TSR

Designing an off-road, user-centered routing algorithm to promote on-foot transport.

University of
St Andrews

George Pestell (200007413)

Supervisor: Prof. Graham Kirby

Department of Computer Science
University of St. Andrews

Date Submitted: January 12, 2025

This dissertation is submitted for the degree of
Integrated Masters in Computer Science

Abstract

This dissertation addresses the challenge of improving accessibility for off-road walking and running by developing a route-planner that extends beyond the limitations of solely on-road/path routing. Drawing on developments from research in areas of robotics and search and rescue, in addition to building upon work done by Evans [1] at the University of St. Andrews, a modular cost-function framework is introduced. This framework allows diverse mapping related data sources to be easily integrated into a mathematical model of terrain features determining the optimal routes selected. Example cost-function presets are evaluated and compared to demonstrate the versatility of this approach for users with different needs and abilities. Finally, a discussion of potential future projects is given, emphasizing the potential utility of such an application.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is 9968 words long, including project specification and plan.

Contents

1 Introduction

Route planner applications allow users to enter a start and end point on the earth's surface, and outputs a suggested travel route between them. They are ubiquitous in both business and consumer settings, with popular consumer tool Google Maps seeing over 1 billion users per year [2].

However, these consumer tools almost all generate routes solely utilizing pre-defined paths and roads. Whilst this is ideal for motor vehicular transportation where off-road travel is almost always unfeasible or illegal, on-foot and bicycle transport offers far greater flexibility in terms of traversable terrain.

These routing tools fail to leverage any off-road travel not pre-defined as a path, which contributes contributes to the low uptake of healthier, and more environmentally friendly transportation methods [3], with recent research finding that the lack of safe routes, and the perceived danger when sharing roads with motor vehicles were two primary deterrents to commuting by walking or cycling [4], [5].

Research into and application of off-road route planning technologies have so far been limited to very specialist fields such as search and rescue [6], and robotics. This project builds off of some of the ideas explored by this research, along with a recent master's thesis by Evans [1] who made a first attempt at creating a more generalized off-road route planner. This project aims to develop and document a framework to model complex relationships between data relevant to off-road terrains, and the user's abilities and preferences, to generate safe and individualized routes, with the goal to improve accessibility to healthier transport methods.

2 Context Survey

2.1 GIS Software

GIS is a category of software which aim to integrate various mapping-related data sources together. They enable visualization and analysis using this data, and are used extensively where mapping data is used.

Popular industry tools use databases to represent this map data in tiles, as mapping data converting large areas of the real-world take up a huge amount of data, and chunking enables only necessary parts to be loaded and processed.

Tools such as GDAL enable the use of GIS data without an underlying database, enabling datasets to be created as files with data layers. These files use specialized GeoTiff and GeoJson formats (amongst others), storing mapping related metadata that can be used to translate pixel positions to lat/long positions, as well as other useful information.

2.2 Shortest Path Search Algorithms

The fundamental problem solved by route planners is a form of shortest-path search problem. Shortest-path search algorithms have a rich history of research and study. Their objective is to find the route with the minimum cost between two given nodes, in a graph where nodes are connected by edges. The two given nodes are often labelled source and target implying directionality of travel. Cost refers to the difficulty of traveling between nodes. Cost is determined by weights, which can be applied to nodes, edges, or other graph features, and are used by a cost function which calculates a cost score for traversing between two nodes. Weights represent some quantifiable metric related to difficulty of travel. A simple example would weigh edges based on their length, and the cost would reflect the distance traveled. Thus, minimizing this finds the shortest path.

2.2.1 Dijkstra's Algorithm

In a seminal piece of research on the topic, Dijkstra [7] found a solution guaranteed to find the optimal route between the two nodes. It achieves this by calculating the minimum cost from the source node to all nodes in the graph, ensuring all potential routes to the target node are evaluated.

To begin, each node's minimum cost is set to infinity, as we have no route to that node so far. The minimum cost of the source node is set to 0, as traversing from a node to itself costs nothing. We then calculate the cost from the current node to each connected node, and update the relevant minimum costs to the calculated cost between the current and connected node, plus the cost at the current node, only if that total value is lower than the connected nodes minimum cost. Initially, with node minimum costs set to infinity, this will almost always be true. This keeps a running total of the minimum cost to get from the source node to each node. We then select the next node with the lowest minimum cost that hasn't already been searched. We mark it as searched, and repeat the calculation and update of costs of connected nodes (ignoring connections to nodes that have already been searched).

The original implementation of Dijkstra's algorithm terminated once all nodes were searched, generating the best route from a source node to all nodes. However, when we mark a node as searched, we are certain to have found the optimal route to it, as it has the minimum current cost of all nodes not already searched. Therefore, all nodes that may possible lead back to that node will have at least an equal cost due to costs being non-negative. Therefore, Dijkstra's algorithm is a best-first search algorithm, and so can be terminated upon finding the target node. This early termination is often referred to as uniform cost search [8].

Whilst Dijkstra's algorithm and uniform cost search are guaranteed to find the optimal route between the source and target node, the original time-complexity for Dijkstra's original algorithm is $\mathcal{O}(\mathbf{N}^2)$, whereas the optimal implementation of uniform cost search has a time-complexity of $\mathcal{O}((\mathbf{E}+\mathbf{N})\log\mathbf{N})$, where $\mathbf{E}$ is the number of edges, and $\mathbf{N}$ is the number of nodes [8].

2.2.2 A* Algorithm

Despite its successes, Dijkstra's algorithm and uniform cost search only select the next node on the graph to search based on the minimum total cost from the source node. This means that there is no intuition in the algorithm to direct towards the target, and thus routes are found to particular nodes incidentally. This often results in a lot of unnecessary cost calculations for nodes that are not a part of the final route.

A* extends Dijkstra's algorithm, giving predictive power to node selection. This is done through heuristics, which inform the algorithm's search order in hopes of finding the best route to the target node earlier. Heuristics are metrics that are relatively cheap to calculate, and aim to predict the remaining cost to the target node. This prediction is added to the minimum cost of the unsearched nodes, meaning nodes are selected to minimize the predicted total route cost.

With a perfect heuristic, this would be a perfect greedy best-first algorithm, finding the optimal route to the target node first. However, to ensure the best-first search characteristic, and thus ensuring the best route to the target is found, the heuristic merely has to be admissible.

An admissible heuristic is one that never overestimates the true remaining cost to the target. If a heuristic is inadmissible, then is it possible that a worse route will be explored first, as the algorithm overestimated the cost of a shorter path. The admissible heuristics that exist for a particular search problem depends on the exact formulation of the search problem. For example, euclidean distance is admissible for the simple distance edge weighting example.

Without a heuristic, the A* algorithm falls back to selecting nodes with the minimum cost, and thus performs the same as Dijkstra's algorithm.

For a terrain sensitive routing algorithm, it may be possible to construct an admissible heuristic, however the task is complex due to the many factors that may influence off-road traversal cost.

2.3 Search Graph Representations

To apply a shortest-path search algorithm to a route planner, the real-world environment must be represented by a graph structure. Existing route planners using pre-defined roads and paths construct search graphs using junctions connecting roads/paths, and dead-ends as graph nodes, with the sections of road/path connecting those nodes as edges. These are weighted by the segment distance and road speed, with modern advances include traffic conditions, and fuel efficiency into the edge weights.

Representations focusing on the connections between roads/paths are topological graphs. They are particularly useful for vehicular transport because the geometric features such as road shape and exact positions of the segments of roads are irrelevant to route selection. Topological representations dramatically reduce the number of nodes in the graph, and

the amount of data that needs to be encoded for each node (e.g. storing a single distance metric instead of a vector road path), thus improving search performance.

However, for off-road search applications, topological representations of real-world terrain are less suitable than geometric graph representations. This is because the level of freedom over areas of terrain are significantly greater than for roads and paths and individuals are free to leave roads and terrains at any time, not just at junctions. Geometric representations preserve the angles and distances in the geometry of the graph.

2.4 Surface Meshes

Surface meshes are a 3D digital models of objects and surfaces. They are constructed by specifying vertices of points along the surface of the object in 3D, along with connecting edges and faces.

Surface meshes are the most popular method of 3D object representation in computer graphics applications, as they accurately model the geometry of objects and surfaces including terrains.

Surface meshes representing the terrain are often called terrain meshes, and are often 2.5D due to restrictions in the datasets. 2.5D models are the same as 3D models, except that every x,y position can have a maximum of one z vertex associated, meaning occluded areas are not represented (e.g. convex cliff overhangs).

Surface meshes are created either through modelling software, or through scanning technologies such as LIDAR, tomography, time of flight scanners, etc. Scanning techniques are popular for creating meshes of real-world objects as their geometry can more accurately be represented.

2.5 Digital Elevation Models

Digital elevation models (DEMs) are the datasets constructed by scanning technologies to represent the topography of the earth's surface.

