time, e.g., zoom in or out, and adjust the visualization style, e.g., RGB or NDVI, to match their needs. While there are existing tools that follow a similar approach, e.g., USGS EarthExplorer, they suffer from one or more limitations. First, they only support low resolution

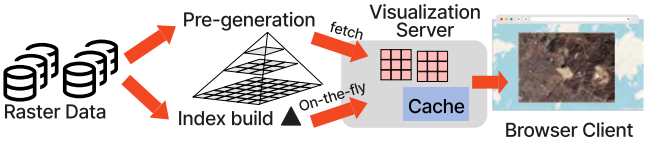


Figure 2: Overview of the proposed visualization process

visualization to reduce the resource demand on the server. Second, many of these systems are not as interactive as the users desire taking up-to tens of seconds to load a different dataset. Third, they only provide pre-rendered images that show the visible bands (RGB) and do not allow the user to style the images differently.

Looking into existing raster processing systems, we found that they all suffer from one or more limitations that make them unsuitable for a visualization system like the on in Figure 1. First, there are several *single-machine* systems such as GDAL [7], Oracle GeoRaster [17], and ArcGIS [1], but they are not scalable due to the overhead of loading data from disk. On the other hand, there are *distributed tools* including Apache Sedona [22, 25], GeoTrellis [8] and Mapbox [15] but they are also limited. For example, Apache Sedona process each raster file in isolation and fails to process real satellite data that consist of thousands of files as one coherent dataset. GeoTrellis requires pregeneration of huge amounts of images to provide the desired level of interactivity which consumes too much processing time and disk storage. Mapbox has similar scalability limiations for large and high-resolution satellite data.

Additionally, satellite data differ from visual RGB images as they often store more than visible bands. For example, some of them contain analytical data with up-to 16 bits per band and up-to 15 bands per dataset. Some of them represent non-visual data, e.g., land cover, soil moisture, temperature, or crop type. This requires more flexibility in the visualization system to allow the users to visualize all this data in a user-friendly style according to the user needs. However, existing systems [7, 8, 15], primarily support generating RGB visual images only, limiting their utility for detailed analytical purposes.

This paper introduces a new approach, DynoViz, for large-scale satellite data visualization. It aims to overcome existing limitations with the following features: (1) DynoViz is a distributed tool for visualization and interactive exploration. (2) It supports loading a single file or a folder containing multiple files. (3) It provides a fixed-resolution visualization, one image covering the entire dataset. (4) It also provides dynamic multi-resolution visualization with multi-level zoom capability. (5) DynoViz equips self-halted pre-generated static image processing and on-the-fly generation. (6) It utilizes a user-configurable parameter to reduce disk usage during self-halted pre-generation process. (7) DynoViz utilizes a lightweight index to minimize on-the-fly rendering time, providing a real-time experience with minimal disk overhead. (8) The proposed tool can fetch and visualize 16-bit non-visual and multi-band analytical imagery. (9) This work also enables customized front-end styling features, such as viewing selected layers or pixel calculation results.

Figure 2 illustrates the pipeline of DynoViz. First, there is a pre-processing step that creates two separate outputs. It generates multi-resolution static images to boot strap the visualization with a configurable parameter to balance the preprocessing time and visualization response time. It also generates a light-weight raster index to allow real-time visualization for the regions that are not pre-generated. Second, it contains a visualization server which uses both static images and the raster index to provide web access to visualized data for any region and zoom level. Third, DynoViz contains a web-based client application that allows the user to explore the data on an interactive map using the tiles served by the server. It also allows the user to change the visualization style on the client side, e.g., RGB or NDVI.

To test DynoViz, we implemented a prototype that run the pre-processing step using Apache Spark and the visualization step using a Scala-based server and JavaScript-based client. Experiments show that the proposed system is up-to an order of magnitude faster in the data preparation step and uses sixty time less disk storage than other Spark-based baselines. At the same time, it maintains the same level of interactivity of baselines.

The rest of this paper includes: Section 2 covers the related work. Section 3 gives the background of this work. Section 4 overviews the proposed work. Section 5 describes the pre-generation and on-the-fly generation strategies. Section 6 describes experimental evaluations of the proposed work. Section 7 concludes the paper.

2 Related Work

This section reviews relevant work in spatial data visualization through web browsers, focusing on both vector and raster data approaches. A common method for visualizing spatial data is through platforms like Google Maps or Bing Maps, which fetches pre-generated static images at various resolutions, allowing users to dynamically explore the data by zooming in and out on a web browser. Spatial data visualization can fall into two main categories: vector and raster visualization, which can be implemented using either a single-machine or distributed approach. Therefore, we begin by discussing vector data visualization and optimization methods, followed by both single-machine and distributed approaches for raster data visualization.

There is a large body of work for visualizing vector data including points, lines, and polygons. Single-machine systems are usually very versatile but cannot scale to big data [4]. Recent work on distributed systems [6, 23, 24] address this problem and can scale to large-scale data. Some techniques use specialized indexes and query processing methods [9, 20] to further optimize the performance. However, the visualization of raster data presents different challenges due to its unique file structure and query processing methods.

There are tools that visualize **raster data on a single machine** including GDAL [7], Oracle GeoRaster [17] and ArcGIS [1]. However, single-machine methods face limitations in terms of efficiency and performance. Additionally, GDAL requires processing an entire image to create the visualization, which can be inefficient, as it necessitates an additional step of combining multiple files into one. This also means that GDAL cannot handle multiple files simultaneously. This limitation is problematic in real-world scenarios where a coherent dataset often consists of multiple files.

