A COLLABORATIVE WEB APPLICATION FRAMEWORK FOR LEAST COST CALORIC PATHS WITH CONSTRAINTS ON HIGH RESOLUTION DATA

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Audrey Waschura

June 2017

COMMITTEE MEMBERSHIP

TITLE:	A Collaborative Web Application Framework for Least Cost Caloric Paths with Constraints on High Resolution Data
AUTHOR:	Audrey Waschura
DATE SUBMITTED:	June 2017
COMMITTEE CHAIR:	Zoë Wood, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Aaron Keen, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Franz Kurfess, Ph.D. Professor of Computer Science

ABSTRACT

A Collaborative Web Application Framework for Least Cost Caloric Paths with Constraints on High Resolution Data

Audrey Waschura

Historically, humans traveling on foot do not travel as the crow flies in a constant direction; rather, they follow the path of least resistance. The path of least resistance could mean avoiding mountain ranges, or following a stretch of terrain or avoiding large bodies of water. These paths are taken because they are easier to traverse and allow the traveler to expend the least amount of energy, or calories. Least cost caloric paths are a more realistic model of how humans travel on foot across terrain.

Previous work has been done in computing and visualizing least cost caloric paths. This thesis presents a framework and implementation for a web application spatial data visualization tool, built in conjunction with expert input from an anthropologist, that strives to serve as an interactive tool for analyzing human travel across terrain. In addition to using least cost caloric paths across high resolution Digital Elevation Model (DEM) data, the system supports user-specified constraints on the travel. The collaborative web-based tool allows multiple users to view the same DEM model while specifying, computing, and visualizing multiple paths. In addition, the tool includes the ability for a user to specify constraint origin and distribution on the terrain to support specific study constraints such as no-travel zones (to account for hostile communities or commonly known latrine areas) and other sociopolitical boundaries affecting human travel across terrain. As the tool is intended for use via the web, usability and interactivity are key factors when considering the application design and implementation. To this end, we also present an error analysis of data resolution trade offs in computation time versus accuracy.

ACKNOWLEDGMENTS

Thanks to my family and friends for their endless encouragement and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Human travel across terrain, along with many other aspects of human behavior, is understood to follow the Principle of Least Effort [30, 27]. Humans are more likely to traverse areas that are easily accessible, thus limiting the cost of travel. This cost can be measured based on environmental, cultural, and physiological factors [27].

One such physiological factor is energy expenditure in kilocalories. This thesis continues previous work [28, 22, 24] computing and visualizing pedestrian travel along the path of least caloric cost. Such paths often avoid steep mountain ranges and large bodies of water and tend to follow flat stretches of terrain. This work also allows the integration of environmental and cultural factors to be modeled through user-specified constraints. Represented as lines and polygons over the landscape, these constraints can be used to simulate no travel zones such as hostile communities and commonly known latrine areas, as well as other sociopolitical or environmental boundaries.

This work presents a collaborative web application for computing, visualizing, and analyzing least cost caloric paths and user specified constraints. This tool was developed in conjunction with expert input from an anthropologist and strives to serve as an interactive and collaborative tool to be used by historians, anthropologists, and archaeologists in their study of pedestrian travel across terrain. It is written in Java using Google Web Toolkit (GWT) [14] and the graphics libraries Parallax3D [21] and Apache Commons Imaging [1]. The collaborative web application features are built upon the RpcLib library that manages server classes and communicates class properties between server and clients. Dijkstra's shortest path algorithm [3] is used to compute paths on Digital Elevation Model (DEM) data with five options for edge weight functions. Users may draw constraints in the form of polylines and polygons

and each constraint can have a binary, linear, or quadratic distribution to affect the path computations on the vertices affected by these constraints.

The high resolution DEM data used to compute accurate paths contains 100 million data points which can cause long path computation times and slow interactive speeds, a direct conflict with the aim of web applications to be highly interactive and responsive. Therefore, this work employs various data management techniques in order to allow users flexibility in the trade-off of path accuracy versus path computation time and interactivity by selecting which resolution data to use in path computations. An error analysis is presented which calculates the error in a path computation generated with low resolution data versus one generated with high resolution data. This analysis suggests that using low resolution data to rapidly compute paths with a slight error is a highly interactive alternative. Feedback is presented from novice users and archaeologists using the web application tool.

1.1 Contributions

The contributions of this thesis are listed below:

- Interactive web application tool to examine least cost caloric paths with user-specified constraints
- System design framework for caloric path computation and visualization in a web-based format
- Error analysis of data resolution trade-offs in computation time versus accuracy

Chapter 2

PREVIOUS WORK

2.1 Energetic Analyst

"Energetic Analyst" [28, 29], created by Brian Wood, exhibited the benefits of a human-centric metric for calculating the distance of paths across terrain, rather than distance as the crow flies. In this work, each terrain data point is a vertex with edges to its eight neighboring data points in order to form a graph. Caloric expenditure equations are used to determine the caloric cost for each edge, and Dijkstra's shortest path algorithm is used to find the path with the least caloric cost to traverse. Users are required to input start and end points in latitude and longitude coordinates. Furthermore, "Energetic Analyst" maintains all DEM data and data structures in memory, limiting a machine with 512MB of RAM to 650 x 650 data files at 90 meter resolution before performance is significantly degraded. This lack of interactivity, the requirement of Dijkstra's algorithm to traverse paths on edges, memory management, and relatively low data resolution are issues addressed in the subsequent works.

2.2 Continuous Energetically Optimal Paths

Jason Rickwald's "Continuous Energetically Optimal Paths" [22] builds upon "Energetic Analyst." Rather than Dijkstra's shortest path algorithm, Rickwald uses the Fast Marching Algorithm to determine the path of least caloric cost. This allows for paths to cross the faces created by the vertices of data points and their edges, which in turn creates a better path approximation than Dijkstra's due to the limitation of paths which only traverse edges. Rickwald demonstrated his tool with data at 30 meter resolution, thus further improving the accuracy of computed paths. Further-

more, the system allows for the selection of start and end points using mouse input on the terrain viewing application, an interactive upgrade from "Energetic Analyst." Rickwald also developed a multi-threaded version of the Fast Marching Algorithm which introduced a small amount of error in order run in one-third of the time of the standard Fast Marching Algorithm. This multi-threaded algorithm allows for larger, gigabyte-scale datasets due to its memory management technique, which examines blocks of the data at a time. Thus a fraction of the data is in memory while the rest is swapped to disk. However, large multiple-gigabyte data still had a day-long runtime due to the data format of the swapping.

2.3 Energetic Path Finding Across Massive Terrain Data

Andrew Tsui's "Energetic Path Finding Across Massive Terrain Data" [24] builds upon both previous works. His visualization tool adds interactivity via user selection of one of four path computation algorithms: Dijkstra's, an introduced multi-threaded version of Dijkstra's (Fast Dijkstras), A*, and Single-Query Single Direction PRM (Probabilistic Road Map algorithm). The system operates in a two-pass fashion with the first global search identifying an approximate path on a simplified dataset and the second detailed search using the approximate path in order to eliminate unnecessary search space, memory, and computation time. Tsui introduces Fast Dijkstra's algorithm which is a bidirectional multi-threaded algorithm which reduces runtime and maintains accuracy. The system is run on a large dataset of the contiguous United States which results in a large search space. To counteract this issue, Tsui introduces a restrictive tiling scheme to hierarchically divide the search space. This tiling allows for the first pass of the system to identify the applicable sections of the data that are included in the second pass of the search space. As a point of comparison, this system created a path for the Oregon Trail in less than an hour while it took most of

a day to create the same path with Rickwald's system.

2.4 Desktop Application versus Web Application

This thesis builds upon these three previous works on least cost caloric paths, each of which are standalone desktop applications. Conversely, this thesis aims to be a collaborative web-based tool which supports multiple users interacting with the system simultaneously.

Chapter 3

BACKGROUND

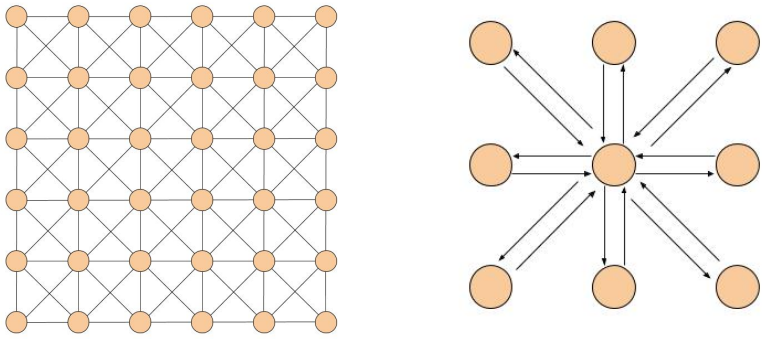
3.1 DEM Elevation Data

A Digital Elevation Model (DEM) is a 3D representation of the surface of some terrain. A .hdr header file indicates the extent of the region in latitude and longitude coordinates and a .img data file contains a grid of data values that each represent a sampled elevation (in meters). DEM data files are characterized by their resolution; 90m resolution DEM data has an sampled elevation data point every 90 meters. Many DEM data files are freely available online from various space shuttle programs. Common resolutions offered are 90m, 30m, and 10m (though usually for a fee). Northeastern Papua New Guinea was chosen as the region of study for this thesis due to ongoing research in this area. The highest resolution DEM data available in this region has 30m resolution. Using satellite images, we generated a 13.9m resolution DEM data file; this process is described in Chapter 4.

3.2 Dijkstra's Shortest Path Algorithm

This system uses a DEM dataset that considers each elevation data point as a vertex with edges to its eight neighboring elevation data points, see Figure 3.1a. Thus between each pair of neighboring vertices, there are two edges, one in each direction, see Figure 3.1b. The calculation of each edge's weight is described in the next section.

This graph and its weighted edges are used in the calculation of the least caloric cost path. The system uses Dijkstra's single-source single-destination shortest path algorithm, a well-known algorithm for determining the minimum path between two



(a) DEM data as grid (b) Eight neighbors

Figure 3.1: DEM data to graph: (a) DEM elevation data points are interpreted as vertices connected in a graph (b) The center vertex and its 16 total connections to its eight neighbors

vertices on a graph [3]. Using a minimum priority queue, the shortest path from the source to each visited vertex is determined until the destination is reached. The computational complexity of Dijkstra's shortest path algorithm with a priority queue is $O\Big((|E| + |V|)log|V|\Big)$. The grid formed by the DEM data contains eight outgoing edges for each vertex, thus $|E| = 8|V| + c$ where c is a constant to account for vertices on the edges of the file which do not have all eight outgoing edges, so we can say the algorithm's time complexity is $O(|V|log|V|)$.

3.3 Caloric Cost

This system provides five options for algorithms to calculate edge weights, detailed in Table 3.1.

[1]This algorithm is to be used for comparison purposes only. For a more accurate estimate of caloric cost see Wood & Wood.

Table 3.1: The five options for algorithms to calculate edge weights

2D	Each edge weight is the 2D distance in meters, this is equivalent to the "as the crow flies" distance
Weighted 3D	Each edge weight is the 3D distance in meters, with more cost associated with uphill travel
Hudson Hikers	Each edge weight is the result of a hiking caloric expenditure model found on the Hudson Hikers blog [17][1]
Hiking Science	Each edge weight is the result of a hiking caloric expenditure model found on the Hiking Science blog [16][1]
Wood & Wood	Each edge weight is the result of a caloric expenditure model described below which is used in "Energetic Analyst" [28, 29], "Continuous Energetically Optimal Paths" [22], and "Energetic Path Finding Across Massive Terrain Data" [24, 25]

3.3.1 Wood & Wood Caloric Cost Calculation

The model used by previous work [28, 29, 22, 24, 25] is implemented in this work as Wood & Wood. This model is based on energy expenditure models developed through metabolic rates with and without load carriage [20, 4, 5, 15, 19]. The model is based on the pedestrian's age, weight, height, and load carried as well as the distance traveled, the speed, and the slope and type of terrain. Equation 3.1 is the metabolic rate calculation for level or increasing slopes while Equation 3.2 is the metabolic rate calculation for downhill slopes.

$$MR_{uphill} = M \tag{3.1}$$

$$MR_{downhill} = M - C \tag{3.2}$$

Where:

$$M = 1.5w + 2.0(w + l)\left(\frac{l}{w}\right)^2 + n(w + l)(1.5v^2 + 0.35vg) \tag{3.3}$$

$$C = n\left(\frac{g(w + l)v}{3.5} - \frac{(w + l)(g + 6)^2}{w} + 25 - v^2\right) \tag{3.4}$$

And:

$$g = 100 * \frac{elevation_b - elevation_a}{dist_{2D}} \tag{3.5}$$

MR	Metabolic Rate (Watts)
w	Pedestrian's Weight (Kilograms)
h	Pedestrian's Height (cm)
a	Pedestrian's Age (years)
l	Load Carried (Kilograms)
v	Velocity (Meters per Second)
g	Slope of Grade
$elevation_a$	Elevation of **a** (Meters)
$elevation_b$	Elevation of **b** (Meters)
$dist_{2D}$	2D Distance from **a** to **b**
n	Terrain Factor

Terrain factors [19] are defined as:

1.0	Treadmill/Paved road
1.1	Dirt Road
1.2	Light Brush
1.5	Heavy Brush
1.8	Swampy Bog
2.1	Loose Sand

This metabolic rate is calculated for each edge in the path computation. However, the calculated downhill metabolic rate can under-predict the caloric expenditure, therefore the standing metabolic rate (SMR) is compared to the calculated rate in Equation 3.6.

$$MR_{downhill} = max\{MR_{downhill_{3.2}}, SMR_T\} \tag{3.6}$$

Note that BMR and SMR refer to daily values so we must normalize them for the time duration (in seconds) of each edge.

$$SMR_T = SMR\left(\frac{1 \text{ day}}{24 \text{ hours}}\right)\left(\frac{1 \text{ hour}}{60 \text{ minutes}}\right)\left(\frac{1 \text{ minute}}{60 \text{ seconds}}\right)T \tag{3.7}$$

$$T = \frac{dist_{3D}}{v} \tag{3.8}$$

$$SMR = 1.2 * BMR \tag{3.9}$$

$$BMR_{male} = 66 + (13.7w) + (5h) - (6.8a) \tag{3.10}$$

$$BMR_{female} = 655 + (9.6w) + (1.7h) - (4.7a) \tag{3.11}$$

SMR_T	Standing Metabolic Rate Adjusted for Edge Duration (Watts)
SMR	Standing Metabolic Rate (Watts per Day)
BMR	Basal Metabolic Rate (Watts per Day)
T	Time to travel the edge from **a** to **b** (Seconds)
$dist_{3D}$	3D Distance from **a** to **b** (Meters)
a	Starting location of edge (Latitude/Longitude)
b	Ending location of edge (Latitude/Longitude)

Ultimately, the edge weight is converted into kilocalories:

$$edge_{\mathbf{ab}} = \frac{MR * T}{4184 \frac{Joules}{kilocalorie}} \tag{3.12}$$

Each of the following parameters can be set by the user in order to customize the edge weight caloric cost function to a certain research project or study: weight, height, age, load, speed, and terrain factor.

3.4 Existing Methodologies in Web-Based 3D Data Visualization

Web-based 3D visualization and interaction is a vast area of research and development with many applications in a wide variety of fields including medicine, education, manufacturing, entertainment, and transportation [26]. Web-based systems are attractive for their portability, collaborative potential, and the capability to leverage cloud computing and existing available data sources [9]. This work aims for compatibility through the ability to export paths in the .gpx file format to then be imported

into Google Earth and other terrain viewing applications. In addition, 3D visualization tools have migrated from desktop applications to web applications as the success of OpenGL is being applied to WebGL, a rendering context for the HTML Canvas element. This is demonstrated in the fact that WebGL is now a feature of web browsers. There are various wrappers for WebGL (three.js and Unity Game Engine are popular options). This thesis uses the GWT 3D graphics library Parallax3D [21], for all 3D data visualization. This library does, however, have the limitation that meshes in Parallax3D have a maximum vertex count of 22K vertices. As a result, the system sub-samples the terrain data to not exceed this limit, see Section 4.2.1.

There are many existing web-based mapping applications for geographic spatial data visualization. Google Maps is a web mapping service developed by Google to view and share maps and additional data including traffic, street maps, satellite imagery, and street view [12]. ArcGIS Online is a cloud-based mapping platform created by Environmental Systems Research Institute (Esri) for creating, analyzing, and sharing maps on the web [2]. Google Earth for Chrome is a web application version of the classic desktop application used for visualizing 3D terrain data [11].