Large-scale terrain meshes of the earth's surface are often constructed using radar data, that is restricted to 2.5D measurements, as it is restricted to a single point of measurement. LIDAR data is often used for true 3D model terrain representations, but this data has much larger storage and processing requirements, is limited in coverage, and often has strict licensing requirements.

The most common DEM format is raster data, providing a grid of elevation points created from the raw scanning data. Contour lines offer a vector representation which are more compact, but are less widely available, and require more processing to create surface meshes as the contours must be converted to points before beginning the mesh generation.

2.6 Delaunay Triangulations

The process of creating Delaunay triangulations aims to take a point set, and create a surface mesh where each vertex (point) is connected by edges and faces to other vertices [9]. For each edge, a circle could be drawn maintaining these two properties:

(1) vertices in the given edge are on the boundary of the circle, and

(2) all other vertices are outside C

Changing the edges connecting vertices to meet these criteria promotes the generation of triangle faces with as close to equilateral triangles as possible, maximizing the minimum angle. This approach minimizes the number of very thin triangles [9]. This is important for creating useful surface meshes, as very thin faces increases the risk of precision errors representation.

Creating surface meshes using just triangles is the industry standard for computer graphics and video games, due to the wealth of research into trigonometry giving efficient geometry calculations [10]. Triangle-based meshes also are able to represent any surface accurately [11].

When used to construct surface meshes of continuous surfaces (such as the earth's surface), the resulting meshes are known as triangular irregular networks (TINs).

Constrained delaunay triangulation extends the basic triangulation process, but adds an additional specification. Constraints are pre-define edges between vertices that cannot be modified. Therefore, these edges may break the original rules defined. Otherwise, the aim is to generate as close to the original triangulation as possible [12].

This project is the first to apply constrained delaunay triangulation for the purpose of representing vector paths and feature borders directly.

2.7 Coordinate Systems

Coordinate systems define how an x, y data point relates to the real world. There are two fundamental types of coordinate systems:

- Global coordinate systems. Represent the earth as a 3D surface, with points representing the x, y offset from an origin point on the surface of that object.
- Projected coordinate systems. Flatten the surface in 3D space to the 2D plane. Used to create maps, or to display the earth on any display. It is impossible to flatten the globe shape of the earth to a 2D plane without causing some distortion in some areas. The most common projection used today is the Web Mercator projection, and is used by Google Maps.

The World Geodetic System (WGS84) global coordinate system specifies latitude/longitude positions, where latitude refers to the relative angle away from the equator (-180 to 180 degree range), and longitude specifies the relative angle from the prime meridian line (-180 to 180 degree range).

The Mercator projection is a projected coordinate system based on a cylindrical map projection. It is the most popular projection used to create and display maps, as it creates a rectangular map preserving navigation angles. However, its origin at the center results in points further away being increasingly stretched out, making distance calculations increasingly inaccurate.

The Universal Transverse Mercator (UTM) projected coordinate system splits the earth's surface into 60 zones, using the center points of each zone as the origin for a mercator projection, minimizing warping of points within that zone. Distances are also normalized so that one x,y point equates to 1 meter, making distance calculations simple and accurate.

According to The CGAL Project [13], constructing surface meshes of the Earth's surface is more effective when using projected coordinate systems. Global coordinate systems, typically expressed in degrees (spanning -180° to 180° longitude and -90° to 90° latitude), require high-precision decimal representation to maintain accuracy. In contrast, UTM coordinates use linear units (meters) relative to a specific point, allowing for simpler and more precise representation with lower-precision data values. Additionally, surface mesh construction often assumes a flat plane, whereas global coordinate systems inherently describe angles on a curved sphere.

2.8 Existing Research

2.8.1 Terrain Sensitive Cost Function

[1] specifically focused on creating a route planner for humans walking, unlike most other research into off-road route generating routes for robots or specialist off-road vehicles [6], [11].

The cost function implemented enabled other data sources to be used by the cost function as features. The data had to be tagged in some way onto the faces of the mesh generated. The cost function itself had a modular design, starting each evaluation with a default cost of 1. Two categories of features were defined:

- Edge features. Defined as either traversable, or untraversable. Untraversable edges automatically result in an infinite cost for that edge, unless a traversable feature also applies which overrides untraversable features. This enables interactions such as bridges over water.
- Speed features. Define a customizable speed-factor. Where tagged on the mesh, multiply the current cost by some number inversely proportional to the influence the feature has on the walking speed. This was used to model the fact that paths are generally

faster to traverse than off-road terrain. The impact of gradient was also modelled using a speed feature.

The benefits to this approach is the modularity enables new features to be created and defined. As features do not affect the underlying mesh, caching is easily implementable, as sections of a mesh are consistent across searches. Additionally, checking for boolean edge features is performant, and allows calculation of cost using speed features to be skipped entirely if untraversable edge feature apply.

However, the drawbacks to this approach exist. First, the cost function itself is limited in its modelling capacity. For example, many terrain features in the real world interact in complex ways. This approach requires the entirety of that complexity to be written in a single module, as data from other modules cannot transfer data. Additionally, tagging all features onto the mesh in this way can often lead to an inaccurate mesh. For example, paths are often represented as vector lines, and are often small, this implementation tags the whole face as a path if any part of the face contains a path — without any ability to modify the mesh, the accuracy of features are wholly dependent on having a higher resolution DEM dataset than the resolution of the feature data.

2.8.2 Graph Representation

Perkins, Marais, Gain, *et al*. [11] applied a real-time variant of A* and TINs to generate routes for a robot traversing off-road terrain. The nodes of the TIN represented search nodes, and search edges represented mesh edge as well as conceptual edges allowing traversal over the connected face. The TIN was constructed using DEM data, with feature data tagged after mesh construction onto each face. The additional search edges across faces prevented solely traversing over edges where gradient changes. The performance trade-off of using the TIN representation were found to be promising, as dynamically sized triangles allowed few nodes to represent areas of little detail.

Evans [1] chose a regular square-based surface mesh to represent the graph, also based solely on elevation data. Search nodes were used to represent mesh vertices, and search edges represented mesh edges. Using regularly sized faces is not optimal in terms of memory usage or performance, as one can imagine a large flat field, where many small squares could be simplified with no loss of data into a representation using one large rectangle.

Evans' results found that the routing time and mesh generation scaled linearly proportional to the number of nodes in the graph, which may offer a strong performance benefit if the graph structure could be optimized.

2.8.3 Factors Impacting Off-Road Walking

Most of the research into quantifying factors influencing travel time is limited to single factorial studies in non real-world environments. This poses a challenge as quantification

of feature influence is required to construct a cost value to be optimized in an algorithm.

2.9 Existing Tools and Data Sources

2.9.1 Delaunay Triangulation

Several librarys offer delaunay triangulation implementations:

- tin-terrain

  A command line tool written in C++ designed to convert GeoTIFF DEM files into optimized meshes. Open source MIT licensed and available on Github. Offers standard Delaunay Triangulation with no constraints, but has built-in simplification of consistent gradients. Limited documentation.

- Fade (2D and 2.5D)

  A C++ library specifically designed to integrate Delaunay Trainagulation of DEM data. Offers a flexible class output structure. Restrictive academic license, with a maximum number of components at any one time imposed. Offers simplification and processing algorithms specifically designed for terrain meshes. Has comprehensive documentation.

- CGAL

  A C++ library suite offering computational geometry algorithms. Has an open source license, and is well documented. Offers standard and constrained Delaunay triangulation, as well as a large amount of mesh processing tools.

Whilst Fade2.5D is the most complete and tailored library for creating surface mesh TINs, the limitation on the number of vertices that can be used with the academic license severely limits the scale of routes that can be generated. As this project is exploratory, this would require a lot of effort to initially go into ensuring the mesh is well optimized before work on the search algorithm and cost function can be tested. CGAL's license does not come with these restrictions, and additionally offers many point set and mesh processing tools that may be useful in optimising the router later.

CGAL additionally is well maintained, unlike tin-terrain, which has been archived, and not updated since 2019.

One draw-back to this library is it's complexity. Due to offering many algorithms with rich configuration options, they require manual adjustment and testing to find the right settings, as they are not specifically tuned for terrain mesh generation. This is also a benefit, as it allows future experimentation with preprocessing to optimise mesh construction.

2.10 DEM Data

2.11 Terrain Type Data

2.11.1 Water Data

Whether water exists in a particular region of the terrain is a special form of terrain type, which has a special impact on

2.12 Path Data

2.13 Street Lighting

3 Requirements

The following outlines the requirements specification that was used to generate and prioritize tasks. The only change to the original specification made was in promoting the creation of ranking configuration presets to a primary objective, and the demotion of the evaluation to a secondary objective. This change was made relatively early on due to gaining an understanding that evaluation of a routing algorithm is fundamentally subjective, with only a collection of metrics hinting towards the 'goodness' of a route, especially without expertise in off-road route planning. The creation of different presets also better aligns with the overall project aim of creating an accessible and inclusive route planner.