There are some recent systems for **distributed raster data** processing including Apache Sedona [22, 25], GeoTrellis [8] and

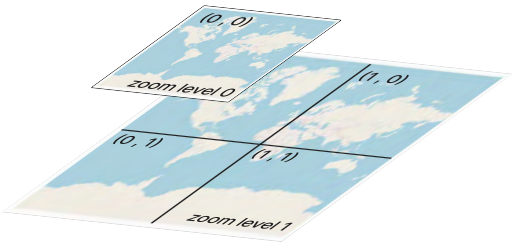


Figure 3: Example of tiled web map at zoom level 0 and 1

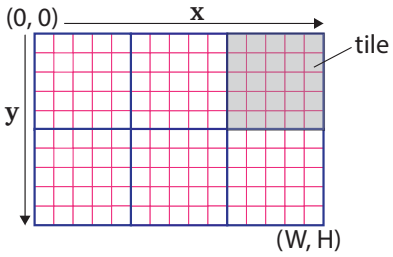


Figure 4: Example of raster data and tile

Mapbox [15]. However, Apache Sedona is currently limited to processing small individual images and not large multi-file datasets. GeoTrellis faces challenges in producing images at high resolution due to significant disk usage requirements. Both GeoTrellis and Mapbox are limited to using only RGB bands and do not support loading analytical satellite images or applying customized styling. This lack of ability to load analytical imagery limits data scientists from conducting complex analyses.

Other studies [5, 10, 26, 27] differ in scope as they focus on visualizing query results from specific query ranges rather than on the visualization and exploration of entire raster maps. Additionally, works like [11] focus more on different tiling strategies, which is also different from the pre-generation of large-scale analytical imagery. Other studies, such as [2, 12], focus on visualizing scientific data but do not specifically contain geographic information such as longitude and latitude. Other work [12] only deals with RGB images. Cloud-based solutions like [14, 18] that use formats like OME-TIFF do not involve the process of generating static tile images, as tiles can be directly fetched from the URL without pre-generation. However, relying on the OME-TIFF approach limits data scientists from the flexibly of choosing datasets or exploring any range of data they already have in hand.

The proposed work, DynoViz, is designed specifically for scalable and dynamic raster visualization. It builds on Apache Spark which addresses the limitation of single-machine systems. It build a tile-based visualization structure that can be easily integrated into web-based maps. To solve the scalability problem with high-resolution data, it proposes a dynamic approach that uses a user-configurable tuning parameter that balances preprocessing time, storage overhead, and real-time processing overhead. Finally, it supports visualization of analytical data by allowing the user to define a custom visualization style in the browser depending on the user requirements.

3 Background

This section provides background information about the tiled web maps, which are widely used in web map visualization, and raster data.

3.1 Tiled Web Map

The visualization of maps through a web browser generally employs tiled web maps. A tiled web map consists of billions of tiles, each measuring 256x256 pixels. At zoom level 0, the entire map is represented by a single tile. As the zoom level increases, the number of tiles multiplies by 4 each time, so it contains 4 tiles at zoom level 1, as shown in Figure 3. A coordinate system identifies each tile denoted as z, x, y, where z represents the zoom level, and x and y mark the location of each tile on the grid on that level. During visualization, the system locates the corresponding x, y coordinates for a specific zoom level and arranges them adjacently within the browser window. For this system to work efficiently, the server should be able to deliver any tile in a fraction of a second.

3.2 Raster Data

A raster file can be represented as a two-dimensional array, starting from the top-left corner at $(0, 0)$ and extending to the bottom-right corner with width and height (W, H), as shown in Figure 4. Each pixel is located in the grid space at its top-left corner, denoted as (x, y). Additionally, each pixel represents a rectangular region of the Earth's surface. A coordinate reference system (CRS) defines the geographical projection that maps between grid pixels and the Earth surface. This geographical relation differentiates it from other scientific datasets, such as biomedical images. A single file is internally organized as a set of non-overlapping tiles, each containing a group of pixels within the raster data, and the figure illustrates the concept of tiles. Each tile has its own starting (x, y) location. Real raster data consists of tens of thousands of raster files, each representing a different geographical region and with a different CRS.

4 Overview of DynoViz

This section gives an overview of the proposed work, as illustrated in Figure 2. We discuss the challenges and motivations that led to this approach. The dynamic multi-resolution process consists of two parts: self-halted pre-generation of static images and on-the-fly image generation. A threshold value is defined to determine when to halt the pre-generation process. For the on-the-fly generation, we use an index that efficiently identifies the corresponding image and its geographical extent, which accelerates the rendering process. Finally, this work supports customized styling on the front-end. Below, we provide an overview of those components and the subsequent section delves into the details.

4.1 Dynamic Multi-Resolution Viewer

In this work, we introduce the dynamic multi-resolution viewer approach to tackle the challenges of visualizing large-scale and high-resolution satellite images. Our approach emphasizes dynamic

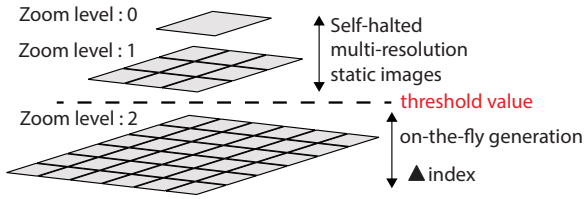


Figure 5: Example of the two-phase process for generating images from zoom levels 0 to 2

multi-resolution, allowing multilevel zoom functionality for web-based visualization. The process is divided into two strategies: self-halted pre-generation of multi-resolution static images and on-the-fly generation during real-time browsing. Several factors led to the adoption of the two-phase approach. The first is the time-consuming process of generating all levels of static images, and the second challenge is the significant disk usage required to store the exponentially growing number of images.

Large data size and higher resolution introduce significant computational challenges. Reprojecting all images to the Web Mercator reference becomes increasingly time-consuming as data size, coverage, and resolution increase. Various of Coordinate Reference Systems (CRS) within a single dataset is one of the factors contributing to this time-consuming process. It is necessary to understand that raster images in the same dataset can have different CRSs. As described previously, a CRS defines a geographical projection that maps a region of the Earth surface to a two-dimensional space corresponding to a raster dataset. A single raster product might have multiple CRSs depending on the geographical location, such as UTM zones, due to how satellites scan the Earth surface. Projecting all raster files into the Web Mercator CRS to ensure an accurate display of these images on a web browser within a unified map is necessary. Failing to unify the CRS could result in displaying raster images in the incorrect order. This process involves applying geographic transformations to each raster image and requires resampling methods, such as nearest neighbor, average, or bilinear interpolation. As data size and pixel resolution increases, these interpolation methods become more time-consuming to apply. Consequently, generating all levels of images onto disks becomes inefficient.