3.5 Client/Server System Design

This thesis follows a client/server system design, common to many web-based applications. One server maintains the system state and serves requests made by multiple clients in this distributed system architecture. The system state is a collection of classes identified by unique names. Each class contains properties, which are each identified by a name key. Each client that accesses the web application is added to a list of clients. When a client clicks a button or interacts with the web application in order to affect some state change, the changed property is sent to the server. The server then updates the state and marks that the state has changed for specific clients.

Each client polls the server for state changes once per second and, upon new changes, retrieves the new state to update itself. When a client retrieves this new state, the server removes the 'has changed' marker for that client. The collaborative nature of the web application system is such that each change is reflected in the server and then each of the clients.

3.6 GWT

Google Web Toolkit (GWT) is an open source development toolkit for creating web applications. GWT allows developers to create AJAX applications. The GWT toolkit compiles Java servlets that run on web servers and simultaneously compiles client side code into JavaScript to run on all browsers, including mobile. Communication between server and clients is done through remote procedure calls (RPCs), which allow the transfer of Java objects back and forth. GWT supplies many pre-existing user interface widgets and abstracts away most usage of CSS and DOM manipulation. Furthermore, the toolkit provides easy external library integration and debugging tools, and is well-documented and well-supported. The design decision to use GWT in this thesis was based on the ability to leverage the convenience and ease of use of the GWT toolkit including the use of Java and its high-level language constructs.

3.7 RpcLib

The RpcLib library facilitates the definition of methods and properties to maintain at the server level and the communication of these methods and properties to clients through remote procedure calls (RPCs). The RpcLib library maintains a list of clients and properties each client has in order to determine which properties must be updated for which clients. Each client polls the server once per second to identify any properties that are outdated so the client can then retrieve the updated information,

updating the server's list of properties on that client. This helps to enable the collaborative nature of the web application, so that all clients share the information maintained at the server.

Chapter 4

SYSTEM DESIGN

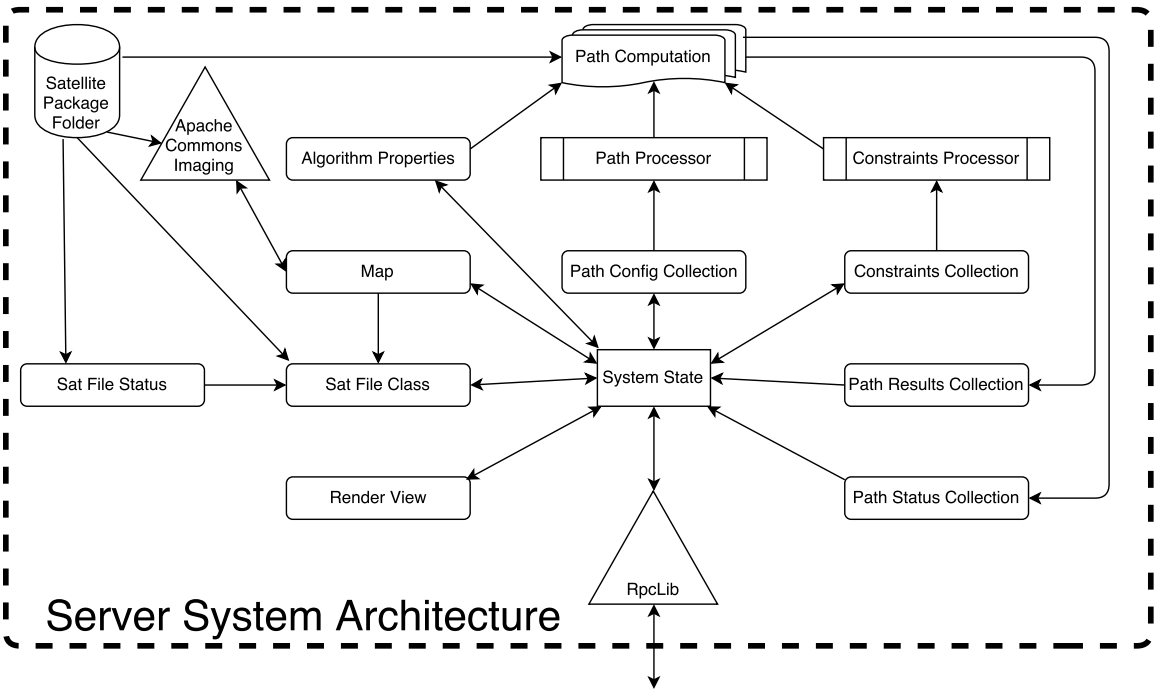


Figure 4.1: Server System Architecture, arrows indicate flow of information

This thesis presents a collaborative web application to compute, visualize, and analyze least caloric cost paths across high resolution DEM terrain data. Figure 4.3 shows the UI. Users are presented with a 2D map view and 3D scene to visualize the terrain data. An Information button shows a dialog listing the basic functions of the application. Users may select one of three data resolutions to use for path calculations. The 2D map view may be navigated with zoom and pan functions. Users can select a rectangle in the 2D map view to identify the render region of interest which is then used to specify the terrain to render in 3D. The 3D scene can be manipulated with orbit, pan, and zoom functions. In a paths menu panel, users can add, edit, delete, and export paths. Paths are displayed in the 2D map and the 3D terrain view. In a

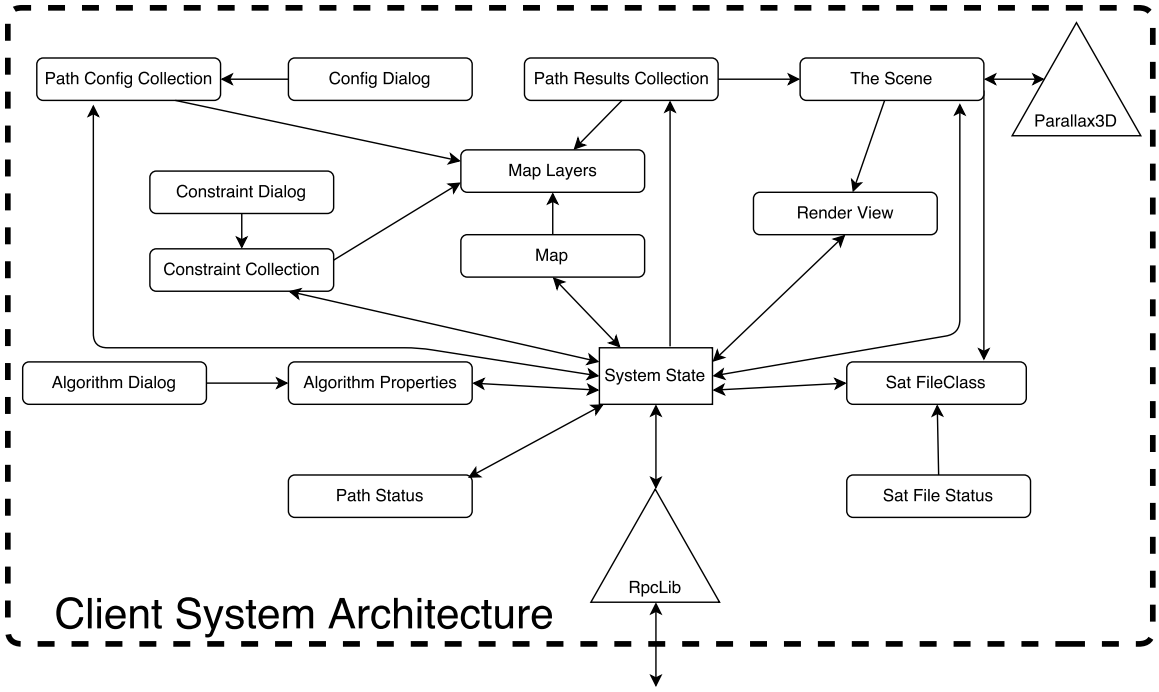


Figure 4.2: Client System Architecture, arrows indicate flow of information

constraints menu panel, users can edit, delete, or draw constraints in the form of lines and polygons. These constraints appear in the 2D map and affect path computations to simulate sociopolitical or geographic boundaries on the terrain which limit travel in these regions.

This system was created using the Apache Commons Imaging library [1] for image processing and Parallax3D library [21] for 3D rendering. The tool is written in Java using GWT [14] and the RpcLib library to manage the web application remote procedure calls (RPCs). The web application uses a Tomcat server running on a Virtual Machine simulating a 2GHz processor with one core, 20GB of disk space, and 8GB of memory.

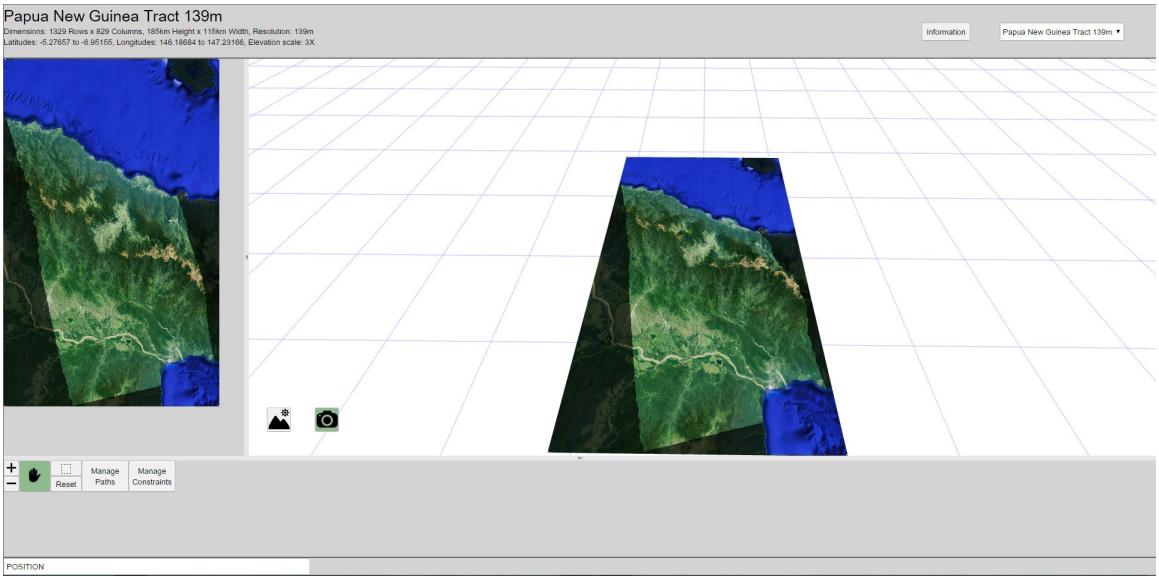


Figure 4.3: User Interface

4.1 DEM Data Acquisition

Due to the desire for high resolution DEM data and the lack of such a caliber of resolution available in the test region of Papua New Guinea, we created a DEM file from satellite images. First two satellite images were downloaded from the Copernicus Open Access Hub [6], a hub for accessing satellite data acquired during numerous Sentinel satellite imaging missions from the European Space Agency. The two images used were collected during the Sentinel-1 imaging radar mission which collected continuous all-weather, day-and-night imagery [8] using Synthetic Aperture Radar (SAR) instruments. The two images used in DEM creation are in Interferometric Wide (IW) mode and packaged as a Processed Level 1 Single Look Complex (SLC) data product. This SLC data contains complex imagery in both amplitude and phase bands [8].

The Sentinel Application Platform (SNAP) [7] software toolbox was used to perform the process of converting two overlapping satellite images into a DEM data file [10]. This process requires two SLC images and a low-resolution DEM (given that

this process creates a high resolution file) of the desired region. The steps required for DEM creation are outlined in Figure 4.4. All DEM creation processes occur in SNAP, with the exception of the Phase Unwrapping of the Filtered Phase which occurs in Snaphu, a software package for Statistical-Cost Network-Flow Algorithm for Phase Unwrapping [23].

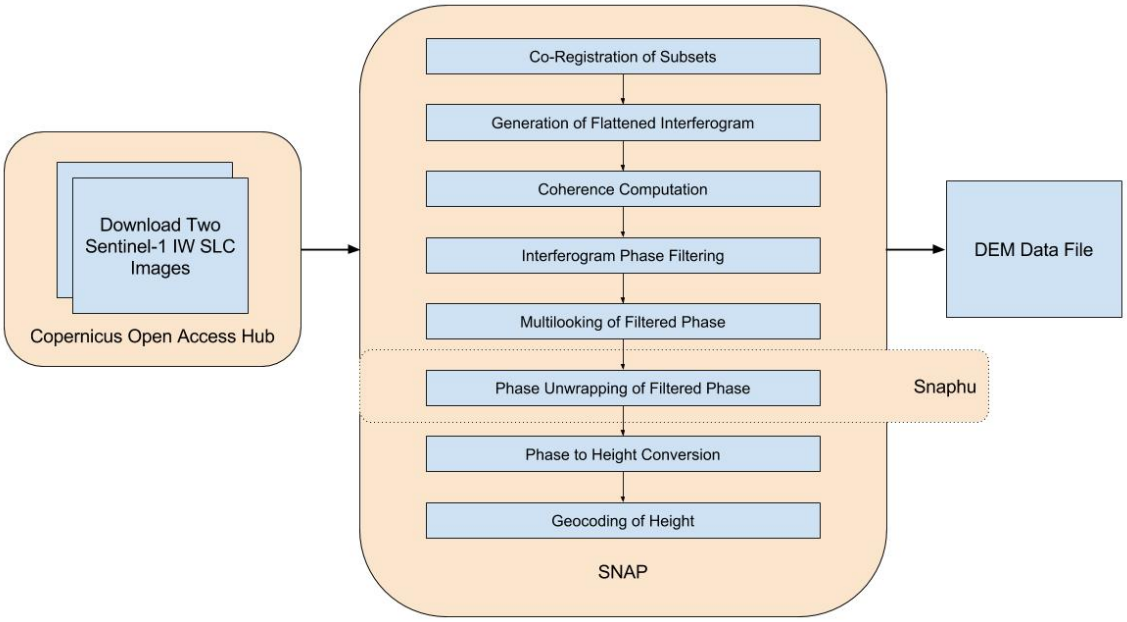


Figure 4.4: The DEM creation process

The resulting DEM has a 13.9 meter resolution and is 13,282 rows by 8,282 columns. As the vertex count of the DEM file is directly related to the running time of path computation algorithm, two additional DEM files were created from the original in order to achieve faster computations (at the price of some error, see Chapter 5). Files decimated by 10- and 5- times in rows and columns create two additional DEM files at 139m and 69.5m resolution respectively, as shown in Table 4.1. Each DEM data set contains a .hdr header file and a .img data file. These two files in combination with a satellite view image of the region comprise a data package. This third satellite image is required for the 2D map view as well as texture map on the 3D terrain. A full package is required to install and display a specified region.

Table 4.1: Three DEM data files at differing resolutions

Resolution (meters)	Rows	Columns	Total Vertices (millions)
13.9	13,282	8,282	110
69.5	2,655	1,656	4.4
139	1,329	829	1.1

The resulting DEM is stored in a rectilinear grid along the latitude and longitude lines of the Earth, while the original satellite data is a swath in an arbitrary orientation. Thus the DEM data rectangle contains areas of invalid data: missing data values and water data values, indicated by the dim shading and the blue color respectively, in the satellite image (Figure 4.5) used in the 2D map and the texture map on the 3D terrain.