3.1 Primary Objectives

3.1.1 Modular Routing Algorithm

The primary artifact this project will create will be a routing algorithm. This will accept a start and end point on the earth's surface, and output a set of points which, when followed from the start point, represent a suggested route over the terrain to the end point.

This route is generated through an algorithm which will construct a representation of the terrain as a search graph, where nodes represent points on the terrain, and edges represent the potentially traversable sections of terrain connecting those nodes. The algorithm then uses a cost function to determine the 'goodness' of traversal over edges, finding the optimal set of edges to traverse to reach the end point from the start point.

The exact definition of 'goodness' should be defined by the configuration of the cost function, but the base pre-set should define 'goodness' as the minimization of overall time of traversal, by considering factors affecting the estimated traversal speed.

This cost function should, at its most basic, take into consideration the following geometric and topological features of the terrain:

- **Gradient**. Walking up a steep gradient is much slower and less pleasant than walking on a flat or slightly decline gradient.
- **Terrain Type**. Where roads and paths usually have relatively consistent characteristics, terrain type influences walking speed over an area and has many interesting interactions with other features.
- **Distance**. The distance of traversal is an essential factor for any route planner as travel time is directly proportional to the travel distance.

The cost function should also be able to determine whether route are safe/unsafe, with the goal to reduce the barrier of the perception of fear to healthier transportation methods.

Along with the basic features to be considered above, the cost function should be modular. This refers to the ability for new features to be implemented such as new relevant data sources, and for the relationships between features and their impact on the cost function output to be configurable. This will allow users to specify their own terrain preferences and abilities, and enables future research to develop more complex and accurate cost functions.

3.1.2 Route Visualization

For evaluation, and improved user experience, visualizing the output in a way that is easily interpreted is essential. At its most basic, a line displaying the route should be applied onto a map or recognizable 3D mesh of the terrain. Additional information such as warnings for potentially dangerous sections will improve the transparency of the cost function's decision making, and the algorithm should be able to display where unsafe-to-traverse sections of terrain exist to warn users when following routes.

3.1.3 Multiple Cost Function Presets

To demonstrate the capacity for the ranking algorithm to cater to different abilities and preferences, additional cost function configurations should be created, to allow comparison and evaluation of the modular framework and presets themselves.

3.2 Secondary Objectives

3.2.1 Critical Evaluation of Presets

With the created pre-sets, a critical evaluation of the output will allow the success of the project to be measured, comparing the stated aims of the presets to differing decisions

between the presets.

A successful implementation should correctly account for configuration options, and differences and similarities between presets should be justifiable.

3.2.2 Additional Cost Function Features

Additional more complex features should be relatively simple to implement given the modular design of the cost function. These features should model more complex data and relationships.

- **Current Weather**. The current weather conditions in an area will affect the traversability of areas in different ways depending on factors such as the gradient of the terrain, and its terrain type.
- **Historical Weather**. The recent weather conditions in an area will often influence the traversal of areas, such as rain making areas more boggy and thus harder to traverse.
- **Fear of Heights**. There exist areas of terrain which are flat, but have sharp gradients near-by, which can have a psychological impact on traversal speed.

3.3 Tertiary Objectives

3.3.1 Interactive User Interaface

Popular, user friendly route planners offer user interfaces allowing the selection of start and end points on a 2D map. Offering this, and the outputting the generated route on that map will improve the accessibility of the project by working in familiar ways to popular technologies.

In addition, the user interface may allow interactive configuration of the cost function presets — enabling an individuals abilities and preferences to be entered easily.

These both work towards the goal of expanding the reach of off-road route planners to a wider general audience.

3.3.2 Route Generation Animation

Creating animations of the routing algorithms decision making may give insights into the routing algorithm's limitations and potential areas for improvement. This could show the decisions that are made and considered at each point on the routing process.

4 Software Development Approach

An AGILE development philosophy was used, utilizing exploratory testing within a SCRUM methodology. Exploratory tests were used towards the end of sprints, giving insights into the current state of the code-base. This enabled up-to-date representations to be maintained in the project backlog, stored using Microsoft Planner.

This philosophy was suited to this project as my initial expertise into the topics required was limited, and so re-prioritization was essential as it was initially more difficult estimating the amount of work required for many tasks. Staying AGILE also promoted consistent reflection, ensuring the projects aims and objectives were being met to my best ability.

Real-world data was used throughout development and testing, as opposed to creating mockup datasets. Mockup datasets allow for a more controlled environment within testing, making it easier to reason about the decisions made by the cost function. However, the testing with real-world data gives more applicable insights into how the router will function for end-users, potentially finding more unexpected bugs in the routing logic, due to the complexity of real-world route planning.

5 Ethics

This project has raised no ethical concerns. The self-assessment form is included in Appendix D.

6 Design

6.1 Search Graph Construction

The search graph is based on a TIN surface mesh constructed from raster DEM data due to the ease in which they can be converted into point-sets. The specific dataset chosen was the COP-30 dataset which has a resolution of roughly 30m, however, the graph construction is designed to be impartial to resolution or particular source.

The decision to construct the graph as a TIN was chosen due to its ability to accurately and efficiently represent geometric details, and the wealth of research into triangular mesh processing/simplification which are used here to reduce the number of search nodes to improve performance. Additionally, Delaunay triangulation can then be used to create the mesh, which doesn't require any alignment or consistency in the input point-set, enabling simplification to occur on the point-set itself, before triangulation. **??** includes a comparison in performance between these simplification methods.

Search nodes are initially created to represent TIN vertices, with two search edges per TIN edge, as edges represent the boundary between two faces, which both could be walked over

when following an edge.

The UTM coordinate scheme was chosen for the surface mesh. Global coordinate schemes are the only coordinate schemes avoiding the inherent projection distortion, but suffer from when creating meshes. UTM has a very low amount of distortion, and distances are equivalent to meters, automatically resolving issues with other projected coordinate schemes requiring translation and accounting for the curvature of the earth's surface.

A significant limitation arising from the selection of the UTM coordinate system is this project can only route between points in the UK — specifically UTM zone 30. With additional work, other zones may be supported, but issues arise when warping data converting multiple zones. However, many of feature datasets chosen only give data for the UK. Solving this issue is possible with additional work (see **??**). This work is beyond the scope of this project, as it is still possible to demonstrate the full routing ability of this project using U.K. example routes.

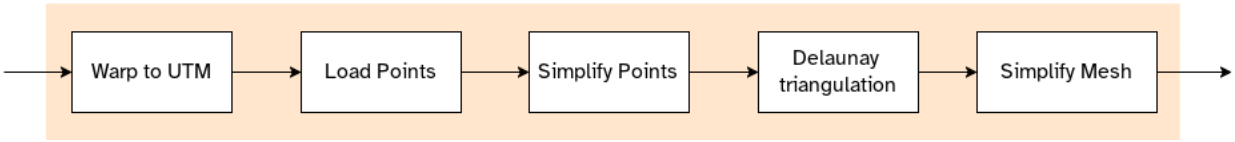


Figure 1: Flow graph showing the process required to generate a mesh from DEM data

Figure 1 outlines the pipeline from initial WGS84 DEM dataset to Delaunay Triangulation. The choice to simplify the point-set before triangulation, and again simplify the mesh after simplification was designed to allow for an exploration into performance differences and tweaks to the simplification, which are discussed later (see **??**.).

6.2 Data Feature Representation

Due to the limitations of W. Evans' approach of constructing the search graph solely using topography data, this project developed a novel approach to more accurately represent feature data within the search graph.

Two types of features were identified: area features, and edge features. When traversing over an edge, some features relate more to the surface (i.e. terrain area) being traversed; whereas others relate to the path itself (i.e. edge). For example, terrain type is best categorized as an area feature, as its volume over the surface is meaningful. Paths, however, are often best represented as edge features, as their width is often is irrelevant to travel on-foot.

The approach to representing these features accurately is similar but requires slightly different setup. Overal, the process aims to apply the geoemtry of both types of features as constraints in the delaunay triangulation.