Disk usage is another significant challenge. Generating all levels of images onto disks demands significant disk space to store the exponentially growing number of images as zoom levels increase. Moreover, with large datasets, moving and sharing these images becomes inconvenient. In section 5, we will discuss more details about how this work is designed to address the mentioned challenges. Therefore, in this work, we introduce an innovative self-halted multi-resolution static image generation process. This method is a step forward in minimizing computational time and disk storage.

We introduce an quick response on-the-fly generation for the rest of the high zoom levels. The on-the-fly generation strategy aims to provide a real-time browsing experience when users zoom in to explore more details. We optimize the on-the-fly process with an index file. The challenge with on-the-fly generation is the rendering time. Considering an end-to-end user experience, users expect real-time visualization with a tolerance of up to 0.5 seconds of delay. This tolerance allows us to generate some images on-the-fly in the browser rather than relying only on pre-generation. To meet this requirement, the on-the-fly generation must be efficient, taking no longer than 0.5 seconds to create an image. We have adopted the use of an index to speed up this process. With the help of the index, we can exclusively query the requested region of the dataset. This technique enables us to offer the real-time visualization feature that users expect. The Figure 5 illustrates an example of how these two phases manage to generate and visualize all required images for all levels. The self-halted multi-resolution static images process creates images for different zoom levels. A threshold value is defined to halt the process at a certain zoom level, which helps to reduce storage usage. For example, in the figure, we only pre-generate images from zoom level 0 to 1 based on the threshold value. Following this, as shown in the figure, we use on-the-fly generation to visualize images at zoom level 2. The index file helps determine the query range (z, x, y) from the entire dataset and only loads the required parts. More details are described in section 5.

4.2 Customized Style for Analytical Imagery

DynoViz goes beyond generating traditional *PNG* or *JPEG* visual images. It supports loading both RGB visual raster images and multi-band analytical imagery, allowing users to customize the frontend style. Users can choose any layer from the image for viewing, such as the near-infrared band from the Landsat8 dataset. They can also view results from pixel calculations, such as the NDVI (Normalized Difference Vegetation Index), which is widely used in agriculture and determined through pixel-wise calculations. Additionally, since the data used for calculating NDVI typically comes from analytical satellite imagery where each pixel is represented by 16-bit numbers rather than 8-bit RGB images, DynoViz is designed to handle and visualize 16-bit data on the web, extending its capabilities beyond just RGB visual images. The implementation details are discussed inin section 5.

5 Dynamic Multi-Resolution Viewer Technique

In this section, we detail the strategy for dynamic visualization. This section describes the motivations. Then we discuss the steps involved in creating pre-generated static images and methods for reducing disk usage. Additionally, we introduce the on-the-fly process and discuss optimization techniques using an index file.

5.1 Natural Properties of Raster Data

Existing work in vector visualization [20] utilizes a threshold to avoid pre-generating all data based on predicting the emptiness of the query result. However, a similar tool for raster visualization is still lacking. Raster data differs from vector data in that raster data can have different distributions. This difference necessitates a distinct metric for measurement. To develop a new metric, it is essential first to understand the natural properties of raster data. Unlike vector data, such as points, which can be skewed (e.g., traffic data in Los Angeles is dense while suburban areas are sparse), raster data tends to show binary skew, with 1 indicating data presence and 0 indicating no data. For example, in satellite images covering multiple countries, land areas contain data, while oceans and other excluded countries are typically blank.

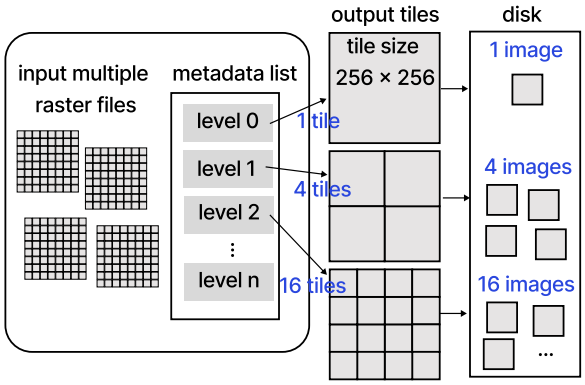


Figure 6: Resampling in parallel

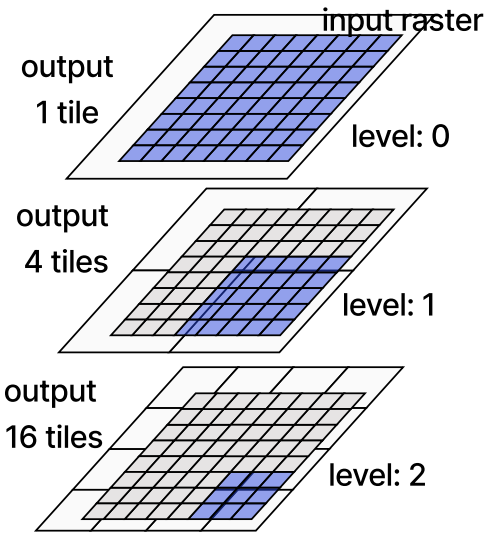


Figure 7: Examples of density definition

5.2 Self-Halted Multi-Resolution Static Images

Recall that pre-generating all levels of static images is time-consuming. To reduce computation time, we designed this process to run in parallel. This approach uses Spark, a distributed system, to handle the pre-generation process by simultaneously reprojecting and generating result images.

Since the fixed-resolution view is straightforward, we will describe the technical details at a high level. We read all raster images and reproject them to the Web Mercator reference. During this process, we also ensure all raster images have the same resolution. The resolution can match the original source data or be a lower resolution or customized value. The output is an image that represents the entire dataset.

The process of generating dynamic multi-resolution images begins by taking the maximum zoom level from the user input and generating all the necessary metadata for each zoom level, starting from 0. This metadata includes the geographical extent, tile size, and pixel resolution for each zoom level in Web Mercator. To ensure that all images at the same level have consistent pixel resolution, we first calculate the total raster width and height in pixels for the entire dataset. Then, we determine the pixel resolution by dividing the raster width and height by the geographical extent in the Web Mercator reference system. Finally, we store all the metadata in a list. We resample the original dataset into required zoom levels simultaneously as illustrated in Figure 6. The figure shows how this method takes multiple raster files and a list of metadata as input. It generates all output tiles at various zoom levels concurrently, with each tile being 256 by 256 pixels in size. Finally, it simultaneously writes all tiles to disk as pre-generated static images, ensuring efficiency and scalability.

