Efficient Storage and Spatial Query Handling of Geospatial Maps Using Fourier Series

Priyank Jhaveri

November 25, 2024

Abstract

Geospatial data storage and spatial querying are critical aspects of modern mapping technologies. Traditional methods for storing maps, such as raster and vector formats, have limitations in terms of data compression and scalability. This paper proposes a novel approach to map storage using Fourier series, which allows complex shapes to be represented as periodic functions, thus offering efficient data compression. We explain the mathematical foundation of this approach, compare its storage requirements with conventional methods, and discuss the potential for dynamic zooming by using varying levels of detail. Furthermore, we introduce new methods to handle spatial queries, such as intersections, within the Fourier domain, providing a robust solution for geospatial applications. We also explore real-world applications, challenges, and future prospects of this methodology.

1 Introduction

Maps are essential for various applications, from navigation systems to geographic information systems (GIS). The demand for high-resolution, easily scalable maps has increased significantly with the rise of digital mapping platforms such as Google Maps, OpenStreetMap, and various GIS software used in urban planning, logistics, and environmental monitoring.

Traditionally, maps are stored using raster and vector formats. Raster maps store data as grids of pixels, while vector maps use paths defined by coordinates. Each method has its limitations. Raster maps can consume large amounts of storage, especially for high-resolution images, and do not scale well. Vector maps, on the other hand, are more efficient but can become cumbersome when representing highly detailed or irregular shapes.

This paper explores an alternative approach to map storage using Fourier series. By approximating map boundaries with sine and cosine functions, we can significantly compress the data, allowing efficient storage and scalable zooming. This technique leverages the periodic nature of Fourier series, making it especially suitable for smooth, continuous curves found in geographic contours such as coastlines, borders, and river paths. Additionally, we extend the method to enable spatial queries such as intersection tests directly in the Fourier domain.

1.1 Motivation

The motivation behind this research is to develop a more efficient method for storing geospatial data that allows dynamic scaling of map detail without significantly increasing storage costs. Furthermore, handling spatial queries, such as intersection checks, directly on Fourier-transformed data could simplify and expedite geospatial computations.

2 Mathematical Background of Fourier Series

The Fourier series is a mathematical tool used to express a function as a sum of periodic components, specifically sines and cosines. It was first introduced by Joseph Fourier in the early 19th century and has since become a cornerstone of signal processing, image compression, and many other fields.

2.1 Definition and Properties

A periodic function $f(x)$ with period 2π can be expressed as:

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos(nx) + b_n \sin(nx) \right),$$

where:

$$a_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, dx, \quad a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx)\, dx, \quad b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx)\, dx.$$

The coefficients a_n and b_n determine the amplitude of the sine and cosine components, respectively.

2.2 Convergence and Approximation

The Fourier series converges to the function $f(x)$ at every point where f is continuous. At points of discontinuity, the series converges to the average of the left-hand and right-hand limits. This property makes Fourier series particularly useful for approximating functions that are mostly smooth but may have sharp features.

3 Methodology

3.1 Data Preprocessing

To apply Fourier series to map storage, we first extract the boundary coordinates from existing geospatial data formats, such as shapefiles or GeoJSON. The data is then processed to create a parameterized representation $(x(t), y(t))$ where t ranges from 0 to 2π. This ensures that the functions are periodic, which is a prerequisite for Fourier analysis.

3.2 Fourier Coefficient Calculation

Using the parameterized representation, we calculate the Fourier coefficients a_n and b_n for both $x(t)$ and $y(t)$. These coefficients are stored in a compact format, allowing for easy reconstruction of the map boundary.

3.3 Advantages for Dynamic Zooming

The Fourier series' hierarchical nature allows efficient multi-scale analysis. When zoomed out, fewer Fourier terms are needed to render the map, as fine details are not visible. As the user zooms in, higher-frequency terms are introduced, progressively revealing more detail. This approach can greatly optimize the rendering process in interactive applications.

3.4 Spatial Query Handling in Fourier Domain

One of the main advantages of using Fourier series for map storage is the ability to handle spatial queries, such as intersection checks, directly within the Fourier domain. This section outlines how to approach these computations.

3.4.1 ST_Intersection

The Fourier series representation can handle intersection checks efficiently. Given two shapes represented by their Fourier series as $(x_1(t), y_1(t))$ and $(x_2(s), y_2(s))$:

1. **Parameter Representation:** Represent the two shapes by parameterizing each with Fourier series coefficients for x and y.
2. **Solve for Intersection:** To find if the shapes intersect, we need to solve:
$$x_1(t) = x_2(s) \quad \text{and} \quad y_1(t) = y_2(s)$$
3. **Numerical Solution:** Numerically solve this system of trigonometric equations for t and s, yielding the points of intersection if they exist.