For edge features, adding a constraint onto the mesh ensures that edge exists in the final mesh. For area features, we can ensure that feature tagging can be done accurately by applying the edges of data into the mesh as constraints. This ensures that the border of

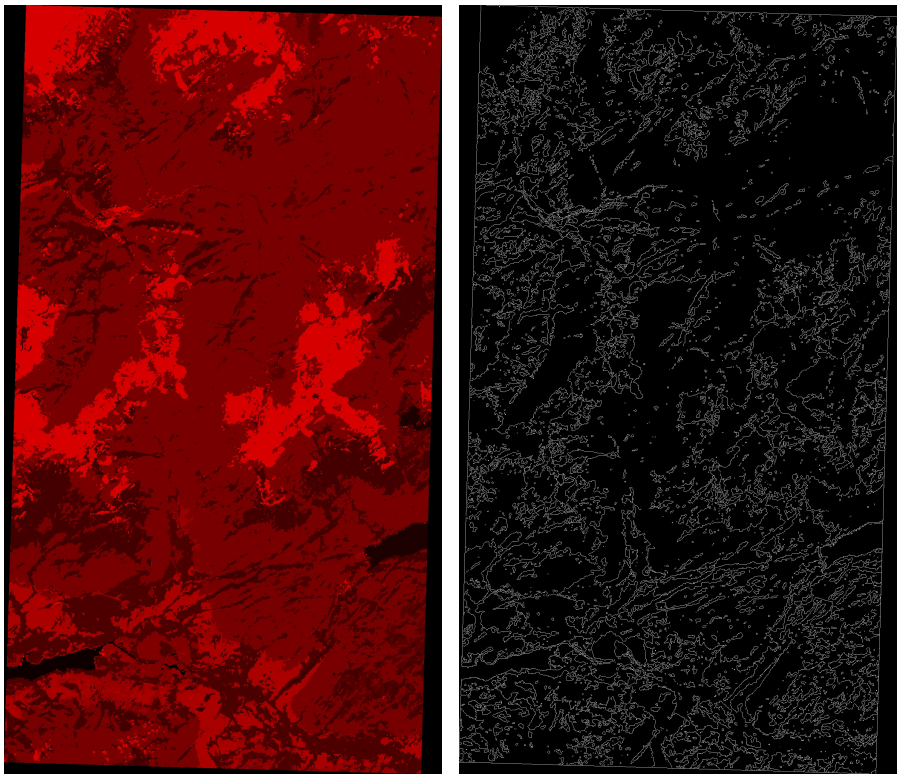
data are maintained with edges, allowing traversal avoiding that particular feature, and ensures no faces overlap the border, preventing issues where a face could be tagged with either data value depeneding on where in the face is used for tagging.

These constraints are added based on the X, Y coordinates of the contours, and are placed at the relevant Z coordinate on the original mesh to prevent unwanted changes to the topography.

This method works well for most features, as mapping data usually impact the whole area specified, and does not come with height data. Therefore, this approach is likely sufficient to model most useful data features (i.e. paths, terrain types, building positions, etc.). Some notable 3D features that cannot be represented however are: heights of bridges, which may be relevant due to the fear-of-height response; and the height of objects which may be climbable (although legal restrictions usually prevent climbing over objects captured in GIS datasets).

6.2.1 Vector Features

6.2.2 Raster Features



(a) CEH Terrain Data

(b) Detected Edges of the CEH Terrain Data

Figure 2: The edges detected from a chunk of terrain-type data warped to UTM.

Computer vision enables vector edges to be detected and extracted from raster data, which can then be converted into a set of constraints that can be applied onto the mesh.

Applying data edges as constraints is sufficient to accurately represent feature locations. Figure 2 shows the detected contours along with the original terrain type data. Every region defined by the contours have a uniform terrain type. Constraints ensure that edges fully cover each data border, and so every triangle will fall within, and not overlap any, terrain section — getting the pixel value for any point in these triangles will therefore give the correct terrain type.

Using constraints to define data borders enables a high accuracy dataset to be represented even on a low DEM resolution mesh.

6.3 Routing Algorithm

The design of the routing algorithm itself uses uniform cost search, with a custom modular cost function. Whilst A* would be more performant in many cases where there is an admissible heuristic, the admissibility of a heuristic is tied to the cost function, which here is designed to change and be customized with new features are relationships. Therefore, an admissible heuristic for one particular cost function would not be admissible for another. The ability to add heuristics could be relatively simply added, but given the wide target audience would need to be designed to protect against inadmissible heuristics.

6.3.1 Cost Function

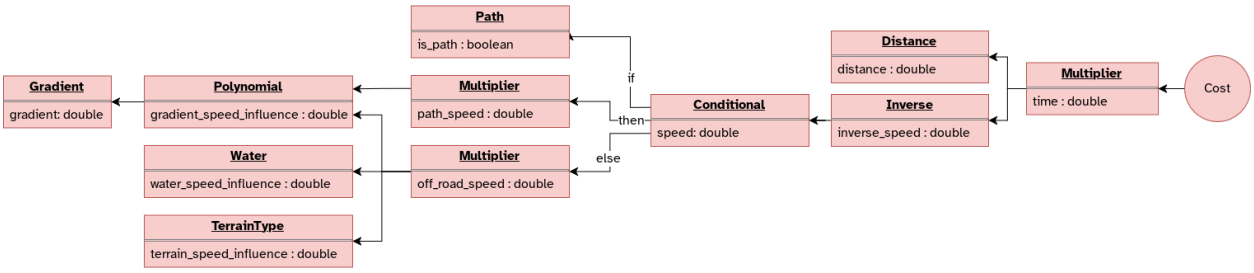


Figure 3: Example DAG cost function design. Output models an estimated travel time using three factors influencing travel speed.

A primary contribution this project has made is with the design of the cost function. The cost function is represented as a directed acyclic graph (DAG) of features, where each feature outputs a single value per cost-calculation, and connections between features allows the output of one feature to be used as an input to another feature forming dependencies.

Figure 3 demonstrates the capacity for the cost function to model mathematical functions. Here, the model estimates travel time following the calculation:

$$\text{Time} = \frac{\text{Distance}}{\text{Speed}} = \text{Distance} \times \frac{1}{\text{Speed}}$$

6.3.2 Features

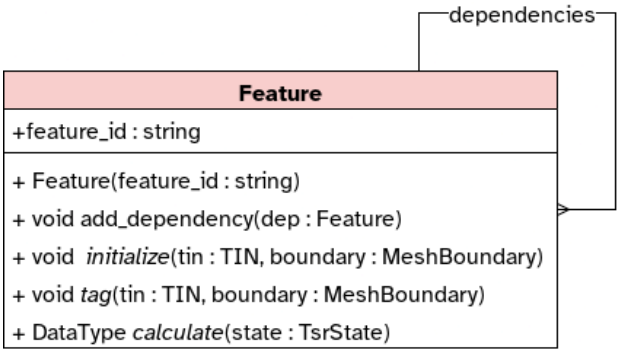
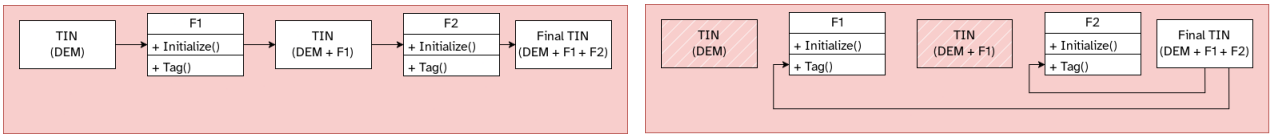


Figure 4: Feature class UMl design.

Features are designed to be able to represent any data or mathematical concept. Some features are categorical, numeric, or informational. Therefore, features have been designed to be able to output any data type. Figure 4 shows the design of a basic feature. Features are designed to use other feature's output as input, and so the output types must match expected dependency types. In addition, the final cost output feature must output a number value representing the cost. This allows complex data relationships to be modelled in a modular way, which makes re-using components in different configurations easier as the data can be outputted in it's original type, with different interpretor features used to model different factors related to that data.

6.3.3 Features Setup



(a) Initialization (b) Tagging

Figure 5: Initialization and tagging phases of feature setup. Showing how features interact with, and modify the TIN search mesh.

Each feature follows the same three step setup process: configuration allowing users to define option values; initialization of the feature, which may modify the search graph as described in subsection 6.2; and finally tagging.

Features by default set in the first stage just a unique identifier, which is used to check for dependecy cycles in the cost function. Features may add configuration options to this

construction. This allows user-preferences to be defined, benefiting the accessibility of the final product such as the degree to which paths are preferred over off-road terrain. These configuration options also enable data simplification and processing options to be modified, making it easier to test different processing options in testing.

The initialization and tagging stages of the setup process are outlined in Figure 5. The initialization stage is designed to allow for features to prepare data if required, and add contours to the TIN. The tagging stage is designed to enable caching of data results to improve performance during cost-calculations. For example, it is more performant to open a dataset once and tag each face with the terrain type, than to open and close the file each time a cost is evaluated. However, tagging would be invalidated if the TIN is modified, hence why tagging must be done separately to initialization, so that all features can add contours to the TIN before tagging takes place.

6.3.4 Data vs Logic Features

Whilst all feature types are treated the same, it is useful to distinguish data features, with logic features. Logic features take input, apply some logic, and output the value. Data features use datasets to give information about some feature of the terrain.

Every data feature will require a very similar design to their initialization and tagging setup functions.

For initialization, the example data features, and likely most data features, will fetch the required area of data for the given search area. Then, if required, contours will be extracted from that data feature and applied onto the mesh. Additionally, any caching of information that is not related to components of the mesh itself can be stored in initialization, as other features may manipulate the mesh further. Tagging enables mesh-specific data to be related in feature attributes.