This work introduces a new metric to further reduce computation time and address the issue of heavy disk storage usage. Building on the natural properties of binary skewed raster data described earlier, we define a metric that helps determine the maximum pre-generation level based on raster geographical coverage, input raster pixel numbers, and output raster pixel numbers. The following content defines this metric in detail.

The input and output raster size in pixels is important because, for instance, when downscaling an image, more pixels from the input raster are needed to generate a single pixel in the lower-resolution output image. Conversely, if the input and output pixels are similar in size, we may only need one input pixel to resample the corresponding output pixel. As shown in the Figure 7, generating one output tile at zoom level 0 necessitates using the entire raster. The colored cells indicates the number of pixels needed to generate the lower right corner tile in the output layer. As the zoom level increases, the number of pixels required from the input raster to produce each output tile decreases. For instance, at level 2, only a small number of pixels from the input raster are needed to generate one output tile. Thus, we can halt the pre-generation process at the point where roughly needs one input pixel is required to produce each output pixel.

Additionally, the actual raster data coverage area also impacts the metric. For example, we might have high resolution data that covers only a small region, such as California, and low resolution data that covers the entire world, yet both datasets have the same total raster width and height. In this case, the high resolution California data would require more levels of pre-generated static images compared to the globally expansive low resolution data. Similarly, even with the same resolution, a larger geographical extent would require more pre-generated static images. We define this new metric as a density value that helps select the maximum zoom level for pre-generation.

Density is defined as the number of pixels needed from the input rater in order to create one pixel in the output raster with the effect of the raster data geographical coverage. We have calculated the number of pixels in the entire raster dataset as it in the Web Mercator reference and this result is defined as *IRasterSize*.

We define the output tile size as *OTileSize*, and in a constant size 256 by 256 pixels. Note that the total number of tiles in the web browser increases by a factor of four as the zoom level increases, and the total pixel numbers for each level should be $OTileSize * 4^i$. The effect of the raster data geographical coverage is defined as *GeoCoverage%*, which is calculated as the percentage of geographical area of the input raster data relative to the entire world at zoom

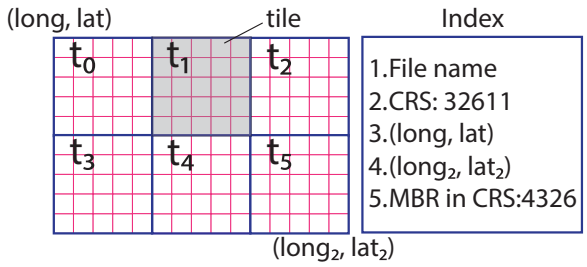


Figure 8: Example of raster file with EPSG:32611, and its index record. The raster contains 6 tiles, each has tile ID.

level 0. This is also a fixed value. Therefore, we calculate the density as:

$$\frac{IRasterSize * GeoCoverage\%}{OTileSize * 4^i}$$

where i is the zoom level, and $i \geq 0$.

From this equation, it is straightforward to determine the density for any zoom level. The input raster size, IRasterSize, remains fixed within the same dataset, and the output tile size, OTileSize, is also constant. The GeoCoverage% can be calculated by dividing the geographical extent values of the raster data in Web Mercator by the total area of the entire world. The pixel resolution for each level in a web map is based on dividing the Earth's circumference in meters by the tile size in pixels at each zoom level, as demonstrated below:

$$\frac{Earth's\ Circumference}{(256 * 2^i)}$$

where Earth's Circumference is 40,075,000 in meters, and i is the zoom level, $i \geq 0$. For example, the calculations show that at level 12, the pixel size on Web Mercator is approximately 38 meters, which is close enough to Landsat8 resolution.

It is important to note that DynoViz does not render upsampled images, as upsampling during the pre-generation process is unnecessary. It is acceptable to stop generating images at a maximum zoom level that closely matches the original pixel size. For example, a 38 meter resolution is sufficient compared to a 30 meter resolution. Since mapping services like OpenLayers can perform upsampling on the browser side, there is no need to generate these upsampled or higher-resolution images in advance. According to the pixel resolution formula, zoom level 13 yields a 19-meter resolution, which would require upsampling and is therefore unnecessary if the original resolution is 30 meters.

5.3 Lightweight Geo-Shadow Index

Before jumping into the on-the-fly generation process, we first build an index on the raster dataset. The index is created by reading the header of each raster file and generating a record for each one. As illustrated in Figure 8, it provides a raster file with geographical extent in longitude and latitude with CRS in 32611. Each index record includes the file name, the default CRS of the original file (32611 in this case), its extent in the original CRS, and its coverage as a minimum bounding rectangle (MBR) in the CRS:4326 reference. The MBR in CRS:4326 is in case the original raster is not in the standard geographical reference system. Figure 9 offers a clearer representation of the index structure. In the figure, we have multiple

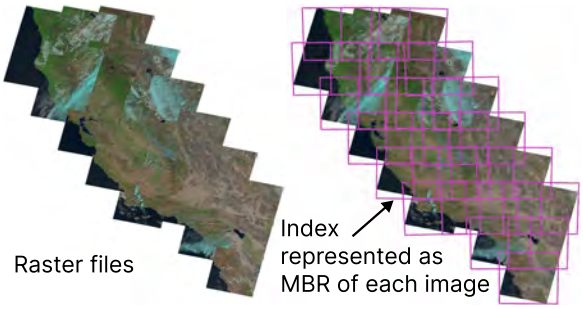


Figure 9: Example of raster files and the index representations associated.

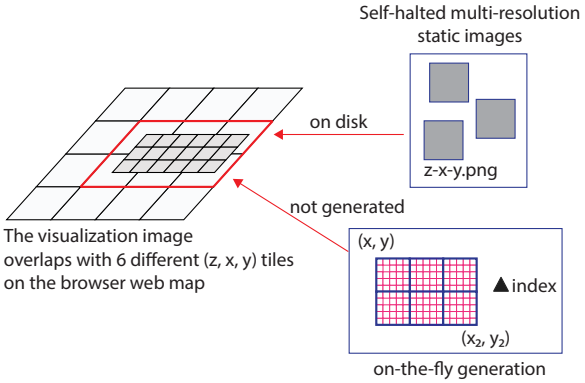


Figure 10: Example of on-the-fly generation process

files that cover the entire state of California, and each raster file is associated with a corresponding rectangular shape, indicating that the index is built based on the geographical extent rather than the actual data content. The index building time is just few seconds.