4.2 DEM Data Management

4.2.1 3D Terrain Resolution

In order to minimize the data transferred from server to client and avoid the vertex limit imposed by Parallax3D [21], a subset of the installed DEM data is displayed as 3D terrain. First the rendered region of interest is identified. If it is smaller than the full region, this area is delineated by a white rectangle in the 2D map view as seen in Figure 4.6. Then the full resolution data of the region of interest is sub-sampled in order to create an array of elevation data which is sent to the clients to be displayed in the 3D scene. Due to the vertex limit in Parallax3D, we limit the total 3D terrain data vertex count and calculate the dimensions of this sub-sampled 3D terrain data based on this maximum. Once the dimensions have been calculated and the elevation data collected, this data is sent to the clients to be displayed. The 3D terrain is

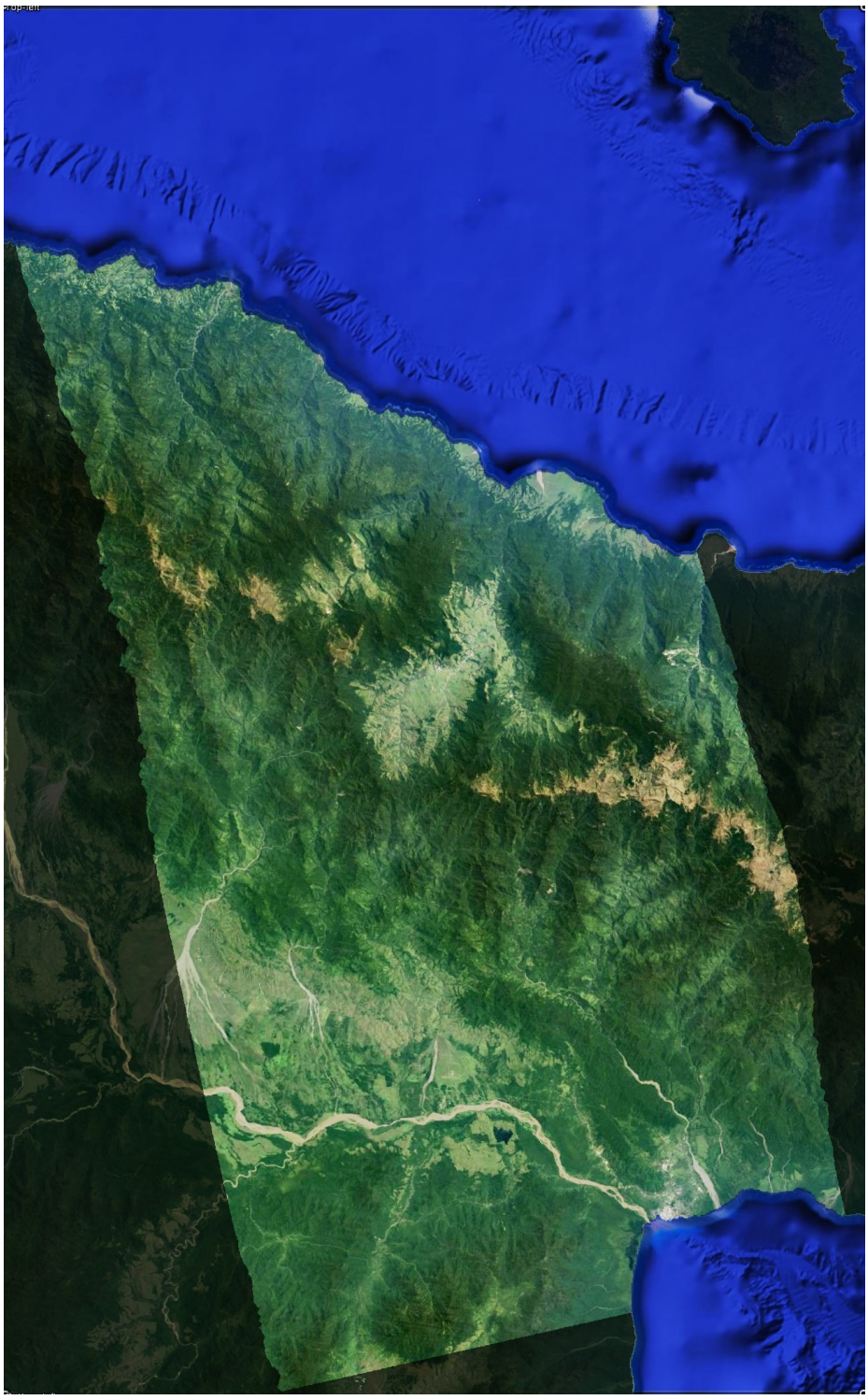


Figure 4.5: The satellite image used in the 2D Map and as the texture map for the 3D terrain mesh

displayed using a plane mesh element. The plane is composed of rows and columns in x and y to match the dimensions of the sub-sampled terrain data. Each elevation data point is used to set the z value for a vertex on the plane mesh. This turns the plane mesh from a flat plane into a region of terrain with varying heights. This sub-sampling of DEM data to display in the 3D scene is a good approximation of the terrain, as can be seen by the registration of the 3D terrain and the texture mapped 2D satellite image of the region, specifically in the visual aspects of the mountain peaks in Figure 4.7. In order to make the terrain appear more interesting and make changes in elevation more drastic, the elevation is displayed at a scale three times the actual elevation scale. This elevation scale exaggeration factor is displayed to users at the top of the web application.

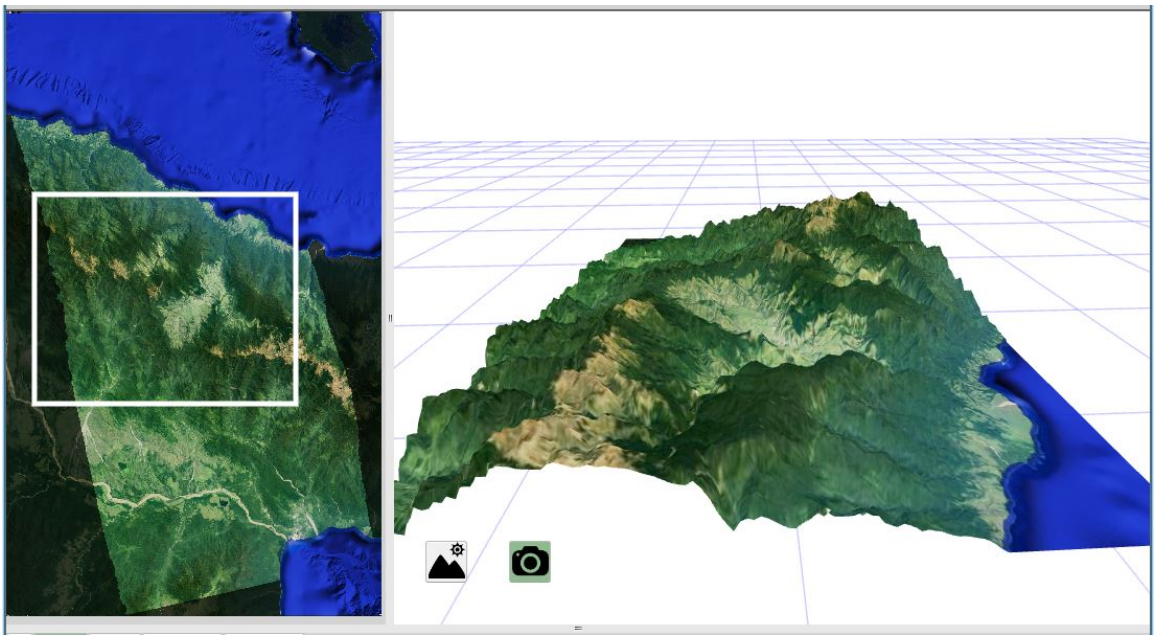


Figure 4.6: The white rectangle on the left 2D map sets the region to be rendered in 3D, viewed here from the east looking west

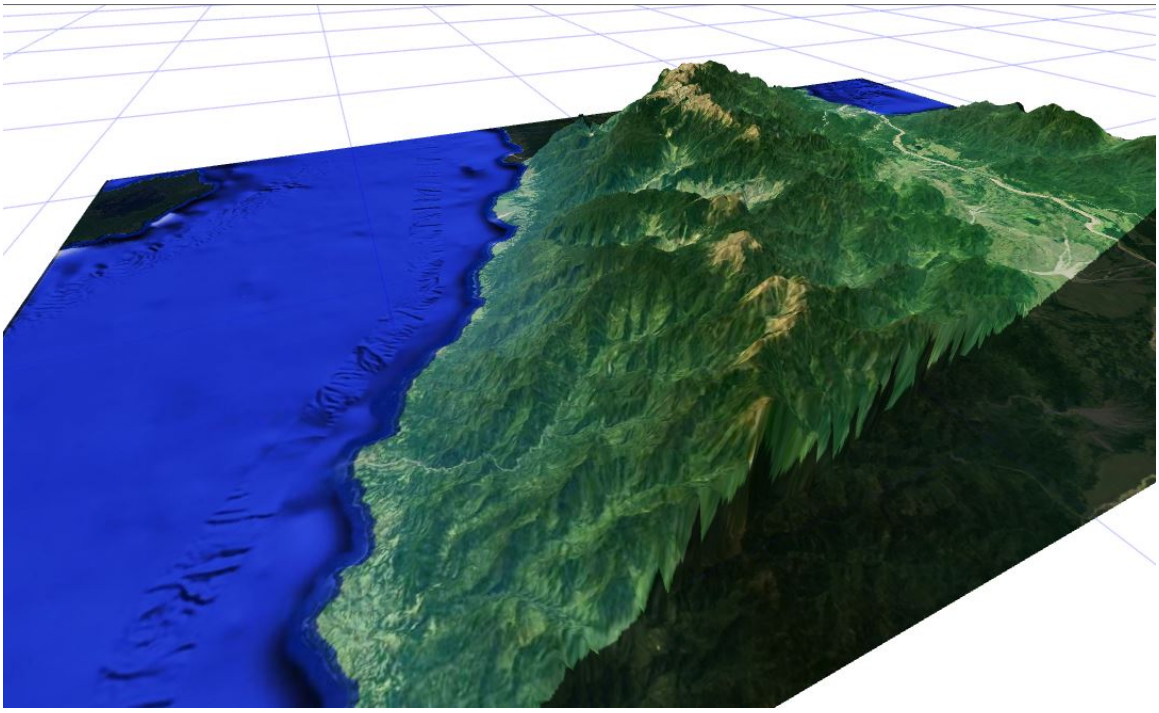


Figure 4.7: 3D terrain data registration with the texture mapped 2D satellite image

4.2.2 Path Resolution

At any point in time, there is one installed DEM file: either the 13.9m, 69.5m, or 139m resolution file package. In order to create the 3D terrain data, this installed DEM will be sub-sampled to not exceed the vertex limit in the 3D scene. Path computations, on the other hand, are done on the complete installed DEM. This is in order to preserve accuracy when exporting the path for use in other applications.

Since the paths are computed at the resolution of the installed DEM file, and the 3D terrain is sub-sampled to be at a lower resolution, when the paths are displayed in the 3D scene, they do not align with the terrain data. To remedy this display misalignment, a second pass over the path data is required. After the path is computed, each path point is a tuple of latitude, longitude, and elevation. During this second pass, the elevation of each path point is altered to match the level of the 3D terrain elevation at that latitude/longitude location. To achieve this, the location in

the 3D terrain data mesh must be identified where the path point exists. The exact mesh triangle that the path point lies on must be identified. This triangle mesh is then interpreted as a plane and a vertical line is formed through the path point and through the path point projected into the x-y plane. This triangle plane and line are intersected to identify the point where the path point lies directly on the triangle plane. For each path point, an adjusted elevation is calculated which indicates where the path point lies directly on the 3D terrain data at that point. It is this adjusted elevation data that is used to display the paths in the 3D scene so that the misalignment is eliminated and paths lie directly on the 3D terrain.

4.3 Path Computation

4.3.1 System Level

In the interest of making multiple path computations quickly, each path computation occurs on its own thread. The installed DEM file is accessed using a shared memory mapped file. This allows the installed DEM to be accessed through virtual address space in the same way dynamic memory is accessed. Thus the installed DEM file is concurrently shared by each of the path computation threads. During computation, a data structure is used to keep track of each vertex's previous vertex. For very long paths on high resolution data, this data structure becomes very large. In the worst case it maintains a previous vertex entry for every vertex in the file. Rather than create and maintain a data structure in memory, a random access file is used to contain this data. The data is indexed by vertex number, thus adding and retrieving data is $O(1)$. Once the endpoint has been found and the building phase of the algorithm is complete, this file is used during the back traversal to find the path from the endpoint to the start point.

4.3.2 Status

Table 4.2: Statuses posted for a path during its computation

Status Message	**Condition**
New	When a new path is created by a user
Start	When the data structures are being initialized for the algorithm
H% (XK hh:mm:ss)	During computation (H is the heuristic percent complete, X is the thousands of vertices traversed, and hh:mm:ss are the hours, minutes, and seconds of the computation time)
Publishing Results	When the algorithm is back traversing from the end point through each vertex's previous vertex to determine the path
Done 100% (XK hh:mm:ss)	When the algorithm has finished and the server publishes the path results (X is the thousands of vertices traversed and hh:mm:ss are the hours, minutes, and seconds of the computation time)
Empty	If start and end points are equal or invalid
Error	If an error has occurred during computation

Especially on the high resolution DEM file, path computations can take many hours to complete. In order to ensure users that paths are indeed computing and the system is still responsive, each path has a status associated with it, see Table 4.2.

One rudimentary way to calculate a percent complete heuristic for a path calculation is to take $\frac{|\text{vertices traversed}|}{|\text{total vertices}|}$. However, this is a gross underestimate due to the regions of invalid data within the DEM file. These invalid data points are not in-

cluded in the path computation and are skipped entirely, so using the total number of vertices (including those which are invalid) is a poor choice. The heuristic used to determine the percent completion of the path computation is a ratio of the current position relative to the start and end points:

$$ratio = \frac{dist_{\text{start to current}}}{dist_{\text{start to current}} + dist_{\text{current to end}}} \tag{4.1}$$

In order to avoid a rapid increase at the start followed by a slow increase towards the end of the computation, an exponential function is used to "slow down" the heuristic:

$$ratio_{slow} = \frac{100^{ratio} - 1}{99} \tag{4.2}$$

During the computation, the algorithm will traverse out from the starting point, so in order to make the heuristic monotonically increasing we keep track of the maximum aforementioned ratio and use this maximal value to display the heuristic percent complete.

4.3.3 Export Path

Once a path computation has completed, it may be exported from the system. A user may select a path in the path list, click the "Export Selected" button, and the .gpx file of the path data will download in the browser. GPS Exchange Format (GPX) is an XML schema for coordinate data. It can store waypoints, tracks, and routes and is used by many GPS and spatial data applications [13]. Tracks are used to describe where a person has been while routes are suggestions of where they might travel in the future. Therefore, this thesis exports a path as a route to suggest where a pedestrian might travel. This exported route is composed of routepoints, each representing a

path data point as a latitude, longitude, elevation tuple.