Logic features on the other hand likely require little initialization or tagging. Examples such as gradient and distance are best calculated during routing, as the gradient and distance for a search edge is likely only useful for when that edge itself is having it's cost calculated. This reduces overal computation when uniform cost search skips nodes once the target node is found.

However, these distinctions are conceptual only, and features are welcome to put any logic in any of the three core setup and calculation logic, as long as the TIN is only manipulated in the initialization stage.

6.4 Graph Traversal

Dijkstra's algorithm was selected to serve as the basis of this search algorithm. Specifically, uniform cost search, as the ultimate goal is to find the optimal route between two nodes. Uniform cost has lower space complexity due to the storage of a best-cost only once the node has been searched, unlike Dijkstra's initial design which requires setting all nodes to

infinity best-cost initially.

6.5 API

Constructing both the initial topographical TIN and setting up data features require datasets from online APIs. Given a search, this application is designed to automatically fetch the GIS data required. These APIs have different, yet similar formats, and designing a simple interface that can be customized to work for any API format makes adding new features easier. Having automatic fetching of data from APIs is important for the user-experience and accessibility of the application, as otherwise each dataset would have to be manually downloaded and given to the application in some way.

6.6 Chunking

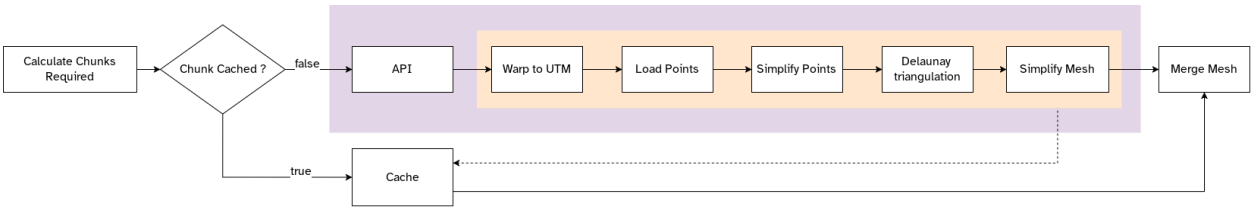


Figure 6: Final search graph construction process with automatic chunking, api calling, and caching.

For extensive searches, the system efficiently handles large volumes of Digital Elevation Model (DEM) and feature data by dividing it into manageable square tiles of a specified size. This method, known as chunking, ensures that data is loaded and processed safely while maintaining consistent partitioning across different uses of the application.

By automatically dividing land into uniform tiles, the system optimizes API calls and caching. This reduces the number of requests and processing required for subsequent searches, improving both speed and efficiency. Entire tiles are downloaded and processed, including those overlapping the search boundary, allowing complete caching of these tiles for future use.

This approach minimizes redundant data retrieval, accelerates operations, and ensures seamless handling of large datasets. It not only enhances performance but also reduces computational and network resource demands, making the system robust and scalable.

6.6.1 Parallelism

After the required chunks are determined, the process of fetching the data from the API and constructing the mesh shown in Figure 6 is designed to be completely independent

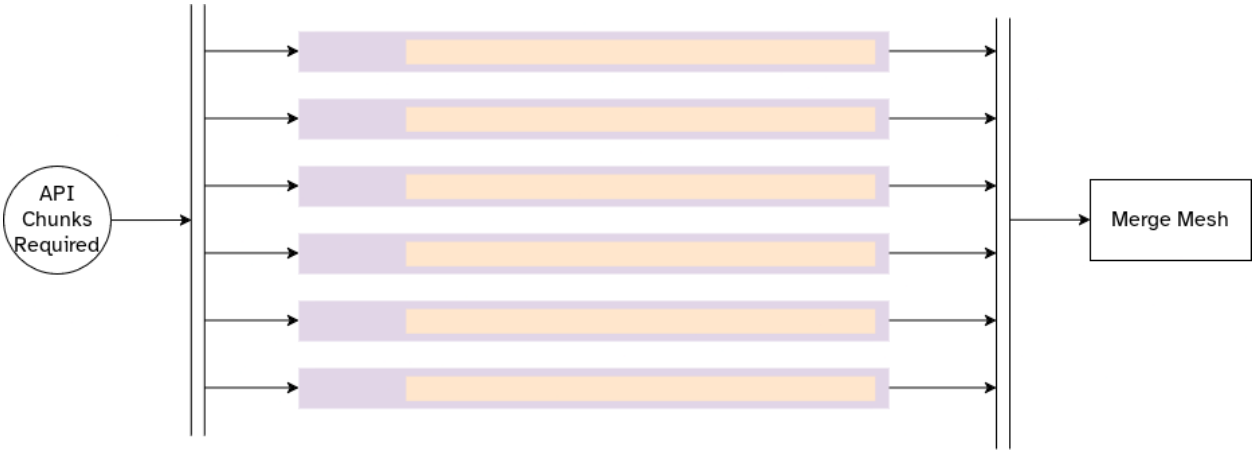


Figure 7: Parallel design of processing DEM chunks into a completed TIN.

for each chunk, and only once all chunks are processed are merged together. This enables a pipeline parallelism structure to be designed as shown in Figure 7.

6.6.2 Caching

Caching data is used to dramatically reduce the API calls and processing requirements required for subsequent searches with overlapping search boundaries.

7 Implementation

This section discusses some of the specific challenges encountered, and the solutions developed during the implementation of the design of this route planner.

An important note: UML diagrams are used to represent C++ classes. Pointers are shown with asterisks (e.g. `Feature *`). Where used, and unless otherwise specified, these refer to smart pointers. Smart pointers give far better memory safety, as objects are automatically freed from memory when out-of-scope, preventing the need to rememeber to manually free objects which easily leads to memory leaks if ommitted.

7.1 Output Visualization

The route planner generates KML files as output for use with Google Earth. A path is drawn between the source point and end point, following the recommended route given by the router. Colours are applied from green to red representing the gradient difficulty.

Where applicable, warnings are displayed telling users of potential hazards in the surrounding area, giving a brief description and estimated location. This gives insights into the reasons routes were chosen over others, and increases the safety, as users are aware

of specific features to avoid. To improve readability, only warnings for areas close to the generated route are displayed, which are likely the only warnings relevant when following the route.

7.2 Mesh Derivation

CGAL was used for converting the DEMs to TINs. CGAL offers constrained Delaunay trinagulation, which is an essential design feature of the mesh. In addition, to enable initial experimentation with mesh optimisation, CGAL does not impose a limit to the number of components in the mesh like the Fade library.

This 2D Delaunay triangulation projected onto 3D is less complex than using a 3D Delaunay triangulation, which constructs tetrahedra. Perkins, Marais, Gain, *et al*. [11] found that routing over tetrahedra is also less performant. In addition, the DEM data used is 2.5D, and so little benefit would be gained from a 3D triangulation.

7.3 Mesh Processing

The initial implementation triangulated the raw DEM data with no pre-processing to remove the number of points. Performance quickly became an issue for reasonably small search distances, due to the fact that 30m resolution DEM data includes 2916 points per square mile.

Two forms of simplification were implemented: pre-processing the DEM point set, and processing the constructed mesh itself. Both use CGAL's tools for this process. Both processing steps must be done before features are intialized, as they are likely to impact the accuracy of contours added. This two step processing enabled experimentation of different approaches, as research on CGALs implementations has not been conducted. Further experimentation would benefit the accuracy of the resulting mesh, as some finer details are affected by the processing implemented.

7.3.1 Point Set Pre-Processing

To simplify the point set, the `hierarchy_simplify_point_set` function is used - allowing the setting of the maximum size and maximum variation of clusters. This groups points into clusters of a given size, and removes vertices that are within the maximum variation bound set. In addition, the `wlop_simplify_and_regularize_point_set` tool was used, which supports parallelization out of the box, allowing confuguration of the maximum percentage of points that can be removed, and the closeness of similar points that can be flattened into a single point.

Extensive visual experimentation was done to find settings that maintained the visual accuracy of the topography. The final result reduces the overal mesh size by an average of 60%.

Pre-processing was found to have far greater performance gains due to the Delaunay triangulation that is required before mesh processing can occurr, in addition to the complexities of maintaining and simplifying a mesh data structure in comparison to a point set. However, certain processing steps were only available once triangulated into a mesh.

7.3.2 Mesh Processing

The mesh simplification tool used for this project was initially the `remesh_almost_planar_patches` utility. This allows sharp edges to be detected and protected, and almost planar (almost flat) regions to be identified. This then simplifies those planar regions. This supported parallelisation, but was found to still be a large performance hit. Additionally, issues with mesh integrity were found, and unresolved issues have been documented on their website. Further developments to this tool may be useful for this project, as this tool does not impact finer details as much as the point-processing method.

What is implemented in the final submission are simple removal of invalid elements, and the removal of duplicate vertices.