The index helps to filter out query ranges. For example, in Figure 10, we have a query range of 6 (z, x, y) tiles. We can go over the index file quickly to find the overlapping rasters. By applying CRS transformation on the MBR extent of longitude and latitude to Web Mercator (x, y), we can easily check for overlap. This index file avoids opening the raster file multiple times and only requires reading a few bytes of text data. We also adapt the default index structure in raster data, which uses tiles, to help filter and retrieve only the necessary parts of the data. This approach will be explained further when discussing on-the-fly rendering.

5.4 On-the-fly Hawk Render Tiles

The on-the-fly process occurs on a local machine, generating the required images as the user explores using a web browser. DynoViz can render the raster images in the web browser within 0.5 seconds as users zoom in or hover around. The following content describes the method used to achieve this performance.

Frstly, it checks whether the raster image within the current browser window exists on disk as a pre-generated static image. If it does, the web can fetch the required static images directly; otherwise, it generates the missing images on-the-fly. Figure 10

illustrates an example of this process. It first looks for a static image on disk with the required (z, x, y) tile name. If a match is found, the web browser fetches the static image. Otherwise, it invokes on-the-fly generation.

When processing images on-the-fly, we obtain the required (z, x, y) coordinates from the web browser. We then refer to the index file to identify overlapping images by checking if the MBR overlaps with the required range. If the file needs processing, we determine which parts of the tiles need to be loaded from the entire raster file. This method ensures that we only load the necessary parts of the raster data, reducing the processing time, improving efficiency and lessening memory usage.

We take advantage of the raster file format and manage to retrieve only the required tiles. In Figure 8, it shows a raster with 6 tiles, each associated with a tile ID. A tile is a group of pixels and is usually the same size across the entire raster file. Since raster tiles are located adjacently in an orderly manner, it is easy to calculate the tile ID from its extent location, tile size, and the number of tiles in its row and column order. We then store all required tile IDs into a list. After obtaining all the required tile IDs, we start reading the tiles based on these IDs. A typical raster file includes a header that stores location information, indicating where each tile offset starts within the file. The tile ID can determine this offset, allowing us to access the tile starting point and read the necessary tiles directly instead of scanning the entire file.

The last step involves reprojecting the images to the current exploring zoom level with the corresponding pixel resolution in the Web Mercator reference. The on-the-fly generated result is represented as an array. The web browser can directly render these values from the array buffer, and we also save the images to disk.

5.5 Personalized Frontend Styling

Instead of generating *PNG* or *JPEG* static image formats, Dynoviz creates *npy* files from raster data. These npy files can be fetched using an array buffer, allowing users to select which layers to visualize directly in the web browser. Using an array buffer not only supports the visualization of different bands but also enables pixel calculations in the browser directly. For example, the frontend can efficiently perform pixel-wise NDVI calculations using the array data. This eliminates the need for extensive preprocessing and provides a more dynamic and interactive user experience. By leveraging *npy* files, DynoViz allows for real-time data manipulation and analysis, enhancing the capability to work with detailed and complex satellite imagery datasets. DynoViz also offers personalized frontend styling tailored to specific domains

6 Experiments

This section presents an experimental evaluation that compares DynoViz with the distributed system GeoTrellis [8]. We utilized GDAL command-line tools for comparison and found that it is inefficient, and distributed methods proved to be over 100 times faster. It requires 484 seconds to generate completed pyramid images on the Landsat8 California dataset, while DynoViz only costs 36 seconds. Furthermore, GDAL also require user to merge all files into one large file so that is can generate the pyramid, which is not piratical if have large scale of data. Apache Sedona [23] only generates images at one zoom level, lacking the dynamic visualization capabilities offered by this work. It also fails to take multiple files as one entire coherent image. Mapbox [15] does is not open-sourced for uploading large scale raster data, making it incomparable in this study.

Table 1: Landsat8 Raster Datasets

Dataset	# pixels	Size	Resolution
Riverside	237.97~M	689.86~MB	30~m
California	1.57~B	4.40~GB	30~m
US	52.53~B	146.78~GB	30~m
World	798.47~B	2.18~TB	30~m

The following sections detail the datasets first. The analysis starts with an end-to-end user experience, focusing on the overall performance and disk usage of the entire process. Subsequently, a detailed examination studies the performance and disk usage of the pre-generation process and on-the-fly generation. This section also analyzes the density at various levels and the time required to build indexes. For the pre-generation experiments, all tests are conducted in parallel, while the on-the-fly generation tests are run on a single machine. Lastly, we demonstrate a real-world case of visualizing satellite images on web browsers using both *DynoViz* and *GeoTrellis*, providing a more direct insight into actual scenarios.

Section subsection 6.2 presents the overall performance of the entire end-to-end user experience study, including time performance and disk usage. The time metrics encompass the entire process, while disk usage is calculated based on the size of pre-generated images. Section subsection 6.3 further delves into the detailed analysis of each process phase in *DynoViz*, including pre-generation, index building, and on-the-fly generation. This section examines how the choice of density impacts the entire end-to-end user experience. Lastly, section subsection 6.4 demonstrates the rendering results on a web browser, indicating a real-use environment. This section also tests the systems ability to visualize analytical raster imagery and apply customized styling.

6.1 Datasets and Hardware

Table 1 includes the datasets used in this experiment. In this experiment, Landsat8 [21] data covers varying geographic scales, from cities to the global scale. Landsat8 has a resolution of 30 meters. The table also lists the total number of pixels and the data size for each dataset.

We run *DynoViz* and *GeoTrellis* on a cluster with one head node and 12 worker nodes running Spark 3.5.0. The header node has 128 GB of RAM, 2×8 core processors, and Intel(R) Xeon(R) CPU E5 - 2609 v4 1.70GHz processor. Each worker node has 64 GB of RAM and 2×6 core Xeon processors on CentOS Linux.