4.4 System State

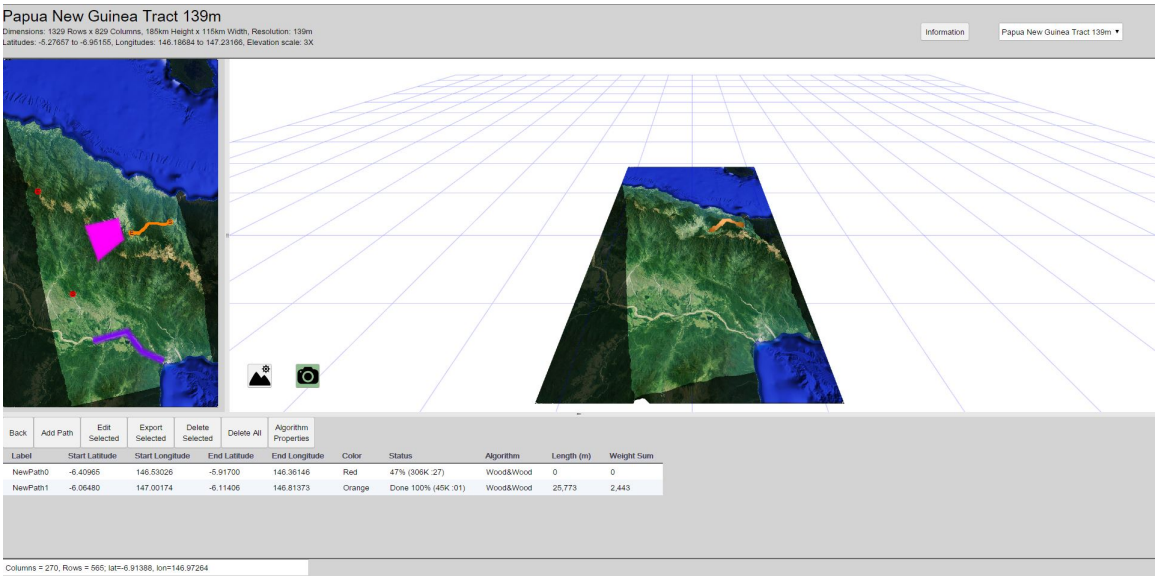


Figure 4.8: Paths panel of UI

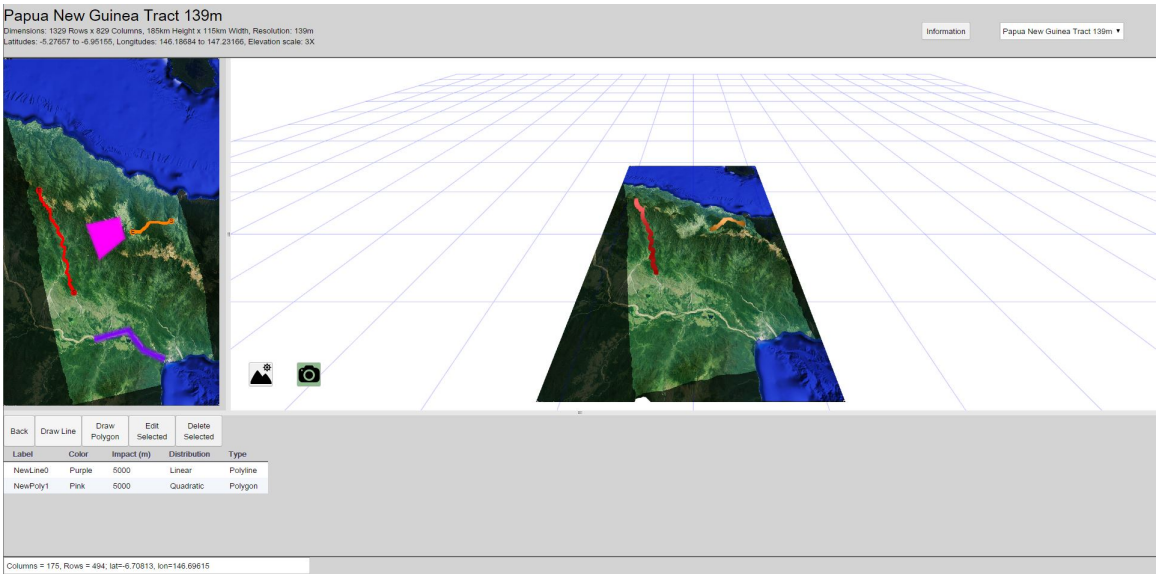


Figure 4.9: Constraints panel of UI

In order to implement the client/server model described in Chapter 3 and outlined

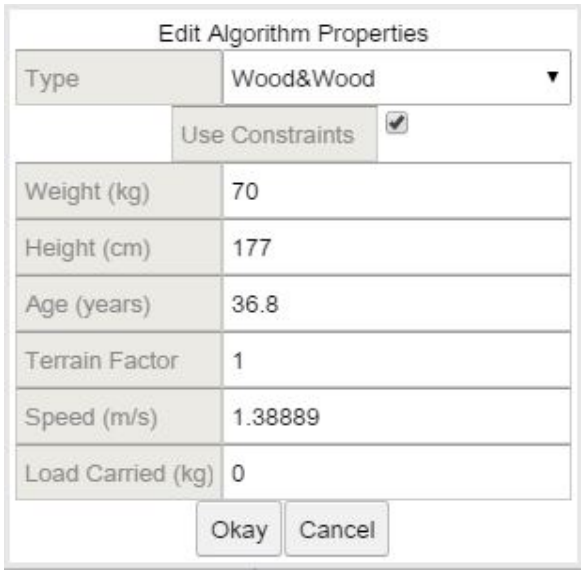


Figure 4.10: Algorithm Dialog, for editing algorithm properties

in Figures 4.1 & 4.2, the following is a list of objects sent between client and server to maintain the system state:

AlgorithmProperties Properties of the path computation algorithm including weight, height, age, load, speed, and terrain factor, see Figure 4.10

ConstraintCollectionClass List of Constraints (each has properties for type (line or polygon), distribution (binary, linear, or quadratic), name, a unique id, impact distance (in meters), color, and data points stored as latitude/longitude pairs), see Figure 4.9

MapClass Properties of the 2D satellite image including the original size of the installed image, the view area of the 2D map view (affected by zooming and panning), the render region of interest (demarcated by the white rectangle in the 2D map), the mode indicating interaction with the 2D map (pan, draw render region rectangle, draw constraint line, draw constraint polygon), the sequence number to track when new 2D map images are created and must be updated

Figure 4.11: 2D Map view with zoom and pan

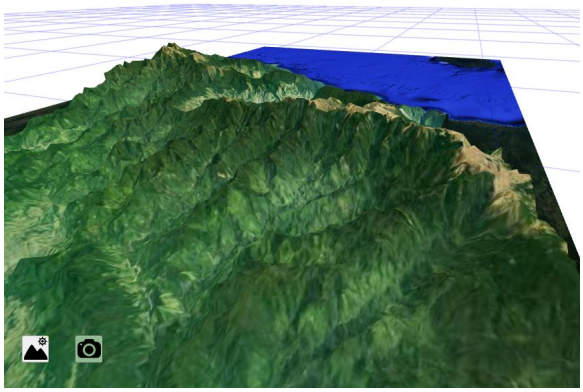


Figure 4.12: Navigation of 3D scene with mode buttons in bottom left

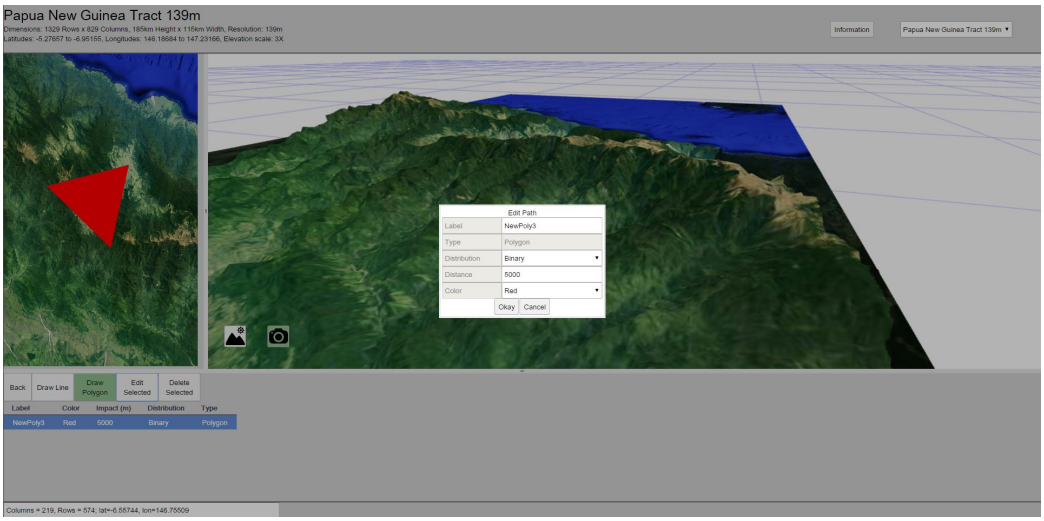


Figure 4.13: Constraint dialog to edit a constraint once drawn

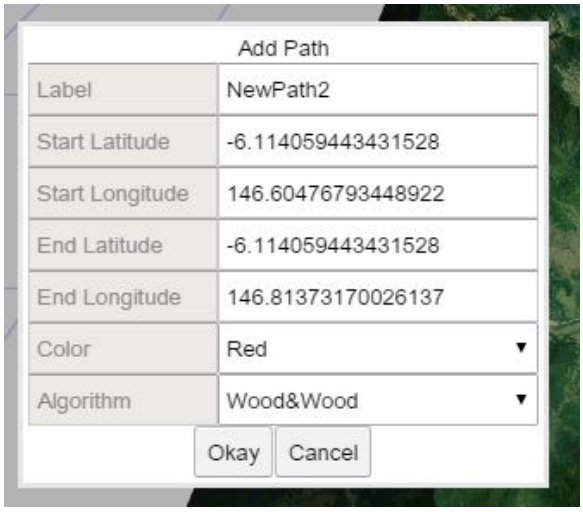


Figure 4.14: Add path dialog

in the client, see Figure 4.11 for a 2D map view which has been zoomed in an panned

PathConfigCollectionClass List of PathConfigurations (each has properties for name, unique id, start point as latitude/longitude pair, end point as latitude/-longitude pair, color, which algorithm used for edge weight calculations), see Figures 4.8 and 4.14

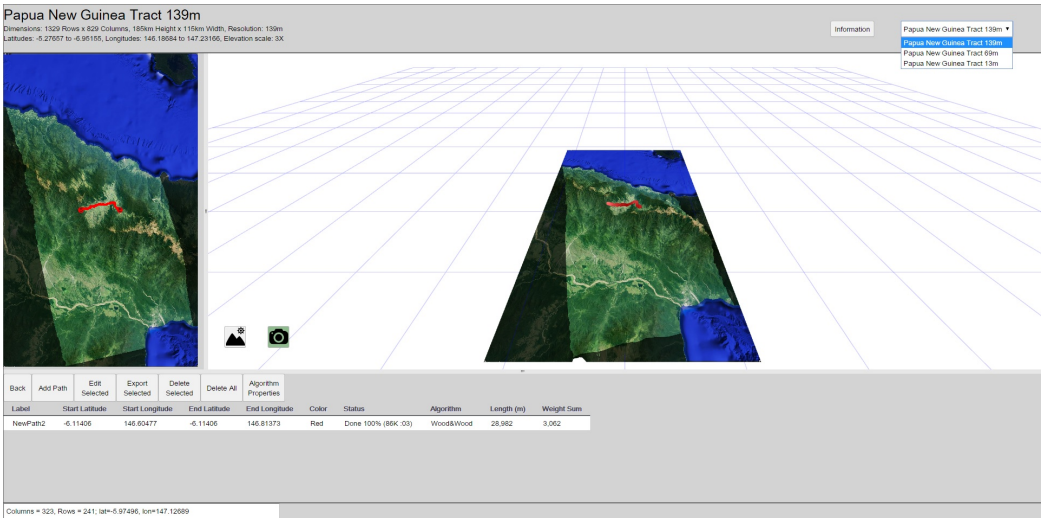


Figure 4.15: File resolution drop-down menu selector

Figure 4.16: File information in top left of screen

PathResultCollectionClass List of PathResults (each has properties for unique id, color, total length (in meters), sum of edge weights, full resolution path data points (stored as latitude/longitude/elevation tuples), adjusted path data (stored as latitude/longitude/elevation tuples)), see Figure 4.8

PathStatusCollectionClass List of PathStatuses (each has properties for unique id and status), see Figure 4.8

RenderView Properties of the 3D scene view including the eye position, look at position, angles θ and ϕ for rotation, the mode indicating navigation with the 3D scene (orbit and pan), see Figure 4.12

SatFileClass Properties of DEM file including list of available files to install, index of which file in the list is currently installed, file name of currently installed,

SatFileStatus (see below), region of interest (center in latitude/longitude, width in degrees, height in degrees), dimensions of data used in 3D scene, array of actual data used in 3D scene, see Figure 4.15

SatFileStatus Properties of installed DEM file including rows, columns, center in latitude/longitude, width in degrees, height in degrees, see Figure 4.16

The client/server model and system state concept is demonstrated in Figure 4.17. This figure shows the sequence of events that occur once a user adds a new path to the system.

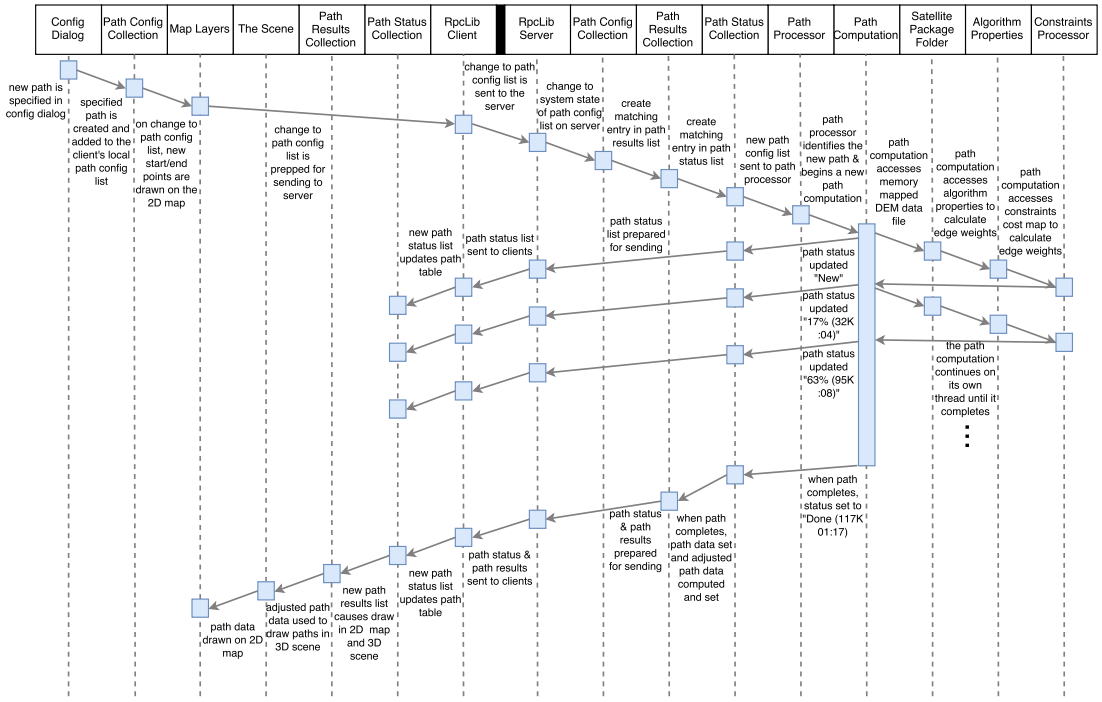


Figure 4.17: Sequence Diagram for creating a new path, arrows indicate time progression

4.5 Constraint Calculations

Constraints are supported by the system in order to allow the integration of sociopolitical and geographic boundaries in the study of human travel across terrain, see the

Algorithm 1: Build Constraints Map

Data: list of constraints

Result: Generated map of (index, cost) pairs representing the indices affected by constraint costs and their associated cost values

```
1 foreach constraint c do
2     if c.type = line then
3         ProcessLineConstraint(c);                    // see Algorithm 2
4     else                                             // c.type = polygon
5         ProcessPolygonConstraint(c);                 // see Algorithm 3
```

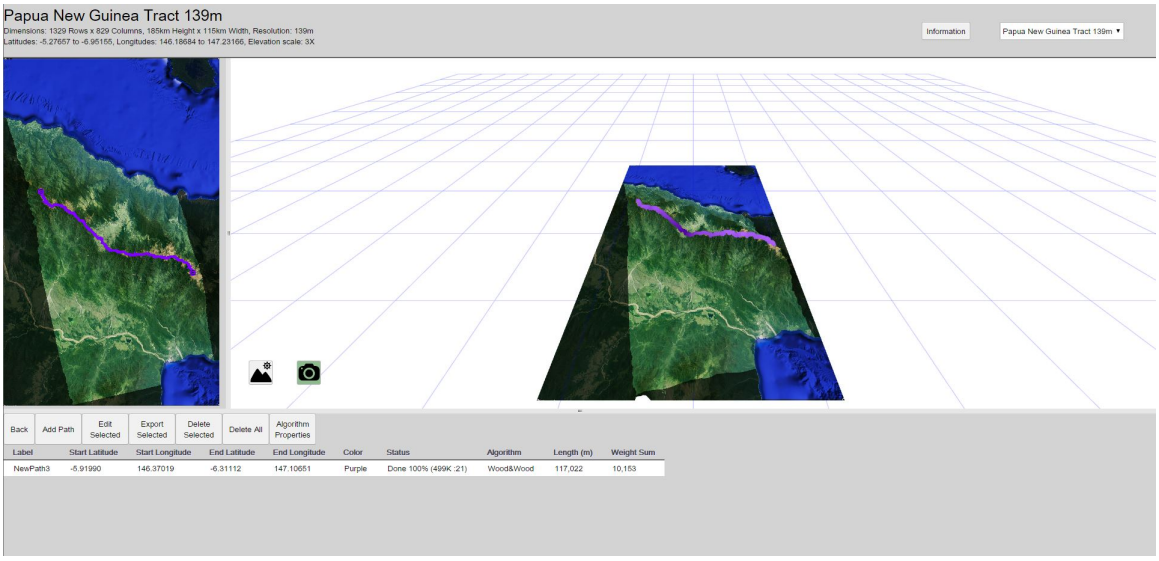


Figure 4.18: Path along mountain range without constraint

same path without and with a constraint in Figures 4.18 & 4.19. In order to have user-drawn constraints affect the path computations, the system maintains a map of affected DEM data vertices. This map is rebuilt each time the list of constraints is changed and each time the file resolution changes. The map is created by Algorithm 1. Constraints are either lines or polygons, have a user-specified impact distance to indicate how far the distribution affects the terrain, and can have one of three distri-