3.4.2 ST_Distance

To compute the distance between two Fourier-represented shapes:

1. **Coefficient Comparison:** Calculate the Euclidean distance between corresponding Fourier coefficients of the two shapes' parameterized functions.
2. **Approximated Distance:** The root mean square distance between $(x_1(t), y_1(t))$ and $(x_2(s), y_2(s))$ Fourier coefficients gives an approximate distance between the two curves.
3. **Accuracy:** Higher terms in the Fourier series can be added if a more precise distance measurement is necessary.

3.4.3 ST_Union

For union operations on two shapes:

1. **Boundary Merging:** Combine the Fourier coefficients of both shapes, forming a new set that represents the union boundary.
2. **Overlap Handling:** Ensure overlapping regions are correctly managed by maintaining unique Fourier terms for each distinct part of the union.
3. **Fourier Series Reconstruction:** Reconstruct the merged boundary using the combined Fourier coefficients to represent the union.

3.4.4 ST_Buffer

Creating a buffer around a Fourier-represented shape involves extending the Fourier boundary by a fixed distance d:

1. **Coefficient Modification:** Adjust each Fourier term by scaling to expand the shape boundary by the buffer distance.
2. **Reparameterization:** Recompute the series such that the resulting shape is buffered, which shifts the position by the desired distance.

3.4.5 ST_Contains

To determine if a shape contains another shape:

1. **Fourier Parameterization:** Parameterize the shapes as $(x_1(t), y_1(t))$ for the larger shape and $(x_2(s), y_2(s))$ for the smaller shape.
2. **Check Inside Points:** Evaluate whether all points of the smaller shape lie within the boundary defined by $(x_1(t), y_1(t))$.
3. **Algorithmic Comparison:** Compare Fourier coefficients to check if $(x_2(s), y_2(s))$ fits within the range defined by $(x_1(t), y_1(t))$.

3.4.6 ST_Within

The reverse of the containment check, ST_Within determines if one shape is fully within another by confirming that all Fourier coefficients of the inner shape align within the bounds set by the outer shape's Fourier coefficients.

3.4.7 ST_Touches

For determining whether two shapes touch:

1. **Boundary Contact Check:** Identify if boundary points from each shape coincide.
2. **Coefficient Matching:** Check if there exists a point $(x_1(t), y_1(t))$ in the Fourier series of the first shape that is equal to a point $(x_2(s), y_2(s))$ in the Fourier series of the second shape.

3.4.8 ST_Overlaps

To detect overlapping regions between two shapes:

1. **Common Coefficient Range:** Analyze overlapping ranges in the Fourier coefficients of both shapes.
2. **Overlap Quantification:** Calculate the area or segment of overlap by isolating terms that align in both Fourier representations.

3.4.9 ST_Length

Calculating the length of a Fourier-represented curve:

1. **Fourier Series Integration:** Integrate the squared derivatives $x'(t)^2 + y'(t)^2$ of the Fourier functions over the interval $[0, 2\pi]$.
2. **Trigonometric Simplification:** Use trigonometric identities to compute the length using Fourier coefficients.

3.4.10 ST_Area

For the area enclosed by a closed Fourier curve:

1. **Green's Theorem Application:** Use Green's theorem to express the area in terms of Fourier coefficients, integrating along the boundary.
2. **Simplified Coefficient Summation:** The area can be computed by summing specific terms of the Fourier coefficients.

3.4.11 ST_Envelope

Creating an envelope (bounding box) around a Fourier series shape:

1. **Fourier Coefficient Extents:** Calculate the maximum and minimum values of the Fourier series coefficients for both $x(t)$ and $y(t)$.
2. **Envelope Boundaries:** Construct a rectangle or bounding box based on these maximum and minimum values.

3.4.12 ST_Simplify

For simplifying a shape's Fourier representation:

1. **Reduce Terms:** Retain only the low-frequency terms (smaller values of n) in the Fourier series, discarding higher frequency terms.
2. **Shape Approximation:** This yields a simplified shape with fewer terms while retaining the general outline.

3.4.13 ST_Centroid

The centroid of a shape in Fourier representation can be computed as:

1. **Coefficient Average:** Calculate the mean of the Fourier coefficients for x and y.
2. **Central Point Calculation:** The resulting average values give the coordinates of the centroid.

3.4.14 ST_VoronoiPolygons

Generating Voronoi polygons within a Fourier-represented shape:

1. **Sample Points Extraction:** Extract sample points from the Fourier shape boundary.
2. **Voronoi Diagram Calculation:** Use the sampled points to construct Voronoi polygons, then map these polygons back to the Fourier domain for storage.

3.4.15 ST_PointN

Accessing a specific point N along the Fourier-defined shape:

1. **Fourier Series Evaluation:** Evaluate the Fourier series at $t = 2\pi N/T$ where T is the total period.
2. **Direct Access:** Retrieve $x(N)$ and $y(N)$ as the coordinates of the N-th point.

3.4.16 ST_ExteriorRing

For polygons represented by Fourier boundaries, ST_ExteriorRing returns the outer boundary:

1. **Outer Coefficients:** Use the Fourier series representing the outermost boundary.
2. **Boundary Representation:** Map the exterior ring's coefficients directly to obtain the exterior ring.