6.2 End-to-end User Experience Study

The Table 2 investigates the **overall time** required for both methods to deliver the final results, and it contains two parts, one detailing the pre-generation time, and the other describing the fetch time from the web browser. The pre-generation phase also includes the

Table 2: Pre-generation performance comparison between DynoViz and GeoTrellis; Max zoom level = 12; Density for DynoViz on Riverside:0.14; California:0.4; US:10.61; World:185.

Dataset	Pre-generation time (s)				Tile fetch time (ms)		
	DynoViz		GeoTrellis		DynoViz		GeoTrellis
	level	time	level	time	level	time	time
Riverside	0 - 2	21.4	0 - 12	205.9	0 - 2	0.83	0.94
	3 - 12	9.0			3 - 12	484.4	
California	0 - 4	27.7	0 - 12	231.8	0 - 4	0.96	0.97
	5 - 12	9.0			5 - 12	61.2	
US	0 - 6	189.4	0 - 12	1793.9	0 - 6	0.88	0.85
	7 - 12	9.8			7 - 12	16.3	
World	0 - 8	6284.4	0 - 12	Error	0 - 8	0.81	Error
	9 - 12	37.3			9 - 12	427	

Table 3: Pre-generation disk usage comparison between DynoViz and GeoTrellis; Max zoom level = 12; Density for DynoViz on Riverside:0.14; California:0.4; US:10.61; World:185.

Dataset	# of files		size (mb)	
	DynoViz	GeoTrellis	DynoViz	GeoTrellis
Riverside	3	3,184	0.035	37.31
California	7	35,116	0.082	411.52
US	119	337,022	1.39	3,949.48
World	43,581	Error	510.71	Error

construction of indexes for *DynoViz* with all details provided in Table 5, and it indicates that the index building time remains fast, even with millions of files. In this experiment, *DynoViz* terminates the pre-generation process when when specific density thresholds are reached for different datasets, whereas *GeoTrellis* pre-generates images at all levels. The termination point is chosen based on the web browser rendering time, which should be less than 0.5 seconds. A detailed breakdown of this is provided in later studies. The fetch time encompasses both on-the-fly generation and direct fetch from the disk. Our observations indicate that *DynoViz* is up to 89% faster than *GeoTrellis* in the pre-generation phase. The on-the-fly generation takes longer when it has to process more levels, resulting in the Riverside dataset requiring more time than both the California and US datasets. As the level and data size coverage expand for world dataset, *DynoViz* also takes longer for on-the-fly generation as writing more images, yet the time remains under 0.5 seconds or 500 milliseconds. Variation in density across different datasets also results that both the US and World datasets having higher densities at their respective halt levels, thereby increasing the running time. Additionally, the disk fetch time is almost negligible for both methods. The slight differences in running times are caused by variations in cluster performance.The table also show the *GeoTrellis* is not robust that it has error when reprojecting to the Web Mercator reference. Moreover, the baseline requires first to reproject all data into zoom level 13 then can provide the reprojection for different customized levels to the correct geographic extent, resulting takes longer time.

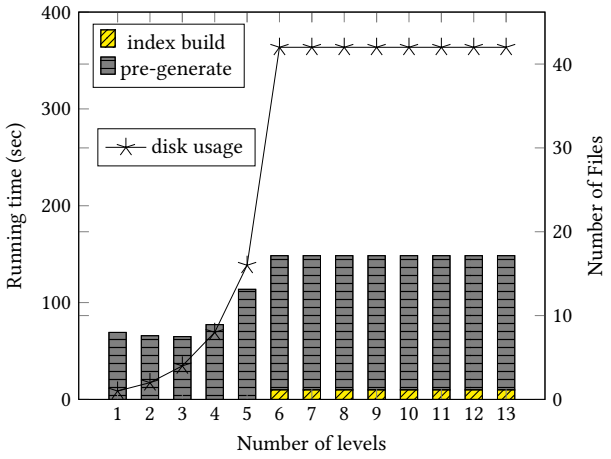


Figure 11: Break down study of end-to-end user experience on US dataset; X-axis as different max zoom level

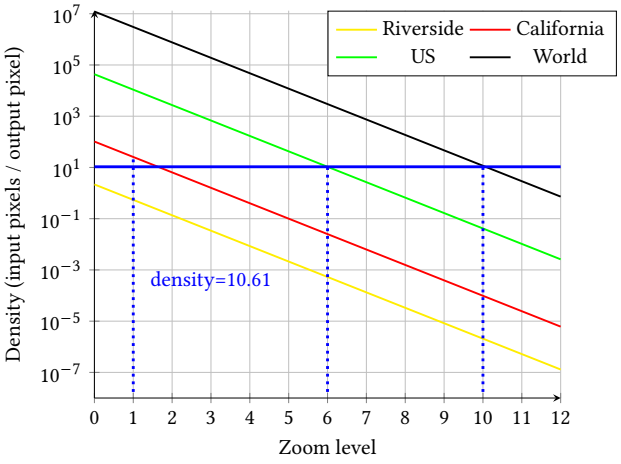


Figure 12: Density study on Landsat8 dataset

Disk usage results are shown in Table 3. Since *GeoTrellis* generates all 12 levels, users must pre-generate all images, resulting in a massive number of files, approximately 60 times more than this proposed work requires. It also produces unnecessary blank images, as evident in Figure 13 (b), which displays black images, and this is the other reason that it requires more disk usage.

6.3 DynoViz Break-down Study

We further experiment with different strategy combinations within DynoViz. This study first breaks down performance and disk usage requirements for each phase. It aims to find the optimal balance that minimizes pre-generation time and disk usage while still providing a real-time experience in on-the-fly rendering. Using the US dataset as a case study, this experiment measures the time required to pre-generate images at various zoom levels and the corresponding file sizes output to disk. It also examines on-the-fly rendering and density at different levels.