7.4 Features

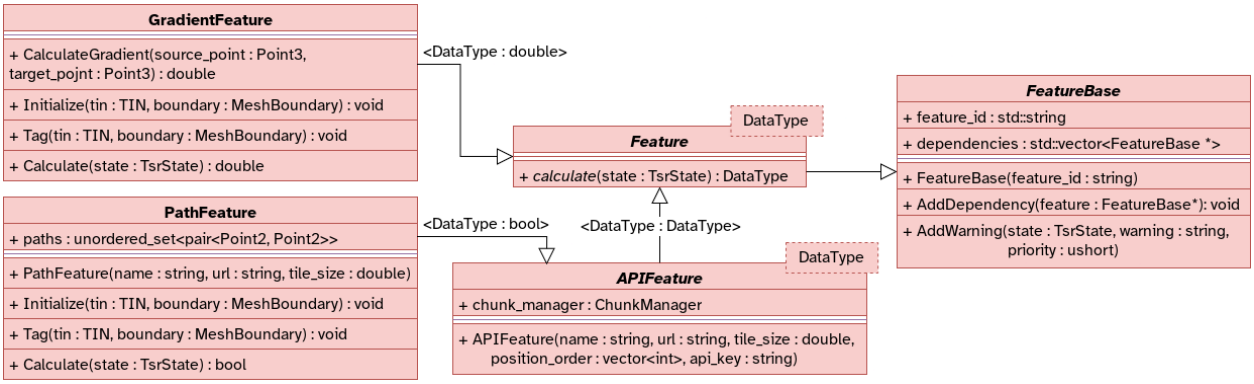


Figure 8: UML diagram showing the basic FeatureBase and Feature interfaces, with the specialized APIFeature interface, and two example of actual Features that can be instantiated.

7.5 Raster Feature Edge Detection

As mentioned in the design section, raster area features use computer vision to detect the edges and convert them to vector contours. The edge detection itself is handled by the OpenCV library, offering canny edge detection, which detects sharp changes in values (i.e.

data borders), which are then converted to a vector of vectors of points representing the contours which are just vectors of points.

7.6 Data Position Alignment

The position of data points within any GIS dataset is essential to accurately extract when relating the data to our mesh.

Aligning vector data is straightfowards, as each point is defined with a coordiante position in whatever coordinate system is used. As discussed, these datasets can be warped as points are extraced, or preprocessed as done in this implementation with GDAL - this was chosen to maintain a similar workflow to both vector and raster dataset processing.

Raster data required more configuration to align. As raster datasets are ultimately images, data is given in a grid of pixels, where each grid position just gives the value. Metadata in raster GIS datasets allows the coordinate of each pixel to be calculated with this calculation:

To calculate a pixels coordinate, we first take the lower left coordinate of the image stored in the `ADFGeoTransform` metadata, and then use the cell size metadata, and pixel x,y value, to calculate how far from that lower left coordinate the individual pixel is. The forumula for this is:

$$x = \text{upper left x} + \text{pixel width} \cdot \text{pixel x} + \text{row rotation} \cdot \text{pixel y} \quad (1)$$

$$y = \text{upper left y} + \text{column rotation} \cdot \text{pixel x} + \text{pixel height} \cdot \text{pixel y} \quad (2)$$

This formula is also useful for applying extracted contours to the TIN, as they are extracted with coordinates relative to the image dimensions and origin.

7.7 Coordinate System Conversion

Warping raster datasets requires a different approach than warping vector datasets. For raster data, GDAL's `GDALWarp` tool was used, which handles resampling and re-projection of datasets. For vector datasets, `GDALTranslate` was used to warp the vector points and geometries. These tools ensure efficient and accurate coordinate system conversions.

The decision to use GDAL instead of developing custom warping tools was driven by GDAL's proven optimization for performance and memory management. These factors are critical for the efficiency of the routing algorithm, as computational overhead can significantly impact results. GDAL's C++ APIs are well-documented and support a wide range of input file types and projections, streamlining the implementation process.

7.8 Uniform Cost Search

The `Router` class enables a particular search to be conducted. This implements the uniform cost search algorithm, using the defined cost function.

It begins by finding the closest node to the supplied source and target coordinates. An alternative approach would insert these points into the search graph as nodes in the Delaunay triangulation. However, this would need to be done before initialization. Since the point will likely point to a face in the mesh, routing from and to the nearest face is sufficient, as the start and points can be replaced with the supplied coordinates with difference to the routing correctness.

The router initializes a `TSRState` object, which contains a hashmap using c++ standard library's `unordered_map` linking nodes (TIN vertices), to RouteNode which contains information about the cost of the optimal route to get to that vertex. This is done by storing the optimal parent node, which is an adjacent vertex (i.e. one connected by an edge), that has the lowest cumulative cost to reach that point. A hashmap is used because of the $\mathcal{O}(1)$ time complexity for addition and retrieval, making it a perfect choice for large meshes.

A priority queue is constructed in the router which is added with RouteNodes when a node is searched. The queue is ordered based on the calculated cost, as described in subsection 2.6.

8 Results and Analysis

This section explores how different configurations and inputs affect the performance of the router in terms of both performance and quality of routes. Visualization of the generated meshes themselves is done through exporting the mesh to an object file, which is imported into the popular 3D modelling tool blender.

8.1 Final Configuration Mesh Generation

Objectively determining the quality of the mesh triangulation is not possible, as there are an infinite number of triangulations that could represent the same terrain. However, we can identify features in meshes which may give insights into their quality.

We initially evaluate the accuracy of the generated TIN mesh by visually comparing it to Google Earth's topography, as shown in Figure 9. Google Earth's horizontal accuracy (30m) and height accuracy (10m), are only marginally higher than the COP-30 DEM used in this project. Therefore, significant differences between the two meshes are likely a result of the mesh simplification process.

However, one limitation of this comparison arises from the differing projections used. Google Earth uses the WGS84 projection scheme, not UTM, and so there are some scaling and warping due to this difference. However, general features should still be identifiable.

(a) TSR's Solid Mesh (b) Google Earth's Topography

Figure 9: Visual comparison of meshes of Ben Nevis. The generated mesh from this application on the left, and Google Earth's topology mesh on the right.

As seen in Figure 9, the primary topographic features in the Google Earth mesh are all visible in the generated mesh. Nonetheless, improvements could be made in finer details such as sharp angles in mountain ridges, and sudden valleys.

In addition, the DEM data used fails to accurately model the topography of areas under water, as seen in the bottom left of Figure 9. However, this does not impact the quality of most routes as usually water is completely avoided, or otherwise the water is swam through, at the surface.

8.1.1 Edge Detection

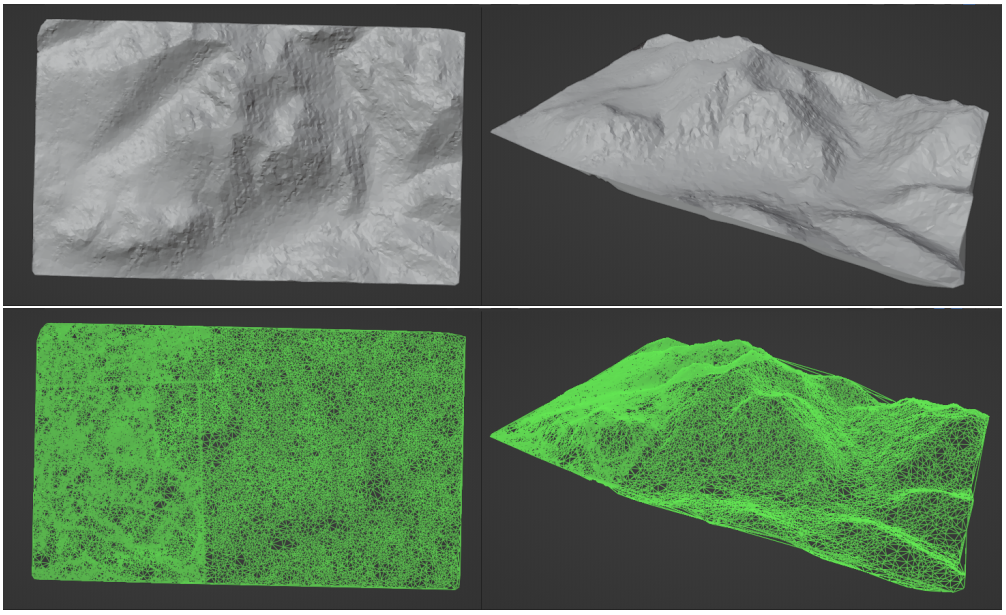


Figure 10: Final model of Ben Nevis, with path, water-feature, and terrain-type contours applied. A solid and wireframe render are shown, in both top-down orthographic, and side-on perspective views. Demonstrating a recognizable, but noisy and non-perfectly optimized mesh.

Figure 2 and Figure 10 demonstrate a drawback to the edge-detection implementation. For each chunk, a different image is used which is warped before fetching the contours. This means that the images have areas of NO_DATA values, and the border between NO_DATA and any value is detected as an edge. An alternative approach could be to run the edge-detection before warping, and using a vector warping on the resulting edges. It would be easier here to prevent edges on the border of images being included as contours in the final mesh.