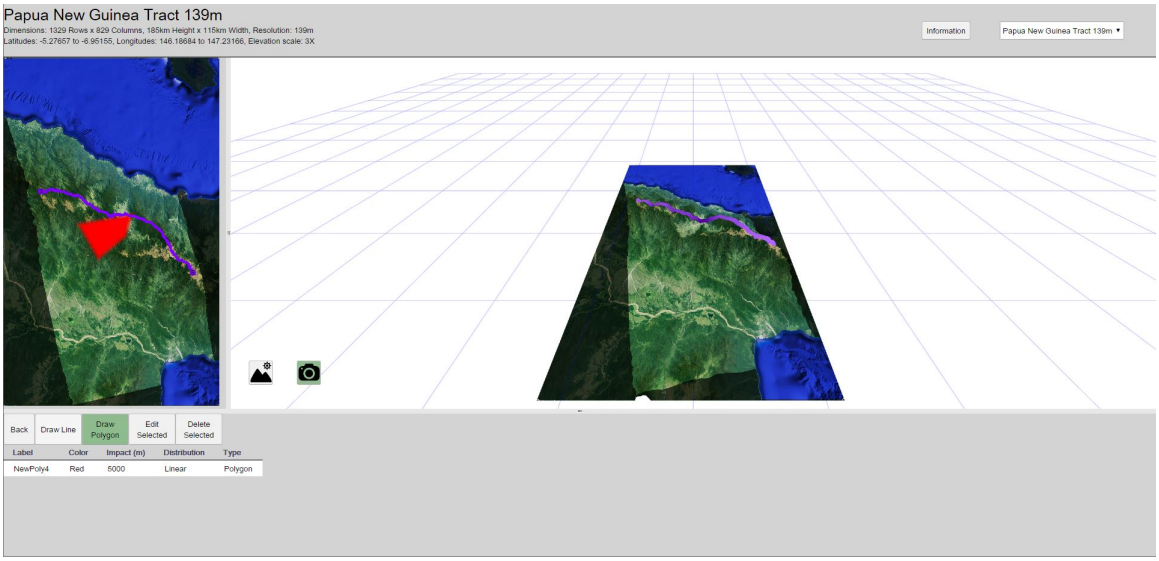
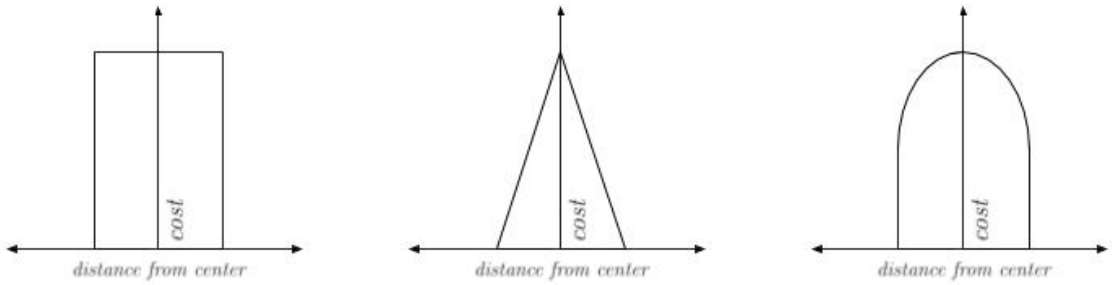


Figure 4.19: Path along mountain range with constraint

bution types: binary, linear, and quadratic. Figures 4.20a, 4.20b, and 4.20c show a section slice of the three distribution types, with height indicating constraint cost and width indicating distance from the center, where the full width is the impact distance. Line constraints are drawn as polylines: a sequence of connected line segments, see Figure 4.21a.



(a) (b) (c)

Figure 4.20: Constraint Distribution Types: (a) Binary (b) Linear (c) Quadratic

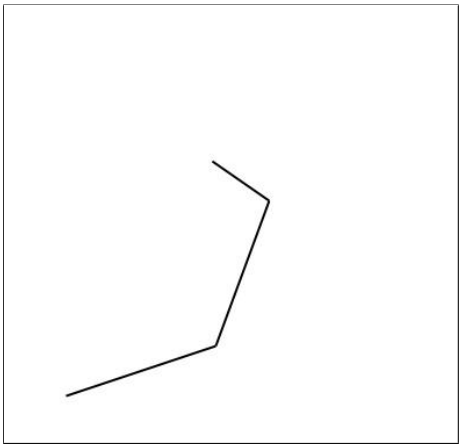
The algorithm to create the constraints map iterates through each constraint. If the current constraint is a line constraint, then in order to find all the DEM vertices influenced by this constraint a polygonal region around the line constraint is iden-

tified. First the line constraint is broken into its line segment components. Next, a trapezoidal region around each line segment is found (see Figure 4.21b). The total height of the trapezoid is equal to the impact distance of the constraint. Then a rectilinear bounding box is fit around all of the line segments' trapezoids (see Figure 4.21c). Then all of the indices within this box are queried to determine if they are within a trapezoid. If so, the distance to the line is calculated and its cost is added to the constraints cost map, see Figure 4.21d. This algorithm to process a line constraint is listed in Algorithm 2.

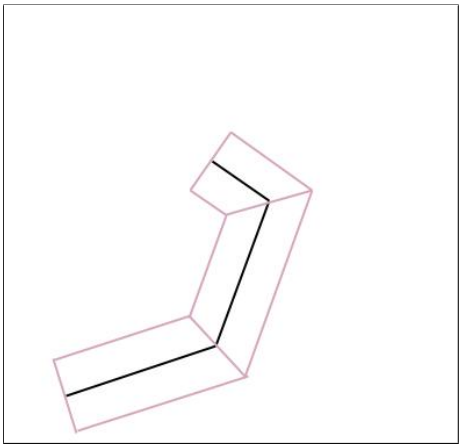
For each polygon constraint a larger polygon is created bounding the original polygon with a border equal to the width of the impact distance (see Figures 4.22a and 4.22b). Then a rectilinear bounding box is fit around the larger polygon (see Figure 4.22c), and all of the indices within the bounding box are queried (see Figure 4.22d). The ray casting algorithm is used to determine whether a point is within a polygon. This algorithm works by casting a horizontal ray through the point and counting how many times a polygon boundary is crossed. If the number of polygon crossings is even, then the point is outside of the polygon, but if the number of crossings is odd, then at one point the ray crossed a boundary and never exited the polygon. If an index is within the original polygon, the constraint's cost is added to the map at that index. If an index is outside of the original polygon, but within the larger polygon, then the the distance from the index to the original polygon is used to calculate the associated cost and add it to the constraints cost map. This algorithm to process a polygon constraint is listed in Algorithm 3.

This method of creating the constraints map does have limitations with concave polygons with overlapping geometry. Therefore these limitations occur for concave polygon constraints as well as line constraints where the bounding trapezoids overlap.

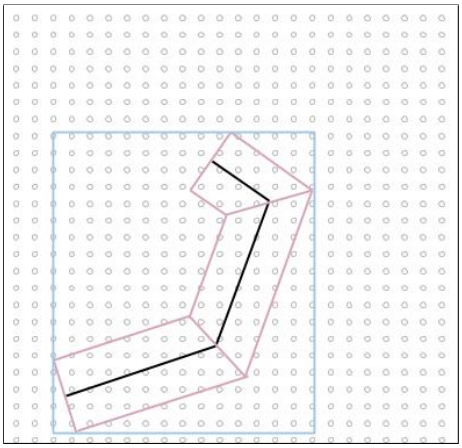
Once the map has been created containing pairs of vertices and their associated



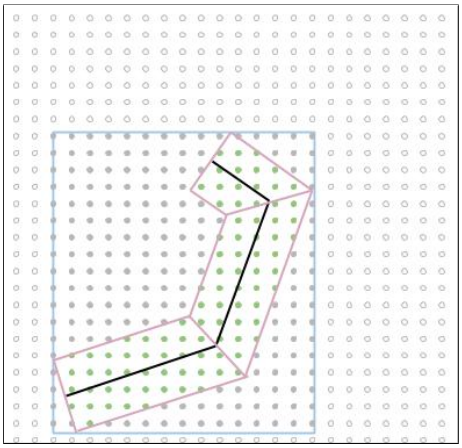
(a) User-Drawn Polyline



(b) Bounding Trapezoids Around Line Segments



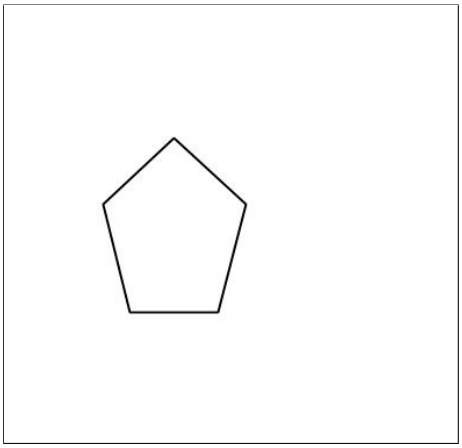
(c) Bounding Box Around Trapezoids



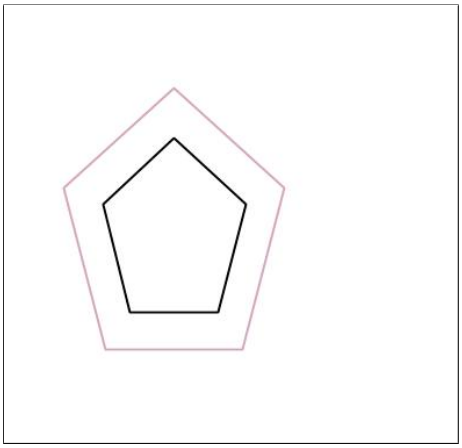
(d) Queried Indices within Bounding Box

Figure 4.21: Processing of a Line Constraint

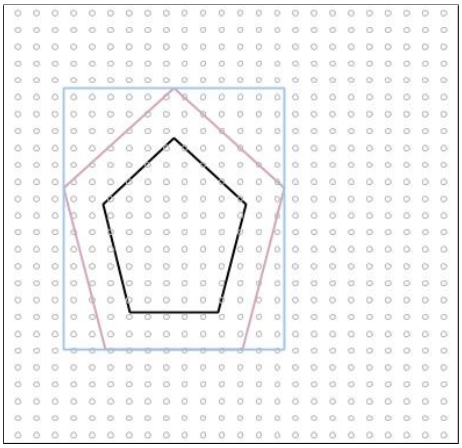
constraint weight sums, it is queried during each path computation. When the weight of an edge is being computed, if the destination of the edge is in the map, its constraint weight sum is added to the edge weight. Therefore the vertices affected by constraints are impacted in path computations. A demonstration of constraints affecting path computations is shown in Figures 4.23 and 4.24. Without constraints, the least caloric expenditure is achieved traveling north of the mountain range and then up and over to the destination. With a linear distribution line constraint and a binary distribution



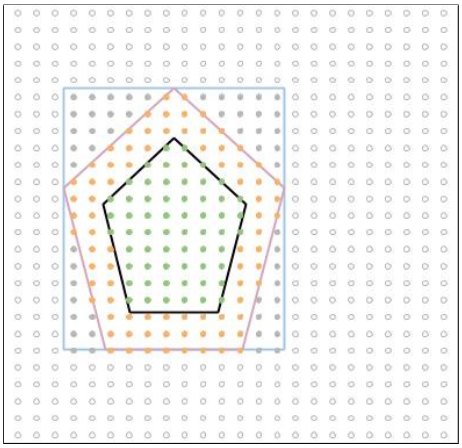
(a) User-Drawn Polygon



(b) Bounding Large Polygon Around Original Polygon



(c) Bounding Box Around Large Polygon



(d) Queried Indices within Bounding Box

Figure 4.22: Processing of a Polygon Constraint

polygon constraint, the previous route is blocked and the least caloric expenditure path approaches the mountain range sooner, then travels to the south of the mountain range to avoid the polygon constraint and ultimately reach the destination.

Algorithm 2: ProcessLineConstraint

```
Data: line constraint l
foreach line segment s in l do
    T[s] ← boundingTrapezoid(s);
B ← boundingBox(T);
foreach index p ∈ B do
    assigned ← false;
    foreach segment s in the line do
        if assigned = false && p ∈ T[s] then
            if l.distribution = binary then
                AddToCostMap(p, l.cost);
            else
                d ← CalculateDistancePointToSegment(p, s);
                if l.distribution = linear then
                    cost ← LinearCost(l.cost, d);
                    AddToCostMap(p, cost);
                else                          // l.distribution = quadratic
                    cost ← QuadraticCost(l.cost, d);
                    AddToCostMap(p, cost);
```

4.6 System Design

We have presented a collaborative web-based system that supports computation, visualization, and analysis of least caloric cost paths across terrain. Satellite image data was used to generate a high resolution DEM dataset and processing tools were used to create two additional lower resolution DEM datasets. Data management techniques

Algorithm 3: ProcessPolygonConstraint

```
Data: polygon contstraint g
1  P ← boundingPolygon(g);
2  B ← boundingBox(P);
3  foreach index p ∈ B do
4      if p ∈ g then
5          AddToCostMap(p, g.cost);
6      else
7          p ∈ P
8      if g.distribution = binary then
9          AddToCostMap(p, g.cost);
10     else
11         d ← CalculateDistancePointToPolygon(p, g);
12         if g.distribution = linear then
13             cost ← LinearCost(g.cost, d);
14             AddToCostMap(p, cost);
15         else                                  // g.distribution = quadratic
16             cost ← QuadraticCost(g.cost, d);
17             AddToCostMap(p, cost);
```

are employed to create a sub-sampled set of terrain data which is sent to clients to be displayed in the 3D scene. Data used in path computations is not sub-sampled, but rather the full resolution of the installed DEM file is used to maintain path accuracy. In order to reconcile these differing resolutions of paths and terrain data in the 3D scene, a set of adjusted path data is calculated in order to ensure each path point lies directly on the sub-sampled 3D terrain data. Constraints may be drawn by users in the form of lines and polygons in order to represent boundaries in the terrain. A map