Table 4: On-the-fly performance study on US dataset

Zoom level	Density	Tile fetch time (s)
		on-the-fly
0	43474.14	30.5
1	10868.53	25.1
2	2717.13	9.75
3	679.28	4.7
4	169.82	3.6
5	42.46	1.0
6	10.61	0.29
7	2.65	0.1
8	0.66	0.1
9	0.166	0.035
10	0.041	0.02
11	0.0000034	0.017
12	0.00000021	0.016

Table 5: Index build process performance over all datasets

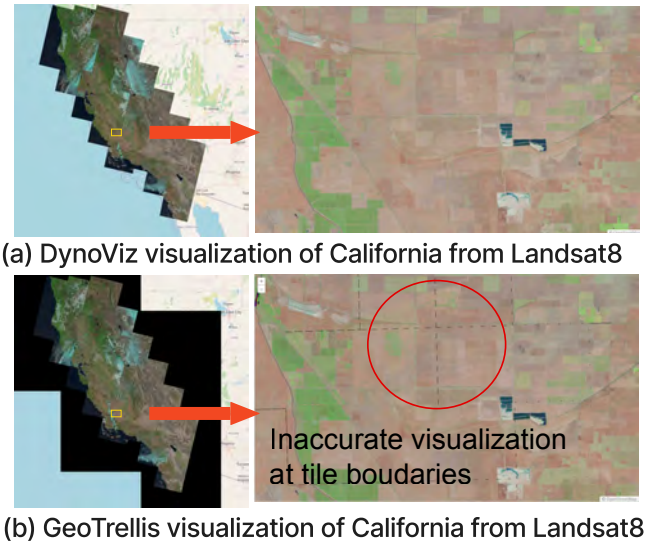
Dataset	# files	time (s)
Riverside	5	8.9636
California	27	9.0404
US	861	9.8343
World	15,135	37.3326

The results of the **pre-generation** study are displayed in Figure 11. As more image levels are produced, the total number of files increases exponentially, and this number remains constant once pre-generation is halted at a specified density threshold. In this study, we halt the process at level 6, so the number of files remains constant beyond this point. The time required for generation also increases with the number of files produced. Index building time remains consistent throughout, as it is conducted once for the entire dataset. It is important to note that index building is only needed when on-the-fly generation is required, which is why the figure includes the index building time starting from zoom level 7. This figure also illustrates that users have the flexibility to customize the zoom level according to their requirements.

This experiment also explores the **density threshold** to find the optimal balance between disk storage usage and on-the-fly generation. As illustrated in Figure 12, the stop level varies, and larger datasets and broader geographic coverage requires the pre-generation of more levels of images. This study reveals that the density value is influenced not only by the pixel size of the images but also by the geographic extent of the entire dataset.

For the **index building** study, we can notice from the Table 5 that the process takes only a few seconds, even for the largest dataset, which requires just 37 seconds. This efficiency is because the process only reads header files, which are just a few kilobytes in size, and the index is stored in a simple text format.

For the **on-the-fly** experiment, we run all tasks on local machine. We study the time for viewing the entire dataset on browser side. The Table 4 shows the density of each zoom level along with the time for fetching one image on that level. The tile fetching time is divided into two ways, the time taken to generate one tile on-the-fly and the time to retrieve one tile directly from disk. The on-the-fly generation time is calculated by averaging the time usage of creating at most 1,000 non-empty images. Direct fetch experiments measure the average time taken to read images from the folder. As showing in the table, it requires more time to generate with low zoom level since it requires more pixels from input in order to produce on pixel in the output. We can find that as density goes close smaller, it requires less time. Meanwhile, the disk fetch time is not affected by levels and is negligible.



(a) DynoViz visualization of California from Landsat8

(b) GeoTrellis visualization of California from Landsat8

Figure 13: Visualization on web browser

By analyzing this table and assuming users can tolerate a latency of 0.5 seconds, we can terminate the pre-generation process at a density greater than 1. In this study, the halt occurs at zoom level 6 with a density value of 10.61 while still providing a real-time rendering experience in the browser. Thus, we can pre-generate all levels from 0 to 6 and then create levels 7 to 12 on-the-fly. From the Figure 11 we know that the pre-generation time for levels 0 to 6 results in 189 seconds with a bit of disk usage. The Table 5 shows that the index building time is only 9 seconds. The Table 4 indicates that the tile fetching time is approximately 0.1 seconds. Combining all these, the total time for this approach is 198.1 seconds, which significantly reduces the overall end-to-end user experience compared to *GeoTrellis*, which requires 1,793 seconds.

6.4 Visualization on Web Browser

The Figure 13 demonstrates how both methods render California dateset from Landsat8 using OpenLayers. From the figure, it is apparent that *GeoTrellis* not only produces more unnecessary blank images, but also fails to correctly generate images. The right part of the figure provides a zoomed-in view, revealing black strips in the images generated by *GeoTrellis*. These are caused by incorrect reprojection, as it processes only within individual tiles and neglects the pixels at the borders that require reading from adjacent tiles, resulting in the observed empty spaces in the images.

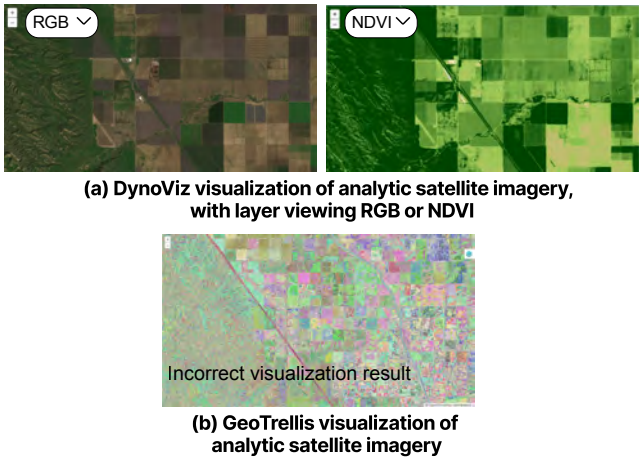


Figure 14: Visualization analytic satellite imagery

The Figure 14 illustrates how both methods render analytical satellite images, each pixel containing four values: blue, green, red, and near-infrared (NIR). Each value is with the 16 bit data type. Figure 14(b) shows that *GeoTrellis* fails to create and visualize the RGB versions of multi-band images because it creates the *PNG* files, which do not support 16-bits data. Figure 14(a) demonstrates how DynoViz visualizes analytical satellite images and also allows for styling by selecting different layers or NDVI to display. The NDVI is calculated using the following pixel-wise formula:

$$\frac{(NIR - RED)}{(NIR + RED)}$$

, where the results range from 0 to 1. All these styling operations are performed on the browser side without any preprocessing.

