Project Profile Finder
Due: Wednesday, October 8
CSC615: Computational Geometry

Objectives

- Input, store, and process basic geometric data (street maps, in this case)
- Apply geometric algorithm techniques to solve a geometric problem
- Use advanced data structures to help process geometric data
- Use approximation techniques to search a geometric space
- Use simplification techniques to reduce complexity (for efficient searching)
- Research, develop, and implement a strategy to compare similarity between monotonic curves

Motivation

Imagine you're a cyclist preparing for an upcoming race in another state. The course is hilly, and the elevation profile is challenging — long climbs, sharp dips, a punishing final ascent. You want to train for it, but you can't ride the actual course until race day. So you ask: *Can I find a route near me that "feels" like this one?* A route with the same rhythm of climbs and descents, similar slopes, and total distance — even if it's on completely different roads. This question, raised by a cyclist friend preparing for a distant event, is the inspiration for this project. It transforms a real-world curiosity into a rich computational geometry challenge: given a desired elevation profile, can we find a local route that matches it as closely as possible?

Overview

You are given a preprocessed, elevation-aware, directed road network in a single file (JSON Lines). Coordinates are projected into a planar metric CRS (meters), and each road segment is at most 100 m long and treated as a straight line with linear elevation change.

Your program must, given a query (C, D, P), compute a bicycle route whose *elevation profile* best matches a target profile P, starting from some location within distance D of center C. The notion of "best" will be chosen by the class, but you must clearly define, justify, and evaluate your team's choice.

Provided Input (Single File)

Road Graph File (JSON Lines): each line is one JSON record.

- **Meta:** `{"type":"meta","crs":"EPSG:3857","units":"meters","max_segment_m":100}`
- **Node:** `{"type":"node","id":`*`int`*`,"x":`*`float`*`,"y":`*`float`*`,"elev":`*`float`*`}`
- **Edge (directed):** `{"type":"edge", "id":`*`int`*`, "u":`*`node_id`*`, "v":`*`node_id`*`, "length_m":`*`float`*`, "climb_m":`*`float`*`, "slope":`*`float`*`}`

Distances and coordinates are in meters; edges are directed; segments ≤ 100 m; elevation is in meters at nodes (edge climb and slope are provided for convenience). You need only parse the file; no projection/preprocessing is required.

Routing Query

A query consists of:

$$(C, D, P)$$

- $C = (x_c, y_c)$: a center location in meters (same CRS as the graph).
- $D > 0$: a search radius (meters). Candidate *start positions* must be within Euclidean distance D of C.
- $P = \big[(d_1, z_1), (d_2, z_2), \ldots, (d_k, z_k)\big]$: the target elevation profile.

Interpretation of P. We treat $d_1 < d_2 < \cdots < d_k = L$ as *cumulative* distances (meters) from the start of the route, and z_i as the *desired elevation offset relative to the start* (meters). For example, $P = \{(20, 5), (60, 7), (140, 3)\}$ means at 20 m from the start the elevation should be $E_P(20) = E_P(0)+5$, at 60 m it should be $E_P(60) = E_P(0)+7$, and at 140 m it should be $E_P(140) = E_P(0)+3$, where $E_P(0)$ is the start elevation.

Problem Statement

Goal: Find a directed path π in the road graph that:

1. **Starts within D of C.** The route's start point S must lie on some edge or node whose (x,y) is within Euclidean radius D of C. If S lies on an edge interior, you must *split* that edge by inserting a temporary node at the projection point and interpolating its elevation.
2. **Has length $\approx L$.** Let $L = d_k$ be the target profile's total length. Your route should have total length within a tolerance, e.g., $|\,\mathrm{length}(\pi) - L\,| \le \varepsilon_L$ (default $\varepsilon_L = \max\{5\text{m},\ 0.05L\}$).
3. **Best matches the target profile P.** Define a matching score $\mathrm{Match}(\pi, P)$ that compares the route's *relative* elevation profile (relative to its start elevation) to P at the distances d_i. Your algorithm should minimize this score (precise choice below).