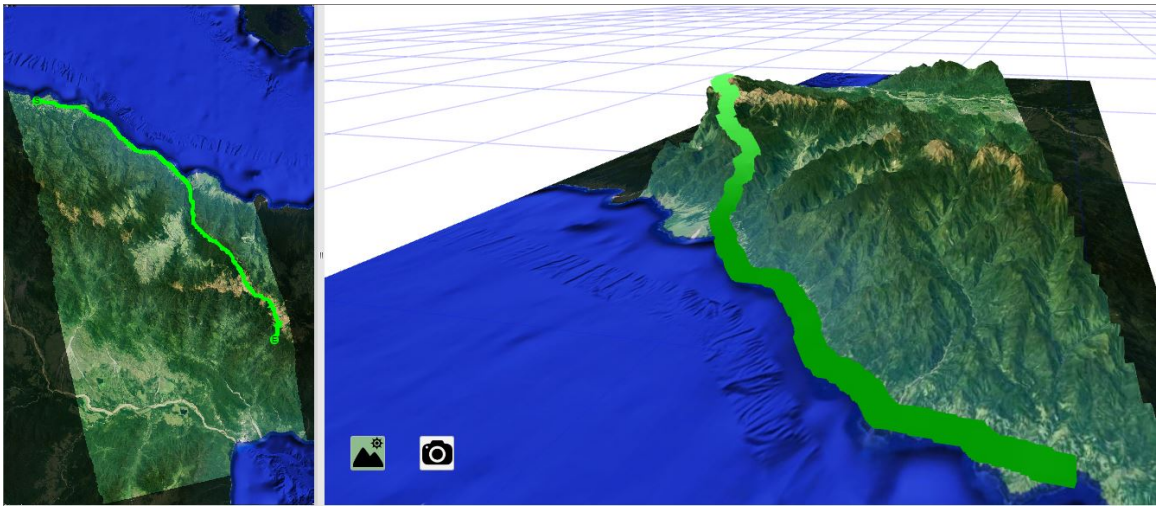


Figure 4.23: Long path without constraints

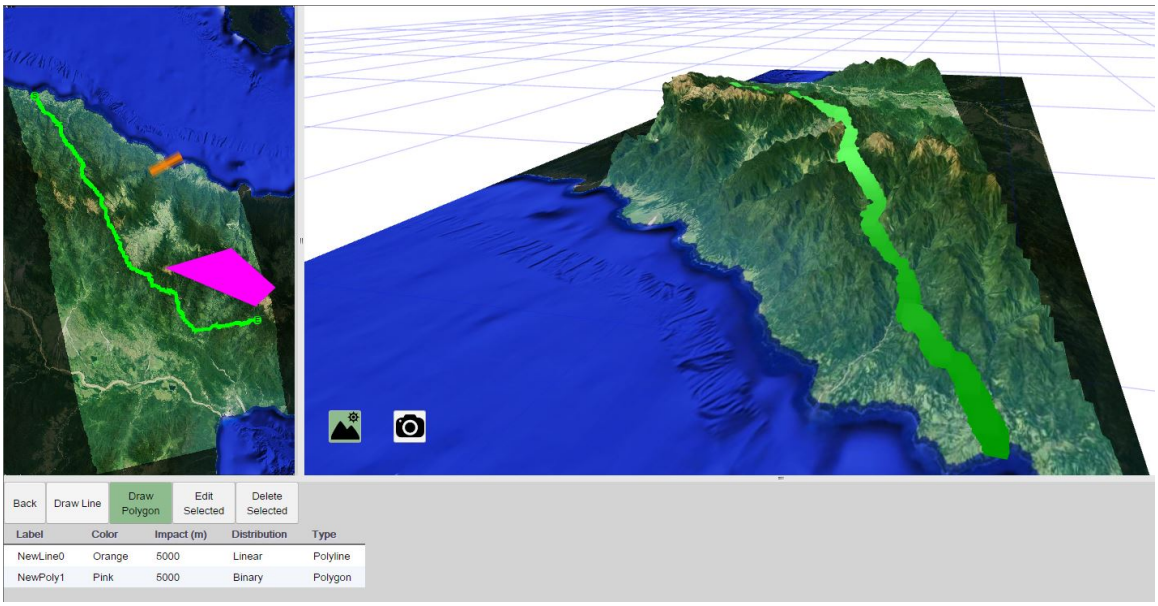


Figure 4.24: Long path with constraints

of DEM data vertices affected by drawn constraints is computed and maintained. During path computations, this map can be queried and the additional constraint cost may be added to the edge weight. The web application maintains system state at the server level so that each client shares the system state, fostering a collaborative environment where researchers can work together on the web application.

Chapter 5

VALIDATION

In order to validate the usability of this thesis, we employed three different techniques: a novice user feedback study, expert feedback, and an error analysis of the trade-off between data resolution versus path computation time and interactivity. Both novice user and expert feedback is used to gauge how effective the web application is for users with varying backgrounds in Anthropology, spatial data visualization tools, and least cost caloric paths. An analysis of the error in paths of differing resolutions is presented in order to examine the DEM data resolution trade-off of path accuracy versus computation time and interactivity.

5.1 Novice User Feedback

This thesis was tested for novice user feedback with a small group of five computer science students working on theses related to computer graphics. During this user study, each student had a computer with the web application open in a browser. One by one, users were directed to perform a task while each of the other users observed the changes updated on their own browsers. The tasks performed were adding a new path, editing the start point of an existing path, changing the algorithm type of a path, adding a constraint, exporting a path in .gpx format and importing into Google Earth, and navigating the 3D scene.

The following quotations were included in the feedback form:

With respect to changing the algorithm type of an existing path:

> *Very easy to see the different options and how changing the algorithm affected the path. Having information about the different algorithms on*

the page was useful.

With respect to adding a new constraint:

Adding a constraint line was intuitive. I added the line and saw an update immediately.

With respect to exporting a path:

It was straightforward to click on path and then export. Would be nice to have the gpx to kmz converter built in though.

With respect to navigating the 3D terrain view:

It was pretty easy. Little bit of lag. Moved around the terrain by clicking and dragging on the view. It was pretty intuitive.

When asked about the differences in the paths produced by the Wood & Wood least caloric cost algorithm and the 2D distance algorithm:

The Wood & Wood algorithm looked more complicated (due to taking in elevation) whereas the 2D distance algorithm was much simpler.

and

The 2D algorithm seemed to take the most direct route possible, as if it were a bird flying above the terrain. The Wood & Wood algorithm seemed to optimize for routes with little change in elevation, which would be easier to traverse.

When asked to indicated whether the Wood & Wood least caloric cost algorithm path looked like the path they would choose to travel from the given start and end-points, all of the participants marked a four or a five on a scale of one to five (one being "Not at all how I would travel" and five being "Yes how I would travel").

When describing what it was like to use the tool, users said:

> *Easy to use.*
>
> *Using this tool was easy and fun to collaborate on with constraints, I liked being able to see everyone's changes. The 3d view was a little frustrating, it was more difficult to collaborate when everyone was modifying it.*
>
> *Adding collaborative markers (2d/3d) might be an effective way to highlight areas of interests.*
>
> *Easy but a separate 3D view would be cool. The collaboration part works really well.*
>
> *The tool had a very nice UI which was easy to use. The 3D view of the terrain was especially cool. It would be nice if the 3D view was local to each user, rather than shared between everyone on the site.*

5.2 Anthropologist Feedback

This thesis was developed in conjunction with expert input from anthropologist and creator of "Energetic Analyst" [28, 29], Brian Wood. Brian guided the design of features, UI, and provided feedback on the end result of the web application. Please see Appendix 6 for Brian's full feedback.

5.3 DEM Resolution Error Analysis

In order to examine the differences in running the different resolutions of DEM terrain data files, 15 paths were computed at both low resolution (139m) and high resolution (13.9m) with the Wood & Wood caloric expenditure edge weight algorithm, without any constraints present. It is assumed that the path computed on the high resolution file is more accurate than the path computed on the low resolution due to the higher sampling rate of the terrain data. For a given pair of paths, the start and endpoints are the same. The paths are labeled with a brief description of the terrain they traverse. For each path pair, the error distance (in meters) was calculated for each point in the low resolution path compared to the high resolution path. These error distances were averaged together to compute a mean error distance per path pair. This analysis indicates that over the 15 path pairs, the paths computed on low resolution DEM data are computed within minutes or even seconds and exhibit an average error distance of 183.13m, or 83.22m when excluding the *bottomHill2* path pair outlier with extreme error distances.

5.3.1 Signed Error Distance

The error of each path pair is measured as follows: for each vertex in the low resolution path, the closest point on the high resolution path is computed and the 2D Euclidean distance between them recorded. 3D distance would introduce larger error distance values in paths with great elevation differences, such as mountainous regions. 2D distance is used to examine the error for both flat and mountainous paths without this extra error due to elevation differences. It is also useful to examine 2D error distance as opposed to 3D because the path is constructed in 2D as latitude and longitude values. A signed distance is used, meaning that from a birds-eye view all low resolution points lying to the left of the high resolution path have a negative sign

and all low resolution points lying to the right of the high resolution path have a positive sign. This sign to side assignment is arbitrary and is mainly used to examine a histogram centered about zero during the error analysis. If a positive distance were used instead, the resulting histogram would be one-sided. These recorded distances for each node of the low resolution path are the error values associated with each node, indicating how far off (in meters) the low resolution path currently is from the accurate high resolution path.

Table 5.1 shows the comparison of vertex count and edge weight sums in kilocalories for the 15 path pairs. The error percentage is taken with the high resolution path as the expected value and the low resolution path as the experimental value. All path pairs have caloric expenditure differences that do not exceed 6%, though all path pairs except for *bottomHill2* and *flatMedium* show slightly larger kilocalorie sums for high resolution paths. This is due to the fact that the Wood & Wood edge weight algorithm is nonlinear with respect to elevation changes. Therefore high resolution paths containing more vertices will result in high caloric expenditure values.

Table 5.2 shows that the 2D distances never differ by more than 4.4% and the 3D distances never differ by 4.3%. As expected, higher 3D distances and error percentages are seen in the paths that traverse mountains and hills as opposed to the flat paths, due to the variation in elevation. Elevation differences at different DEM data resolutions result in differing paths and thus 2D distance as well. Therefore similar error percentages are seen in 2D distance as well.

5.3.2 Error Metrics

Table 5.3 shows the minimum, maximum, mean, mean magnitude, and standard deviation of error distance in meters while Figure 5.1 displays the error ranges and means for the 15 path pairs. The minimum and maximum error distance values indicate

Table 5.1: Vertex counts and caloric expenditures for 15 path pairs

Path	Low Res Vertices	High Res Vertices	Low Res kCal	High Res kCal	kCal Error Percentage
bottomHill1	374	3764	4672.98	4726.27	-1.13%
bottomHill2	387	4089	4225.15	4148.66	1.84%
flatLong	643	6426	5991.28	6016	-0.41%
flatMedium	440	4390	3867.83	3836.08	0.83%
flatNorthCoast	533	5337	5190.52	5294.15	-1.96%
flatShort1	119	1178	986.26	992.63	-0.64%
flatShort2	167	1709	1861.99	1873.15	-0.6%
flatShort3	169	1685	1348.88	1356.94	-0.59%
mountain1	434	4348	6785.32	7137.62	-4.94%
mountain2	476	5068	7996.94	8316.87	-3.85%
mountain3	455	4775	7025.21	7276.78	-3.46%
mountain4	249	2652	4899.03	5200.12	-5.79%
mountain5	212	2178	2917.63	3087.59	-5.50%
mountain6	259	2796	4615.68	4823.76	-4.31%
mountain7	380	4057	5858.73	6186.55	-5.3%

the error of the low resolution path points furthest away from the high resolution path. Minimum error indicates the signed distance value of the low resolution point furthest away to the left of the high resolution path while maximum error indicates the signed distance value of the low resolution point furthest away to the right of the high resolution path.

From these signed error distance values, a histogram is generated for each of the

Table 5.2: 2D and 3D distances for 15 path pairs

Path	Low Res 2D Dist (m)	High Res 2D Dist (m)	2D Dist Error	Low Res 3D Dist (m)	High Res 3D Dist (m)	3D Dist Error
bottomHill1	63074.9	63348.78	-0.43%	63221.37	63507.26	-0.45%
bottomHill2	67776.65	70836.76	-4.32%	67996.29	70987.31	-4.21%
flatLong	104415.18	104316.73	0.09%	104431.35	104336.87	0.09%
flatMedium	66561.07	66453.1	0.16%	66598.18	66485.28	0.17%
flatNorthCoast	87790.79	87234.35	0.64%	87870.6	87348.75	0.6%
flatShort1	17773.27	17731.43	0.24%	17773.68	17732.27	0.23%
flatShort2	25279.39	25678.5	-1.55%	25354.77	25747.46	-1.53%
flatShort3	24213.72	24257.68	-0.18%	24215.33	24259.88	-0.18%
mountain1	73511.67	73765.39	-0.34%	74253.84	74630.74	-0.51%
mountain2	82626.11	85360.25	-3.2%	83562.42	86345.41	-3.22%
mountain3	75899.87	77901.23	-2.57%	76741.21	78786.89	-2.6%
mountain4	40786.16	42304.99	-3.59%	41465.14	43046.68	-3.67%
mountain5	35427.47	35803.65	-1.05%	35881.68	36293.21	-1.13%
mountain6	43207.27	44964.89	-3.91%	43634.26	45427.68	-3.95%
mountain7	63926.89	66601.99	-4.02%	64370.4	67094.57	-4.06%

path pairs in Figure 5.9.

Most of the path pairs have signed error distance histograms with fairly normal shapes, centered near zero. This histogram characterization indicates that the bulk of the population of the low resolution path points has little error. The average error seen over all six path pairs, normalized for path length, is 183.13m, or 83.22m when

Table 5.3: Error metrics for 15 path pairs

Path	Min Error Distance (m)	Max Error Distance (m)	Mean Error Distance (m)	Mean Error Magnitude (m)	Standard Deviation (m)
bottomHill1	-291.52	236.71	-11.54	67.46	94.11
bottomHill2	-81.69	5740.26	1450.73	1454	1705.36
flatLong	-527.66	1870.53	138.92	257.33	422.78
flatMedium	-1161.02	2299.43	269.24	646.48	823.74
flatNorthCoast	-296.53	1919.77	69.84	175.84	404.74
flatShort1	-647.47	124.84	-212.35	214.92	187.01
flatShort2	-586.36	1664.47	603.445	701.91	661.3
flatShort3	-448.24	98.78	-180.64	185.6	129.86
mountain1	-490	515.74	3.03	124.11	173.27
mountain2	-1234.22	1246.32	58.17	163.45	275.76
mountain3	-547.42	1328.75	131.94	211.35	334.58
mountain4	-556.08	1376.30	122.04	219.91	328.81
mountain5	-669.13	238.41	-31.03	87.15	134.19
mountain6	-356.47	443.88	4.61	78.97	113.98
mountain7	-369.56	684.2	23.46	83.38	138.13

excluding the *bottomHill2* path pair outlier with extreme error distances. Framing this in the reference of the high resolution path at 13.9 meters and the low resolution path at 139 meters, this average error is 1.32 low resolution grid squares and over 13 high resolution grid squares. Given that high resolution DEM file paths have

computation times of hours while low resolution DEM file paths have minutes- or even seconds-long computation times, this average error could be worthwhile. This would be especially true when computing path estimates or planning the possible paths to run at high resolution later.

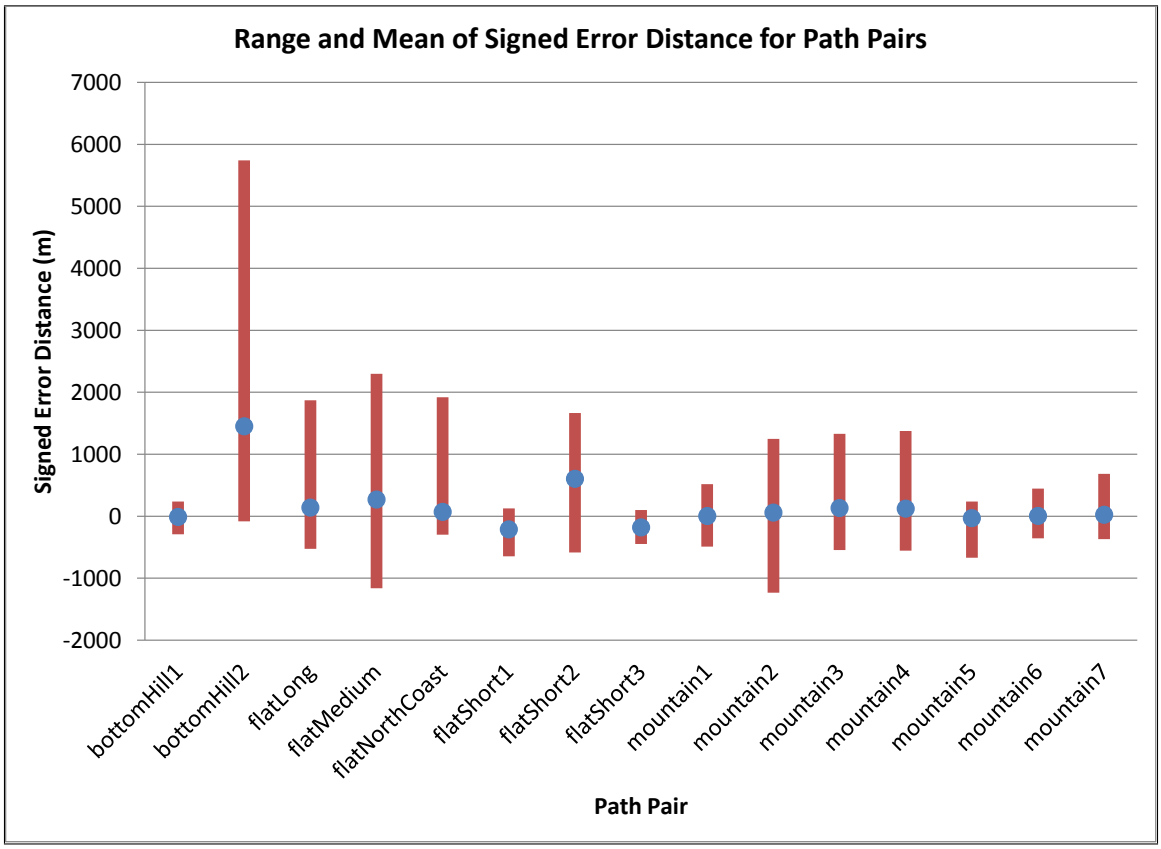
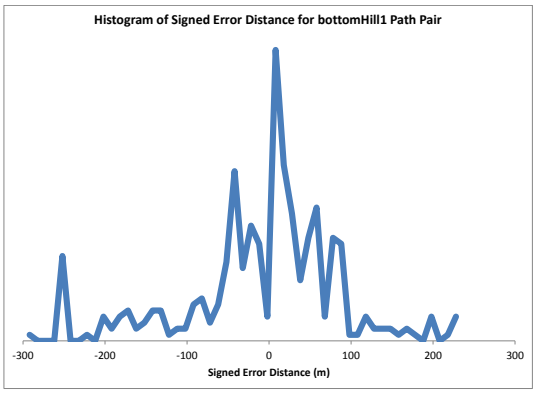
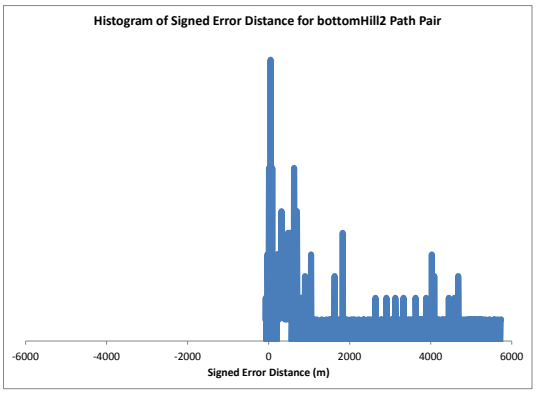