8.2 Generated Routes

This section analyzes the routes generated by this route planner, considering differences between different cost function pre-sets. Evaluation therefore will be focused on the success to which the cost function generates routes according to the goal of the model.

The first preset can be found in Figure 3 and models estimated traversal time. Three features are considered

8.3 Performance

The route planner has multiple major factors influencing the performance of any particular execution which will be discussed and analyzed here.

Performance is analyzed in two parts: search graph construction, and then the actual routing. This enables a focused comparison of performance for different cost function models.

8.3.1 Mesh Size

8.3.2 Cost Function Complexity

8.3.3 Caching

9 Evaluation and Critical Appraisal

The results and analysis in the previous section enable an evaluation to be done on the project in reference to the aims and objectives set out in the initial aims and objectives requirements specification. This section will go through each requirement, and discuss the level to which the submission meets each of the given requirements.

9.1 Aims

The primary aim for this project was to contribute to the development of tools to make walking and running more accessible to those lacking safe roads, or to those who prefer not being restricted to roads and paths. The routing framework is successful in enabling users to define their own preferences in terms of the features through the configuration of features, and the cost function as a whole.

9.2 Primary Objectives

9.2.1 Modular Routing Algorithm

9.2.2 Route Visualization

9.2.3 Multiple Cost-Function Presets

9.3 Secondary Objectives

9.3.1 Critical Analysis of Presets

9.3.2 Additional Cost Function Features

9.4 Tertiary Objectives

9.4.1 Interactive User Interface

9.5 Route Generation Animation

10 Potential Improvements & Future Projects

This project aimed at forwarding research into accessible off-road route planning. Due to the scale of route planning off-road, and the limited previous research, this solution to the problem is not perfect. This section discusses these potential improvements in detail, along with a discussion of some of the extraneous research that would need to be done to solve some of the limitations of this project.

10.1 Parallel Route Search

10.2 Modelling Features

All of the data features constructed for this application were fundamentally first-attempts at quantification of some of the key features impacting routing preference.

This section outlines some improvements that could be made to the existing features, and possible additional features that would increase the modelling capacity of the route planner.

10.2.1 Distance Dependent Features

The implementation of the water feature created allows avoiding or heavily punishing swimming. However, it may be helpful to set a maximum swimming distance, to ensure routes through overly large areas of water are suggested.

This could be done by first getting the distance for the current edge if it is over water. If beyond a determined maximum swimming length, then it could return an infinite cost (implying non-traversable). If lower, then the source vertex's parent could be fetched from the `bestRoutes` map in the state. We can repeatedly check if that edge is over water, and add the distance to a running total until you reach land, to get the total current water distance.

Alternatively, a more performant approach may be to change the RouteNode class, and make it possible for Calculate methods to cache their calculated values to each Node, which could include a cumulative water distance, allowing future nodes to fetch this value instead of calculating for every cost-evaluation.

10.2.2 Cost Function Caching

Whilst there is pre-caching of many data-values, additional caching at-runtime for a particular cost-function evaluation may dramatically improve performance for more complex models.

The DAG structure enables multiple nodes to add dependencies to the same node as long as there are no cycles. However, each of these nodes calls the Calculate function separately.

One solution to this problem may be to cache values when a method is called first, and so subsequent calls do not require redundant calculation.

10.3 Iterative Cost Function

Due to new stack frames being generated at each function call, the current recursive design of the cost function limits its maximum size and complexity, as each dependency runs whilst the parents function is still in scope, thus using memory. Performance in all aspects could be improved through the conversion of this recursive design to an iterative approach.

This may involve using the FeatureManager to handle all calculate requests. Instead of features directly calling calculate to their dependencies and casting these results, the feature manager could maintain a list of the dependencies, and iteratively follow the DAG, not calling calculate until a leaf-node is found (one with no dependencies). The leaf nodes could then be calculated and their values passed as input when calling calculate on its dependent.

However, challenges arise in maintaining the ability for features to output any type, as the feature manager would have to be able to handle the arbitrary output type, and input it into features.

10.4 Improve Contour Extraction

The current implementation allows relevant data features to specify the level of simplification is applied to the contours extracted from datasets. This is suitable for binary data such as water which either exists or not. However, for categorical data, all borders are simplified to the same extent. In the case of datasets such as terrain data, it may be helpful to be able to specify the level of simplification for each data value - enabling stricter borders to be represented around hazardous terrain types such as bog.

10.5 Representating

The current implementation fails to properly represent streams. The WaterFeature data from openstreemaps gives the geometry of most water features. However, very thin and sometimes large water features (such as streams) can be represented as vectr lines with no geometry. This means the face-tagging approach used for water features does not accurately represent these water features.

A potential solution would be to seek alternative datasets specifically designed to represent water geometry. Alternatively, it may be possible to pre-process these water features, and thicken their line width. By enforcing a minimum line-size, we can ensure faces are created and tagged. However, this approach does not perfectly model the real-world, and so further options may be optimal.

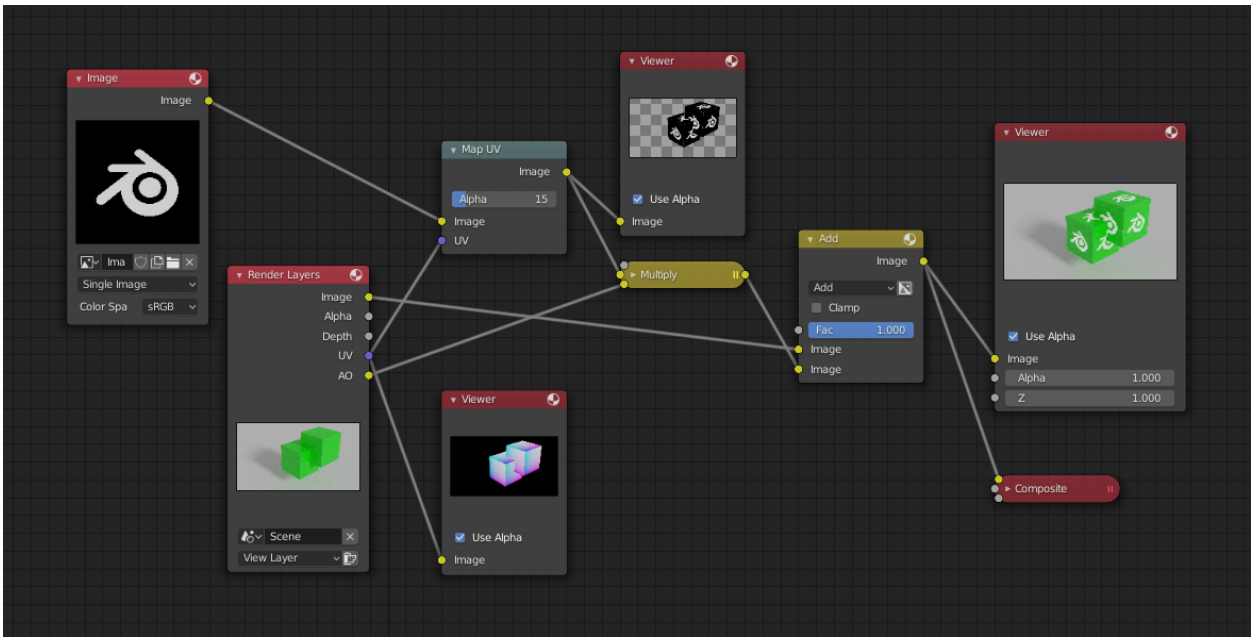


Figure 11: Blender's compositor interface, allowing interactive customisation of the rendering pipeline. Image taken from Blender's user manual [14].

10.6 Interactive Cost Function Configuration UI

A user interface enabling users to drag and drop features onto a user interface, and control the configuration options interactively would be a huge development towards this tool being useful for a general audience. Figure 11 shows Blender's compositor tool, allowing users to define their own DAG for how an object and scene gets rendered. This same interface could be applied to this routing algorithm, which features as nodes, connected defining dependency relationships, and options within the nodes relating to the configuation options. This interface could output either code to be used in preset setup, or it could output JSON, which could be parsed by a FeatureManager and setup.

11 Conclusion

To conclude, the success of this project hinges on its developments into the understanding of the possibilities and barriers to the creation of generalized off-road route planning. The resulting application has a broad capacity to model both mathematical concepts, and physical/psychological research into the impact certain terrain features such as terrain-type, gradient, and fear has on route preference. The potential for further developments and research present a promising opportunity to improve accessibility to walking and running as a form of transportation.

Whilst this project does overall add to the understanding of these topics, the end result is by no means a final product - being limited in both its accuracy, and performance. A large

amount of future research would be required to create a product that could be useful near the scale of traditional route planners.