7 Conclusion

In this work, we introduce *DynoViz*, a novel approach for visualizing large-scale satellite imagery. This method includes both a pre-generation phase and on-the-fly generation. It allows for stopping the pre-generation at a specific zoom level to minimize disk usage while still providing a real-time web browser experience. Additionally, this approach offers customized styling options, even for analytical imagery. The experiment results demonstrate that this new approach is 89% faster than the baseline in terms of the overall end-to-end user experience. Additionally, the on-the-fly generation ensures real-time rendering in the web browser, with times below 0.5 seconds. The customized styling is also meaningful, as it is completely handled in-browser without any preprocessing and delivers more accurate visualizations than the baseline. In the future, this work could include the capability to dynamically predict the optimal density point at which to halt pre-generation based on the resolution and geographical extent of the dataset.

Acknowledgments

This work is supported in part by the National Science Foundation (NSF) under grant IIS-2046236 and by Agriculture and Food Research Initiative Competitive Grants no. 2020-69012-31914 from the USDA National Institute of Food and Agriculture.

References

[1] ArcGIS 2022. ArcGIS. https://www.esri.com/en-us/arcgis/about-arcgis/overview.
[2] Emmanuel Bertin, Ruven Pillay, and Chiara Marmo. 2015. Web-based visualization of very large scientific astronomy imagery. *Astronomy and Computing* 10 (2015), 43–53.
[3] Daniel Brown et al. 2015. Monitoring and evaluating post-disaster recovery using high-resolution satellite imagery–towards standardised indicators for post-disaster recovery. *Martin Centre: Cambridge, UK* (2015).
[4] Isabel F Cruz, Venkat R Ganesh, et al. 2013. GIVA: a semantic framework for geospatial and temporal data integration, visualization, and analytics. In *Proceedings of the 21st ACM SIGSPATIAL*.
[5] Ahmed Eldawy, Mohamed F Mokbel, et al. 2015. Shahed: A mapreduce-based system for querying and visualizing spatio-temporal satellite data. In *2015 IEEE 31st international conference on data engineering*. IEEE, 1585–1596.
[6] Ahmed Eldawy, Mohamed F Mokbel, et al. 2016. HadoopViz: A MapReduce framework for extensible visualization of big spatial data. In *2016 IEEE 32nd ICDE*. IEEE.
[7] GDAL/OGR contributors. 2022. *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation. https://gdal.org
[8] GeoTrellis 2019. GeoTrellis on Spark. https://github.com/wri/geotrellis-zonal-stats/blob/master/src/main/scala/tutorial/ZonalStats.scala.
[9] Saheli Ghosh and Ahmed Eldawy. 2020. Aid*: a spatial index for visual exploration of geo-spatial data. *IEEE Transactions on Knowledge and Data Engineering* (2020).
[10] Mahmudul Hasan, Faramarz F Samavati, and Christian Jacob. 2016. Interactive multilevel focus+ context visualization framework. *The Visual Computer* (2016).
[11] Elsayed E Hemayed, Eman M Farghal, and Samir I Shaheen. 2010. Pyramid-based multiresolution tiling for interactive viewing of remote sensing images. In *2010 International Computer Engineering Conference (ICENCO)*. IEEE, 128–133.
[12] Won-Ki Jeong, Jens Schneider, Stephen Turney, Beverly E Faulkner-Jones, Dominik Meyer, Rudiger Westermann, R Clay Reid, Jeff Lichtman, and Hanspeter Pfister. 2010. Interactive histology of large-scale biomedical image stacks. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 1386–1395.
[13] Daniel Kachelriess, Martin Wegmann, Matthew Gollock, and Nathalie Pettorelli. 2014. The application of remote sensing for marine protected area management. *Ecological Indicators* 36 (2014), 169–177.
[14] Trevor Manz, Ilan Gold, Nathan Heath Patterson, Chuck McCallum, Mark S Keller, Bruce W Herr, Katy Börner, Jeffrey M Spraggins, and Nils Gehlenborg. 2022. Viv: multiscale visualization of high-resolution multiplexed bioimaging data on the web. *Nature methods* 19, 5 (2022), 515–516.
[15] Mapbox. [n. d.]. *Mapbox*. https://docs.mapbox.com
[16] Andreas Holm Nielsen, Alexandros Iosifidis, and Henrik Karstoft. 2021. CloudCast: A Satellite-Based Dataset and Baseline for Forecasting Clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* (2021).
[17] oracle 2023. oracle spatial. https://docs.oracle.com/en/database/oracle/oracle-database/23/geors/georaster-overview-and-concepts.html
[18] Ramon Antonio Rodriges Zalipynis and Nikita Terlych. 2022. WebArrayDB: a geospatial array DBMS in your web browser. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3622–3625.
[19] Elia Scudiero, Todd H Skaggs, and Dennis L Corwin. 2014. Regional scale soil salinity evaluation using Landsat 7, western San Joaquin Valley, California, USA. *Geoderma Regional* (2014).
[20] Akil Sevim and Ahmed Eldawy. 2021. HQ-Filter: Hierarchy-Aware Filter For Empty-Resulting Queries in Interactive Exploration. In *2021 22nd IEEE MDM*. IEEE.
[21] U.S. Geological Survey. 2024. Landsat 8. https://www.usgs.gov/landsat-missions/landsat-8
[22] Jia Yu and Mohamed Sarwat. 2019. Geospatial data management in apache spark: A tutorial. In *2019 IEEE 35th ICDE*. IEEE.
[23] Jia Yu and Mohamed Sarwat. 2021. GeoSparkViz: a cluster computing system for visualizing massive-scale geospatial data. *The VLDB Journal* (2021).
[24] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. 2018. Geosparkviz: a scalable geospatial data visualization framework in the apache spark ecosystem. In *Proceedings of the 30th international conference on scientific and statistical database management*.
[25] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. 2019. Spatial data management in apache spark: the geospark perspective and beyond. *GeoInformatica* (2019).
[26] Jianting Zhang et al. 2010. Supporting web-based visual exploration of large-scale raster geospatial data using binned min-max quadtree. In *22nd International Conference, SSDBM 2010*. Springer, 379–396.
[27] Jianting Zhang and Simin You. 2010. Dynamic tiled map services: Supporting query-based visualization of large-scale raster geospatial data. In *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*.