Route Elevation and Profile Extraction

Treat each edge as a straight segment in (x, y) with *linear* elevation from one end to the other. For a partial distance along an edge, linearly interpolate its elevation. The route's relative elevation at distance s from S is:

$$E_\pi^{\text{rel}}(s) = E_\pi(s) - E_\pi(0),$$

where $E_\pi(0)$ is the start elevation.

Profile Matching: How Should We Compare Profiles?

The core goal of this project is to find a route whose *elevation profile* matches a target profile

$$P = \big[(d_1, z_1),\ (d_2, z_2),\ \ldots,\ (d_k, z_k)\big]$$

where each d_i is a cumulative distance (in meters), and z_i is the desired elevation *relative to the starting elevation* at that distance.

But how should we define "matching"?

This is not a trivial question. A simple approach might compare the actual and target elevations only at the sample points d_i using average error. However, this can fail in practice: a route might pass through those points exactly but oscillate wildly in between, creating a very different ride experience.

Your Task. Your team must define and implement a *profile matching score* that you believe reflects how well a route aligns with a given elevation profile. You will:

- Clearly **define** your scoring metric (mathematically or in pseudocode),
- **Justify** why it is a meaningful measure of similarity,
- **Implement** it to guide your route search, and
- Be prepared to **defend** your choice during the demo.

During presentations, we will visualize each team's route profile against the target profile, and the class will discuss which approaches yielded the most convincing results — based on both numbers and visual alignment.

Suggestions (Optional). You are encouraged to design your own metric, but here are a few ideas to consider:

- **Segment-wise slope comparison:** Compare the average slope between d_{i-1} and d_i in your route to the target slope between z_{i-1} and z_i.
- **Total elevation gain/loss error:** Penalize routes with excessive climbing or descending beyond the target profile.
- **Envelope deviation:** Ensure the route stays close to the target profile at all times (e.g., using maximum or average deviation). Visual the profile being inside a tube, the smaller the tube, the better.
- **Area between profiles:** One student suggestion was to compute the total area (absolute value) between the target and actual elevation profiles — essentially integrating the absolute difference across the route.

Elevation Offset Adjustment. By default, the target profile P is defined relative to a start elevation of zero. However, in practice, your route might better match the profile if it is *shifted vertically* — that is, if we allow your profile to be raised or lowered uniformly by a fixed amount to minimize error.

Your matching score may optionally include this idea: searching for a constant vertical offset z_0 such that the difference between your actual relative profile r_i and the target z_i is minimized when computed as $r_i - z_i - z_0$. This adds another degree of freedom and complexity to the matching process, and you should explain whether or not your score includes such an offset, and if so, how that offset was determined.

Algorithmic Requirements

1. **Input parsing (JSONL).** Build an internal graph (directed). Be sure to keep a copy of this graph intact. When you report out your suggested path it should be in reference to this input graph (or else it would not be possible to recreate the route precisely!)
2. **Start selection within D.** Build a spatial search to find edges/nodes within distance D of C. Consider starting at a point along an edge instead of just at the beginning of an edge. If you choose to have the starting point lie along an edge, split that edge by inserting a temporary node with interpolated elevation and lengths.
3. **Route search to length L.** Implement and describe a method to construct a potential route of length $\approx L$ that minimizes your matching score. You will need to design, implement, and justify your approach.
4. **Profile extraction.** Given a potential route π, compute $E_\pi^{\text{rel}}(d_i)$ via piecewise linear interpolation and report the full piecewise-linear elevation profile P_π.
5. **Profile matching.** Given the target profile P and a potential profile P_π, compute the matching score $\text{Match}(P_\pi, P)$.
6. **Complexity analysis.** Provide asymptotic time/space for both preprocessing (creating a data structure) and query handling in terms of N nodes, M edges, $k = |P|$, and any additional parameters such as error rates. This will be part of your report.