This project could benefit immensely from a higher resolution DEM dataset, along with a comprehensive look into methods and configurations for optimization of the underlying mesh, and constraints, to both improve performance and accuracy.

12 References

[1] W. Evans, “Walk planner. a terrain-sensitive routing system for use in walk planning application,” 2023, [Unpublished master’s thesis].

[2] Google. “The keyword.” (Jul. 22, 2019), [Online]. Available: https://blog.google/products/maps/google-maps-101-how-we-map-world/ (visited on 01/01/2025).

[3] Department for Transport, *National travel survey 2021*, Aug. 31, 2022. [Online]. Available: https://www.gov.uk/government/statistics/national-travel-survey-2021/national-travel-survey-2021-mode-share-journey-lengths-and-public-transport-use (visited on 01/03/2025).

[4] K. Ek, L. Wårell, and L. Andersson, “Motives for walking and cycling when commuting–differences in local contexts and attitudes,” *European transport research review*, vol. 13, pp. 1–12, 2021.

[5] P. A. Singleton, “Walking (and cycling) to well-being: Modal and other determinants of subjective well-being during the commute,” *Travel behaviour and society*, vol. 16, pp. 249–261, 2019.

[6] D. Zhao, L. Ni, K. Zhou, *et al.*, “A study of the improved a* algorithm incorporating road factors for path planning in off-road emergency rescue scenarios,” *Sensors*, vol. 24, no. 17, p. 5643, 2024.

[7] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1 1959. DOI: 10.1007/BF01386390. [Online]. Available: https://doi.org/10.1007/BF01386390.

[8] A. Felner, “Position paper: Dijkstra’s algorithm versus uniform cost search or a case against dijkstra’s algorithm,” in *Proceedings of the International Symposium on Combinatorial Search*, vol. 2, 2011, pp. 47–51.

[9] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[10] S. Marschner and P. Shirley, *Fundamentals of computer graphics*. CRC press, 2018.

[11] S. Perkins, P. Marais, J. Gain, and M. Berman, “Field d* path-finding on weighted triangulated and tetrahedral meshes,” *Autonomous Agents and Multi-Agent Systems*, vol. 26, pp. 354–388, 3 2013. DOI: 10.1007/s10458-012-9195-8. [Online]. Available: https://doi.org/10.1007/s10458-012-9195-8.

[12] L. P. Chew, “Constrained delaunay triangulations,” in *Proceedings of the Third Annual Symposium on Computational Geometry*, ser. SCG ’87, Waterloo, Ontario, Canada: Association for Computing Machinery, 1987, pp. 215–222, ISBN: 0897912314. DOI: 10.1145/41958.41981. [Online]. Available: https://doi.org/10.1145/41958.41981.

[13] The CGAL Project, *CGAL User and Reference Manual*, 5.6.2. CGAL Editorial Board, 2024. [Online]. Available: https://doc.cgal.org/5.6.2/Manual/packages.html.

[14] Blender Foundation, *Compositor*, https://docs.blender.org/manual/en/latest/editors/compositor.html, Licensed under CC BY-SA 4.0, n.d.

[15] European Space Agency, *Copernicus global digital elevation model*, Accessed: 2025-01-01, OpenTopography, 2024. DOI: 10.5069/G9028PQB.

[16] E. Rouault, F. Warmerdam, K. Schwehr, *et al.*, *Gdal*, version v3.4.3, May 2022. DOI: 10.5281/zenodo.6517191. [Online]. Available: https://doi.org/10.5281/zenodo.6517191.

[17] OpenStreetMap Wiki, *Overpass api — openstreetmap wiki*, [Online; Accessed 2025-01-01], 2024. [Online]. Available: https://wiki.openstreetmap.org/w/index.php?title=Overpass_API.

[18] Google, *Google test: Google's c++ testing and mocking framework*, Version 1.14.0, 2023. [Online]. Available: https://github.com/google/googletest (visited on 01/02/2025).

[19] fmtlib, *Fmt: A modern formatting library*, Accessed: 2025-01-02, 2025. [Online]. Available: https://github.com/fmtlib/fmt.

[20] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[21] UXL Foundation, *Oneapi threading building blocks (onetbb)*, Version 12.3, Accessed: 2025-01-02, 2025. [Online]. Available: https://github.com/oneapi-src/oneTBB.

[22] HERE Technologies, *Logging.cpp*, https://github.com/heremaps/tin-terrain/blob/master/src/logging.cpp, Accessed: 2025-01-02, 2025.

[23] B. D. Team, *Boost c++ libraries*, Version 1.75.0, Accessed: 2025-01-08, 2021. [Online]. Available: https://www.boost.org.

[24] C. F. F. Karney, *Geographiclib*, https://github.com/geographiclib/geographiclib, [Online; Accessed 2025-01-01].

Appendix A System Requirements

The following is a list of required c++ libraries that need to be installed. The version used in testing is also given.

- GDAL
- CGAL
- oneTBB
- openCV
- GeographicLib
-

Appendix B Compilation Instructions

To build the library and router, first setup the build directory:

```
$ cd tsr-cli
$ mkdir build && cd build
```

Then configure with `cmake`, making sure to set configuration option values as appropriate:

```
$ cmake .. -G Ninja -DCMAKE_BUILD_TYPE= -DTSR_TEST=
```

- `TSR_TEST=ON/OFF`

  Specify whether to build test suite.

- `CMAKE_BUILD_TYPE=Release/Debug`

  Specify build type. Release is far more performant, but Debug contains much more logging information.

The library, cli-app, and tests can then be compiled using one of the following:

```
# Compile everything
$ ninja

# Compile just the library
$ ninja tsr
```

```
# Compile the tests
$ ninja test-tsr

# Compile the cli application
$ ninja tsr-cli
```

Appendix C User Manual

For the cli application, refer to the user manual below:

```
Usage:
    tsr-cli [options] <start_lat> <start_lng> <end_lat> <end_lng>
      Route between the given start and end points
    tsr-cli [options] --example
      Route between a set of example start and end points

  Options:
    --radii-multiplier  Radii multiplier to apply to domain size
    --output-dir        Directory to put output files
    --cache-dir         Tile cache directory
    --quiet             Reduce stdout output
```

For the web application, run the development server with the following:

```
$ cd tsr-ui
$ npm run dev
```

Appendix D Ethics Self-Assessment Form

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self-Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your ethical considerations.
Tick one box

☐ **Staff Project**
☐ **Postgraduate Project**
☒ **Undergraduate Project**

Title of project

Terrain Sensitive Routing

Name of researcher(s)

George Pestell

Name of supervisor (for student research)

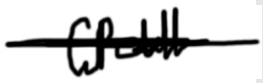
Graham Kirby

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self audit has been conducted **YES** X **NO** ☐

There are no ethical issues raised by this project

Signature Student or Researcher



Print Name

GEORGE PESTELL

Date

23/09/2024

Signature Lead Researcher or Supervisor

Print Name

Graham Kirby

Date

24/9/2024

This form must be date stamped and held in the files of the Lead Researcher or Supervisor. If fieldwork is required, a copy must also be lodged with appropriate Risk Assessment forms. The School Ethics Committee will be responsible for monitoring assessments.

Computer Science Preliminary Ethics Self-Assessment Form

Research with secondary datasets

Please check UTREC guidance on secondary datasets (https://www.st-andrews.ac.uk/research/integrity-ethics/humans/ethical-guidance/secondary-data/ and https://www.st-andrews.ac.uk/research/integrity-ethics/humans/ethical-guidance/confidentiality-data-protection/). Based on the guidance, does your project need ethics approval?

YES ☐ **NO** ☒

** If your research involves secondary datasets, please list them with links in DOER.*

Research with human subjects

Does your research involve collecting personal data on human subjects?

YES ☐ **NO** ☒

If YES, full ethics review required

Does your research involve human subjects or have potential adverse consequences for human welfare and wellbeing?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Will you be surveying, observing or interviewing human subjects?
Does your research have the potential to have a significant negative effect on people in the study area?

Potential physical or psychological harm, discomfort or stress

Are there any foreseeable risks to the researcher, or to any participants in this research?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Is there any potential that there could be physical harm for anyone involved in the research?
Is there any potential for psychological harm, discomfort or stress for anyone involved in the research?

Conflicts of interest

Do any conflicts of interest arise?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Might research objectivity be compromised by sponsorship?
Might any issues of intellectual property or roles in research be raised?

Funding

Is your research funded externally?

YES ☐ **NO** ☒

If YES, does the funder appear on the ‘currently automatically approved’ list on the

UTREC website?

YES ☐ **NO** ☐

If NO, you will need to submit a Funding Approval Application as per instructions on the UTREC website.

Research with animals

Does your research involve the use of living animals?

YES ☐ **NO** ☒

If YES, your proposal must be referred to the University's Animal Welfare and Ethics Committee (AWEC)

University Teaching and Research Ethics Committee (UTREC) pages
http://www.st-andrews.ac.uk/utrec/