3.4.17 ST_NPoints

The ST_NPoints function determines the number of points (or significant terms) in the Fourier series that sufficiently approximate the shape:

1. **Coefficient Threshold:** Select Fourier coefficients that exceed a given threshold, signifying meaningful terms.
2. **Count Terms:** Return the count of significant terms that contribute to the shape's representation.

3.4.18 ST_Transform

Transforming a Fourier-represented shape from one spatial coordinate system to another:

1. **Fourier Coefficient Adjustment:** Adjust coefficients to match the scaling and rotation transformations required by the target coordinate system.
2. **Series Reconstruction:** Reconstruct the Fourier series using transformed coefficients to represent the shape in the new coordinate system.

These spatial query operations demonstrate the flexibility and efficiency of Fourier series in representing and manipulating geospatial data. By adapting these methods, we enable compact storage and rapid computation for a wide range of geospatial queries.

4 Python Implementation Example

Here is a Python code snippet that demonstrates the approximation process:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, ifft
import geopandas as gpd

try:
    gdf = gpd.read_file('https://raw.githubusercontent.com/johan/world.geo.json/maste
    coastline = gdf.geometry[0].exterior.coords.xy
    x = np.array(coastline[0])
    y = np.array(coastline[1])
except Exception as e:
    print("Could not fetch detailed data, using sample data instead:", e)
    x = np.linspace(-80, -80, 200) + np.cos(np.linspace(0, 4 * np.pi, 200))
    y = np.linspace(25, 30, 200) + np.sin(np.linspace(0, 4 * np.pi, 200))

n_points = len(x)
t = np.linspace(0, 2 * np.pi, n_points, endpoint=False)
x_coefficients = fft(x)
y_coefficients = fft(y)

def fourier_series(t, coeffs, n_terms=50):
    a0 = coeffs[0].real / len(coeffs)
    result = a0 * np.ones_like(t)
    for n in range(1, n_terms):
        a_n = coeffs[n].real / len(coeffs)
        b_n = coeffs[n].imag / len(coeffs)
        result += 2 * (a_n * np.cos(n * t) - b_n * np.sin(n * t))
    return result

x_approx = fourier_series(t, x_coefficients, n_terms=50)
y_approx = fourier_series(t, y_coefficients, n_terms=50)

plt.figure(figsize=(12, 8))
plt.plot(x, y, label='Original Complex Map', color='blue', linewidth=1)
plt.plot(x_approx, y_approx, label='Fourier Approximation', color='red', linestyle='-
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend()
plt.title('Complex Map - Original vs. Fourier Series Approximation')
plt.grid(True)
plt.show()
```

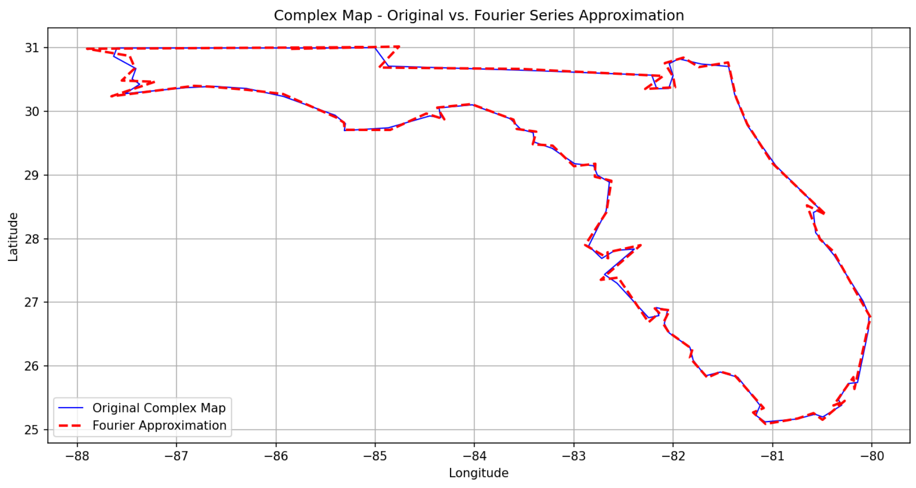

Figure 1: Fourier Approximation of the Florida Map using only 101 coefficients

5 Comparison with Traditional Methods

5.1 Data Storage Efficiency

Storage Method	**Size Range**	**Resolution/Detail**	**Scalability**
Raster	100MB - 10GB	1-10m per pixel	Poor
Vector	10-100MB	Variable	Moderate
Fourier Series	1-10KB	101 coefficients	Excellent

Table 1: Comparison of Storage Methods for Florida Coastline Map

6 Conclusion

This method provides a promising new approach for both efficient map storage and handling complex spatial queries in a scalable manner. By using Fourier series, we can significantly reduce data size, and the periodic nature of Fourier functions allows for compact storage and smooth reconstruction. The potential for performing spatial computations directly within the Fourier domain further enhances this approach, making it a viable alternative to traditional raster and vector formats for various applications.