Running and I/O

Your report should include instructions on how to compile and run your program both for building the data structure and answering queries.

Input: The input for the database will be a JSONL file as described above. You are encouraged to preprocess this structure and store into another data format for faster query processing later.

The input for the queries will consist of the following:

- An integer q on a single line indicating the number of queries
- q lines each describing a profile query:
  - The first two integers indicate a center C_x and C_y.
  - The next integer indicates the distance D
  - The remaining integers are in pairs d_i and z_i representing cumulative distance from start of profile and relative elevation.

Output: The output should consist of one line for each query.

- The first two floating points s_i and t_i representing the partial amount of the first and last segment to use (see below).

- This is then followed by the ordered list of traversed edge IDs (directed) separated by spaces.

Since the start and end of the route might be inside a segment, we use fractional amounts to indicate what portion to use. The value s_i indicates how far from the start of the first segment to the end of the first segment to begin the route. E.g. $s_i = 0.25$ would be about a quarter of the way from the start of segment 1 to the end of segment 1 (or start of segment 2). The value of t_i is the same but for the last segment.

Note: Using $s_i = 0$ and $t_i = 1$ would use both segments completely.

Deliverables

1. **Code**: The code to both preprocess the data and to answer queries (can be different program files).
2. **Report (2–3 pages)**:
   - (Short) formal problem restatement and assumptions.
   - Your matching metric (definition, weights, tolerance); why this metric is appropriate; and how this metric was computed.
   - Your search algorithm including how the road data was preprocessed to expedite searches and how queries were answers.
   - Complexity analysis (theoretical) of both the preprocessing step and the query reporting; and a brief empirical runtime on provided sample data.
   - Future work: Limitations and potential improvements.

Grading Rubric (100 pts)

Code and Report (50 points)	
Route identified meets all requirements	10
Route matches profile best	10
Matching metric (clarity & justification)	10
Algorithmic design & complexity analysis	10
Code clarity & documentation	5
AI usage provided	5

Presentation and Demo (25 points)	
Prepared	5
Clarity of techniques used	5
Justification of techniques used	5
Demo	5
Presentation length	5

Peer Evaluation (25 points)	
Communication of members	9
Workload balance	8
Collaboration	8

Constraints and Notes

- You may assume the graph is connected enough to find routes of length $\approx L$ in the search area, but handle failures gracefully (report “no feasible path within tolerance”).
- All geometry is planar Euclidean (in meters). Elevations are already embedded.
- You may use standard libraries for JSON parsing, plotting, and basic geometry. If you use additional packages, list them in your report.
- You may use AI tools including LLMs. But you must indicate them in your report.
- **Teams**: collaboration is allowed and encouraged (see course policy). Each team submits one codebase and report, with a short division-of-labor note.

Starter Suggestions (non-binding)

- Build a spatial index (grid hash or k-d tree) over edges to find candidates near C quickly. Project C to each candidate edge, keep those within D.
- Let P define target slopes over intervals $[d_{i-1}, d_i]$. When expanding a path, prefer edges whose slope is near the next target interval's slope, subject to staying on track to meet L.
- After an initial route is found, try local edits (swap a subpath along an alternate corridor) to reduce profile error without changing length much.
- Consider a hierarchy approach that computes approximate profile matches within estimated ranges and searches substructures that have potential to give good matches.

Presentation and Demo

On the due date, during class, each group shall present their algorithmic solution, analyze their algorithm's efficiency, and demonstrate their code on both the sample input query set and some new query sets presented that class. The first 15 minutes of class will be available to test running on the new data sets but then presentations will begin.

Group Work and Submissions

- You are welcome to work in groups of up to 5 people.
- You should include the list of all members in the comments and the report.
- Only one member of the team needs to submit the code.
- Every member should participate in the presentation and demo of the code.

Tips

Here are a few tips to get you started:

- Work with smaller test cases first. Efficiency is important but it should be correct first.
- For efficiency, I suggest using a hierarchical approach, a mix of graph simplification and quadtrees or k-d trees.