Figure 5.1: Error Distance ranges and means in meters for 15 path pairs

The 15 path pairs can be partitioned as the two mixed hill paths (*bottomhill1* and *bottomhill2*), the coastal path (*flatNorthCoast*), the valley paths (*flatShort1*, *flatShort2*, *flatShort3*, *flatMedium*, and *flatLong*), and the mountain paths (*mountain1*, *mountain2*,*mountain3*,*mountain4*,*mountain5*,*mountain6*, and *mountain7*). Figures 5.2a, 5.2b, 5.4b, 5.5a, 5.5b, 5.3b, **??**, **??**, 5.6a, 5.6b, 5.7a, 5.7b, 5.8a, 5.8b, and 5.9a show the histograms of all of these 15 path pairs while Figures 5.10, 5.11, 5.15, 5.16, 5.17, 5.13, 5.12, 5.14, 5.18, 5.19, 5.20, 5.21, 5.22, 5.23, and 5.24, show the overlaid path pairs respectively. The *bottomHill2*, *flatShort1*, *flatShort2*, *flatShort3*, and

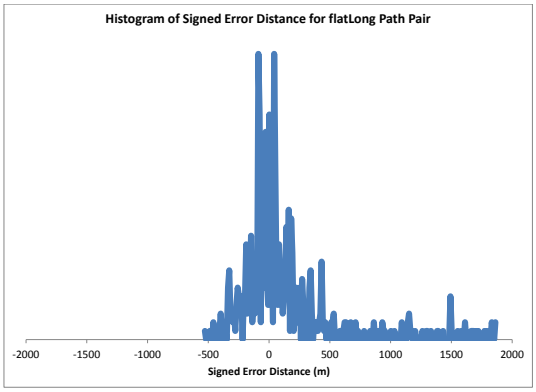


(a) Histogram of bottomHill1

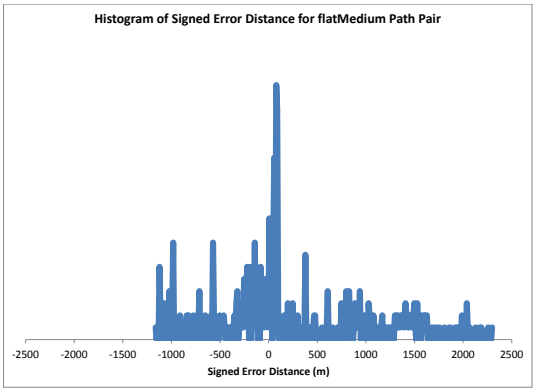


(b) Histogram of bottomHill2

Figure 5.2: Histograms of error distances for bottomHill paths



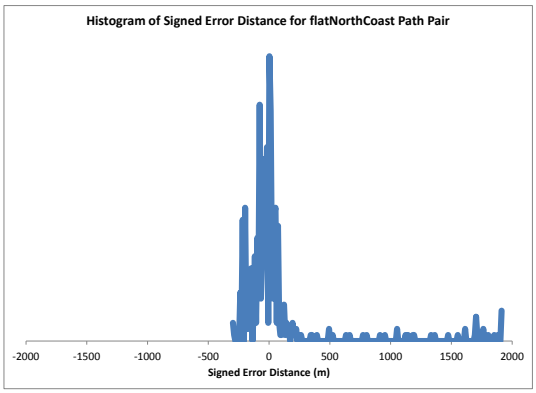
(a) Histogram of flatLong



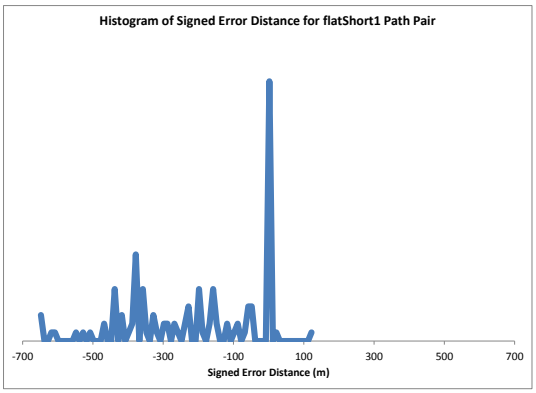
(b) Histogram of flatMedium

Figure 5.3: Histograms of error distances for flat long and medium paths

mountain4 paths have skew in their histograms. Figures 5.11, 5.15, 5.16, 5.17, and 5.21 show the path pairs side-by-side on the 2D map of the region. There are many large differences between the low and high resolution paths in all of these path pairs, indicating why the histograms look like they do. The *flatShort1* path pair has a lot of error in a short path. Most of the error occurs when the low resolution path is further north, or to the left of, the high resolution path (given the lower left starting point and upper right endpoint). This prolonged one-sided error distance is demonstrated in the skew of the histogram for this path pair. For comparison, Figure 5.18 shows the *mountain1* pair of paths overlaid. There are few and minimal differences in

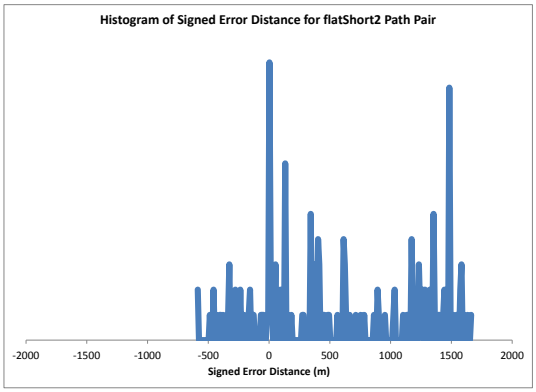


(a) Histogram of flatNorthCoast

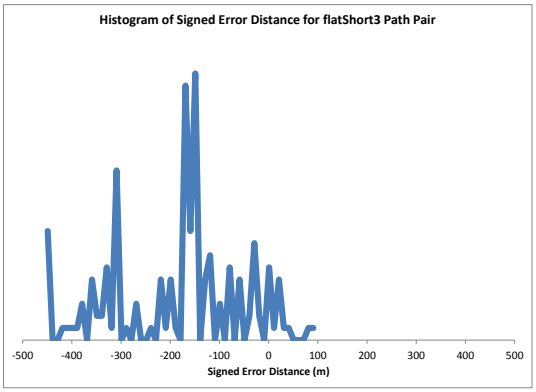


(b) Histogram of flatShort1

Figure 5.4: Histograms of error distances for flat north coast and short1 paths



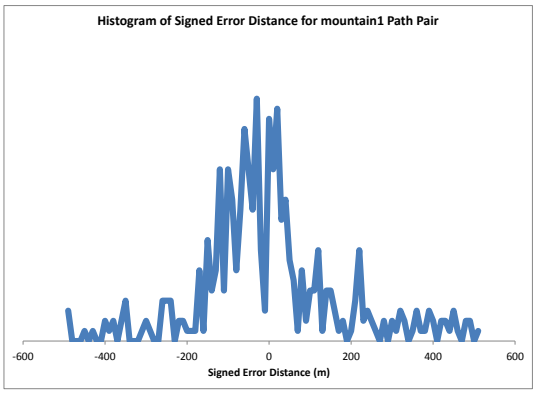
(a) Histogram of flatShort2



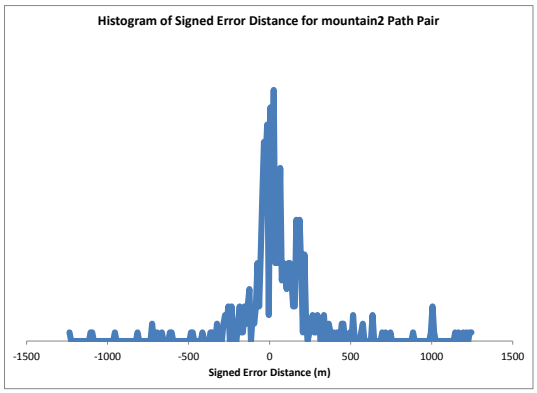
(b) Histogram of flatShort3

Figure 5.5: Histograms of error distances for flatShort paths 2 and 3

these paths, which is mirrored in the characterization of the path pair as a histogram centered very close to zero.

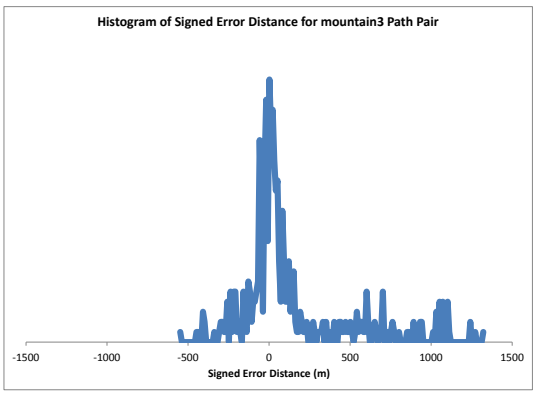


(a) Histogram of mountain1

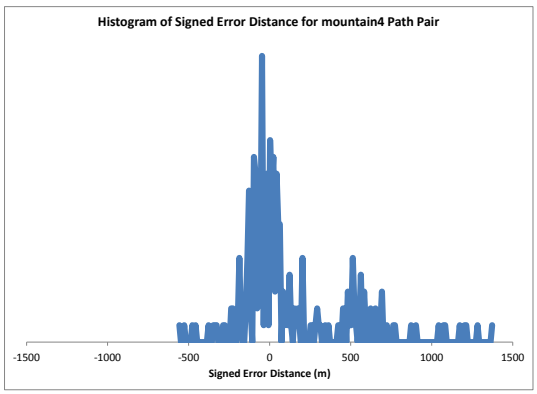


(b) Histogram of mountain2

Figure 5.6: Histograms of error distances for mountain paths 1 and 2

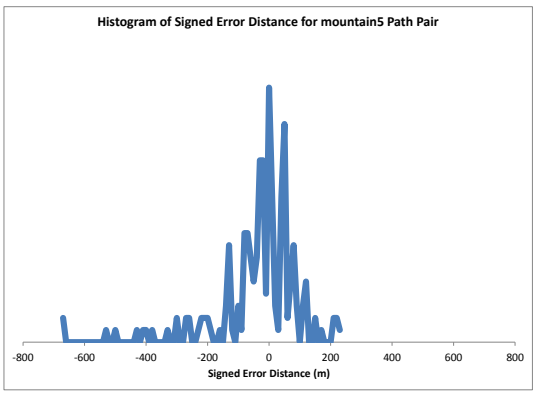


(a) Histogram of mountain3

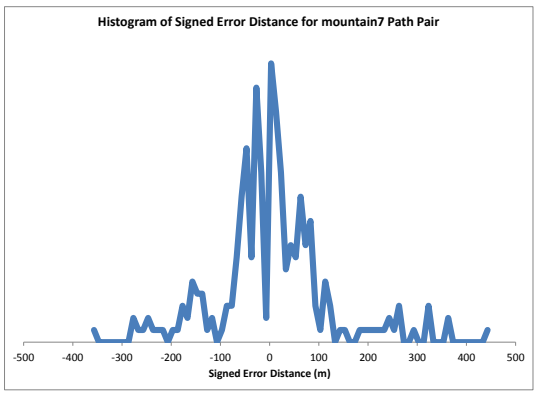


(b) Histogram of mountain4

Figure 5.7: Histograms of error distances for mountain paths 3 and 4

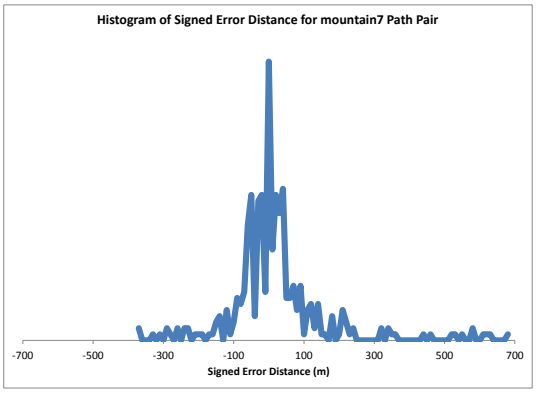


(a) Histogram of mountain5



(b) Histogram of mountain6

Figure 5.8: Histograms of error distances for mountain paths 5 and 6



(a) Histogram of mountain7

Figure 5.9: Histograms of error distances for mountain7 paths

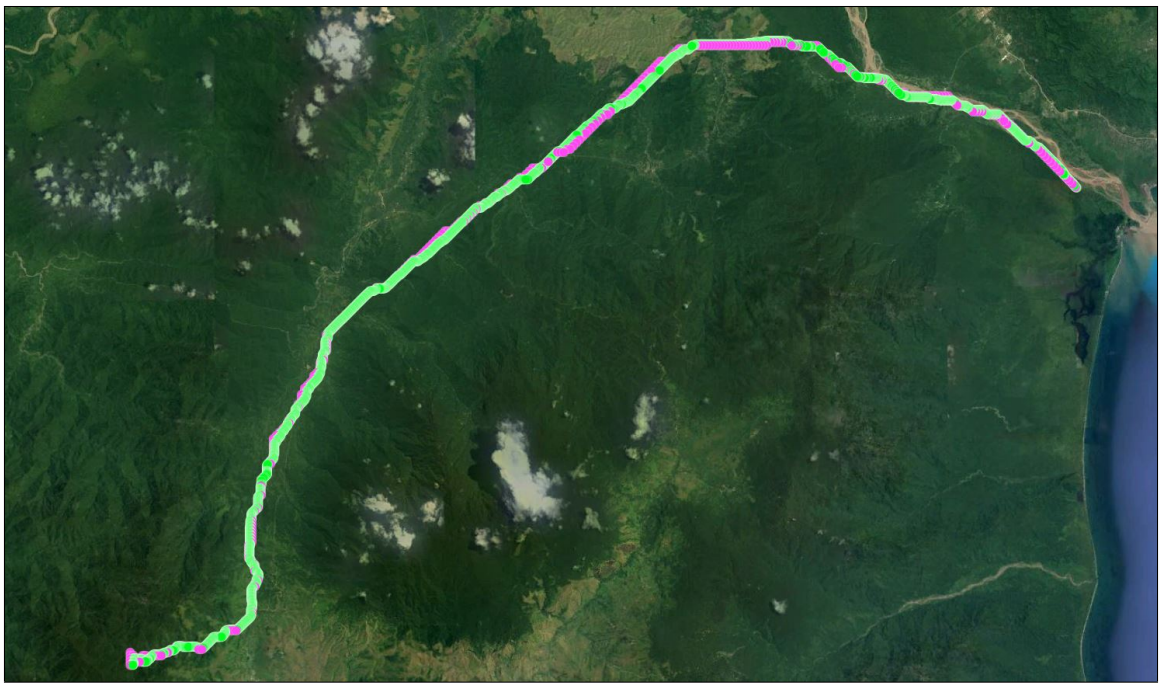


Figure 5.10: Low (pink) and high (green) resolution *bottomHill1* paths

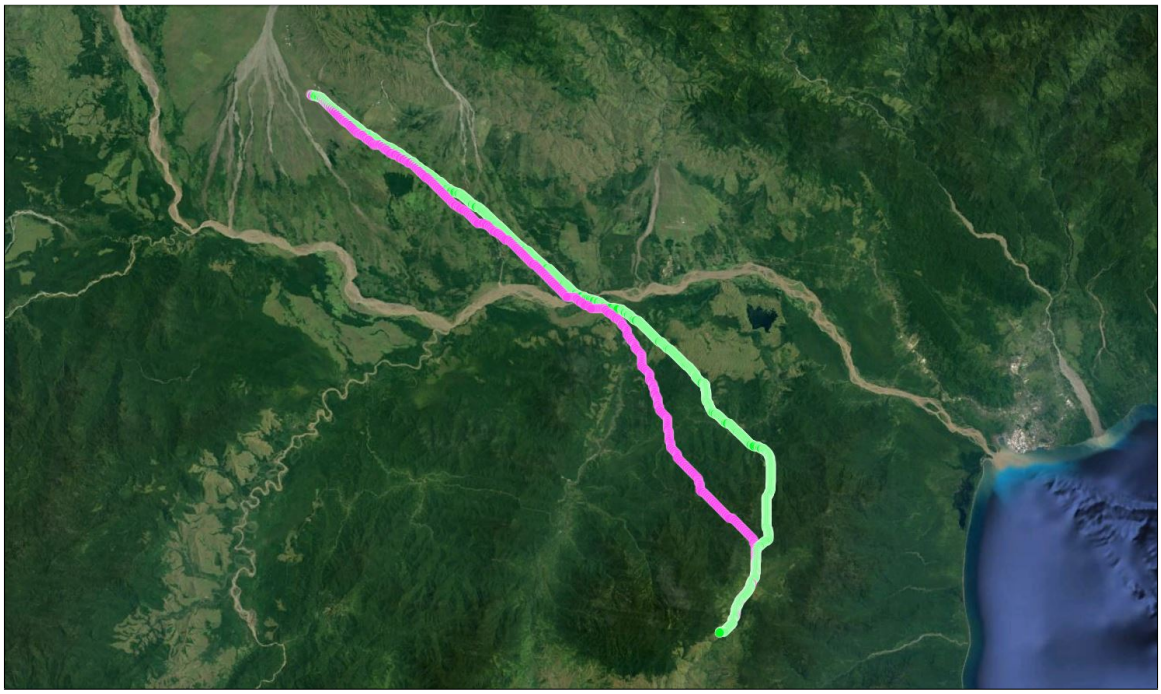


Figure 5.11: Low (pink) and high (green) resolution *bottomHill2* paths

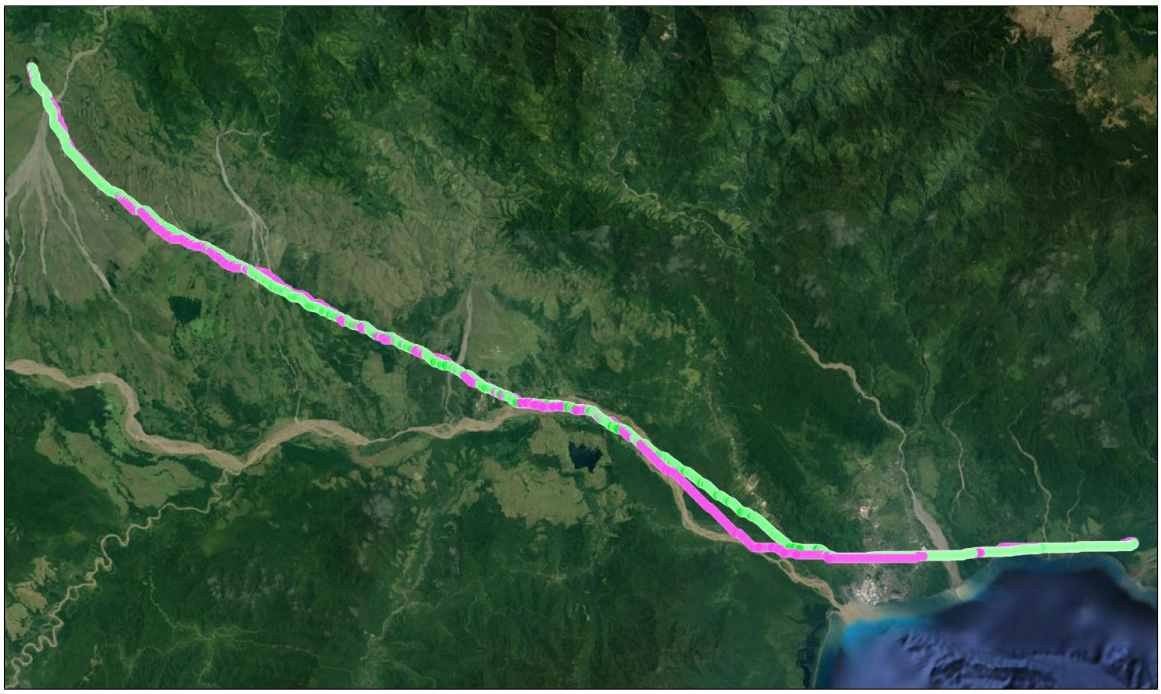


Figure 5.12: Low (pink) and high (green) resolution *flatLong* paths

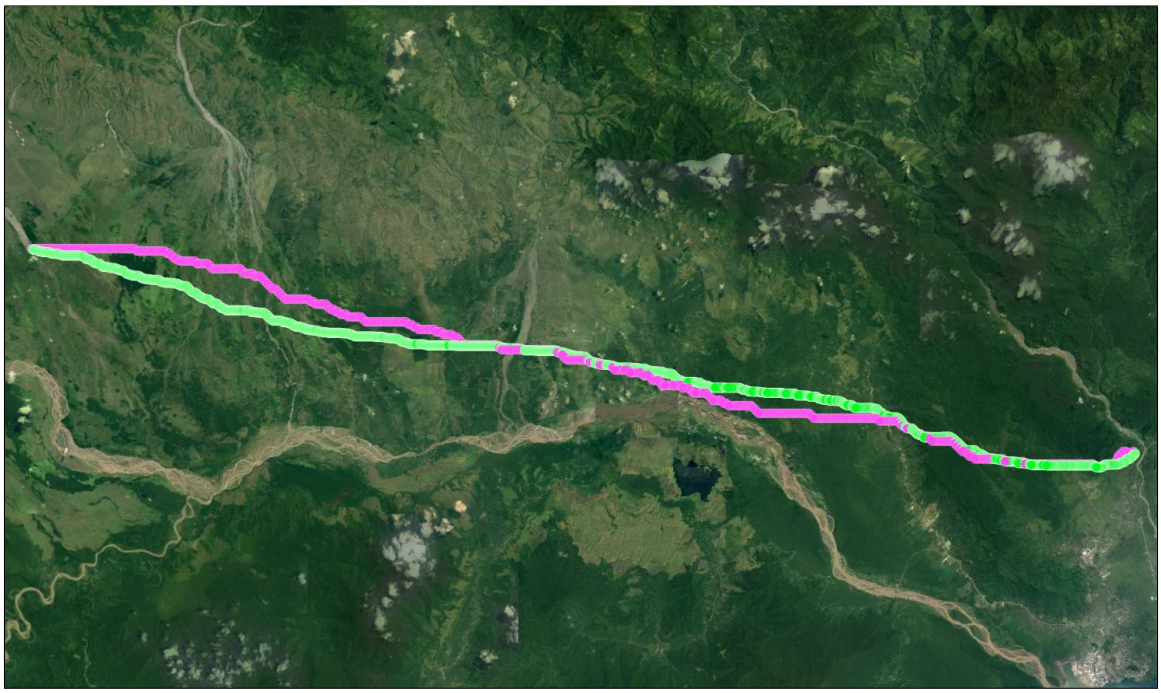


Figure 5.13: Low (pink) and high (green) resolution *flatMedium* paths

Figure 5.14: Low (pink) and high (green) resolution *flatNorthCoast* paths

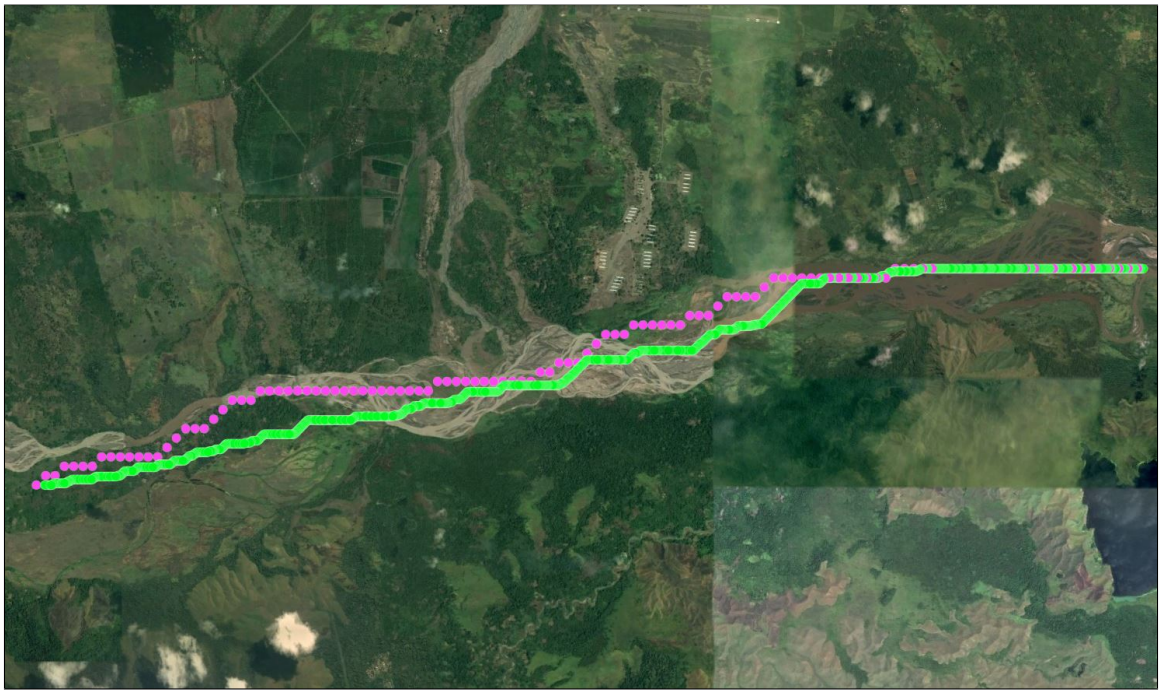


Figure 5.15: Low (pink) and high (green) resolution *flatShort* paths

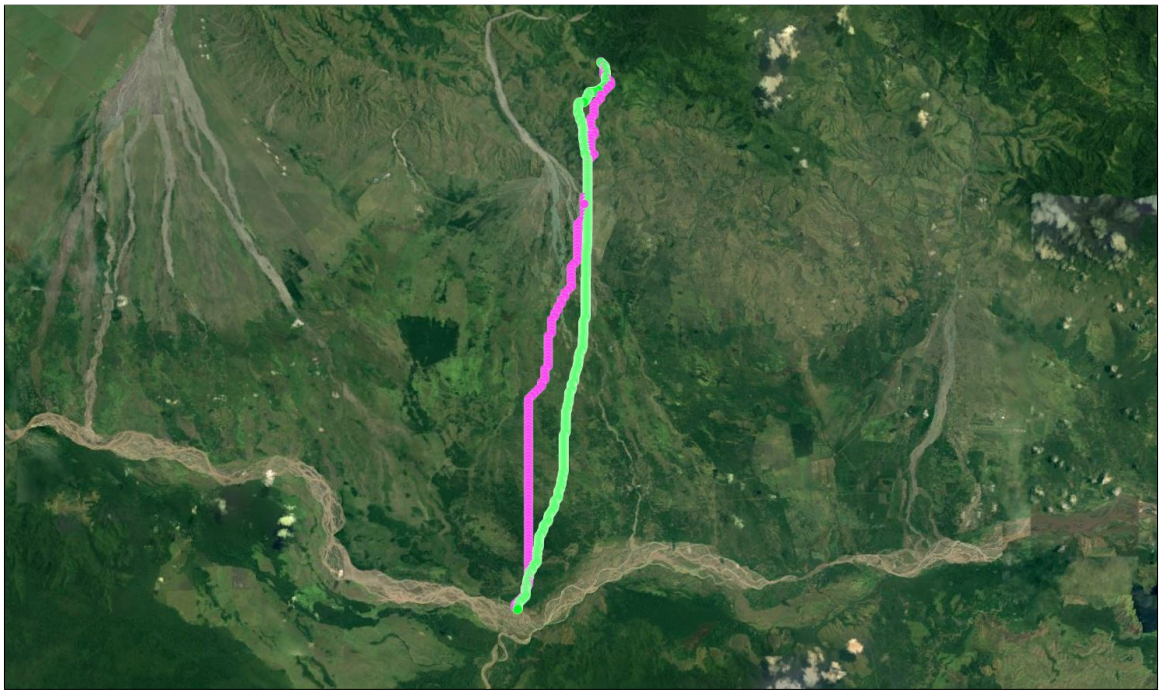


Figure 5.16: Low (pink) and high (green) resolution *flatShort2* paths

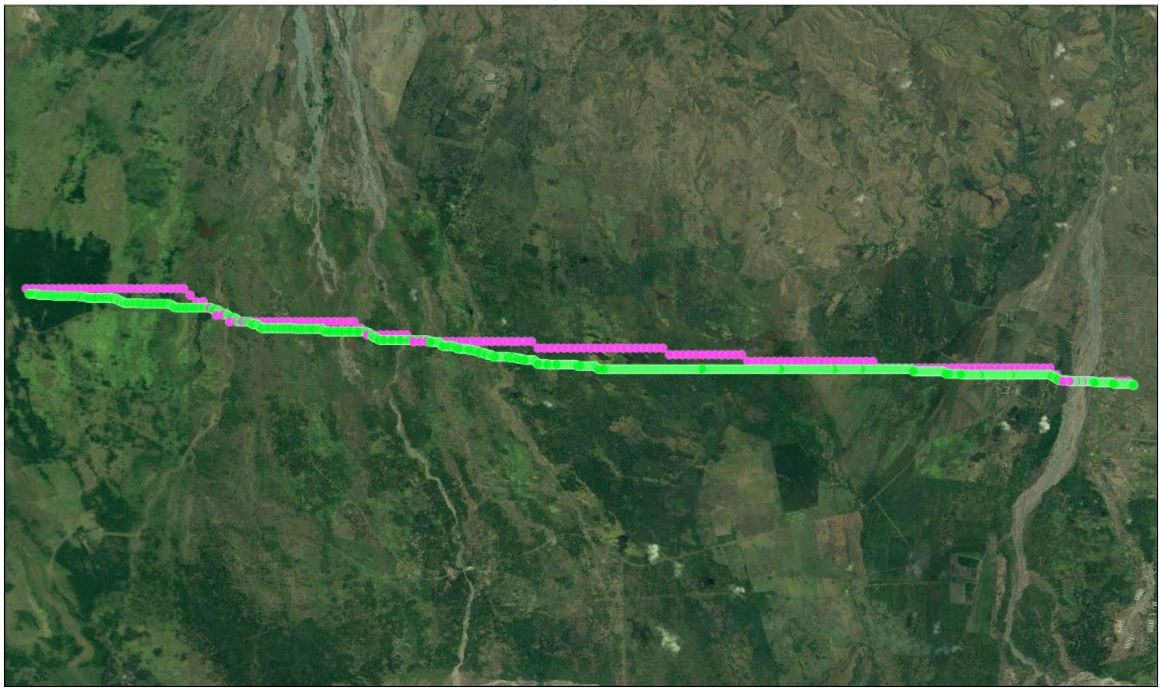


Figure 5.17: Low (pink) and high (green) resolution *flatShort3* paths

Figure 5.18: Low (pink) and high (green) resolution *mountain1* paths

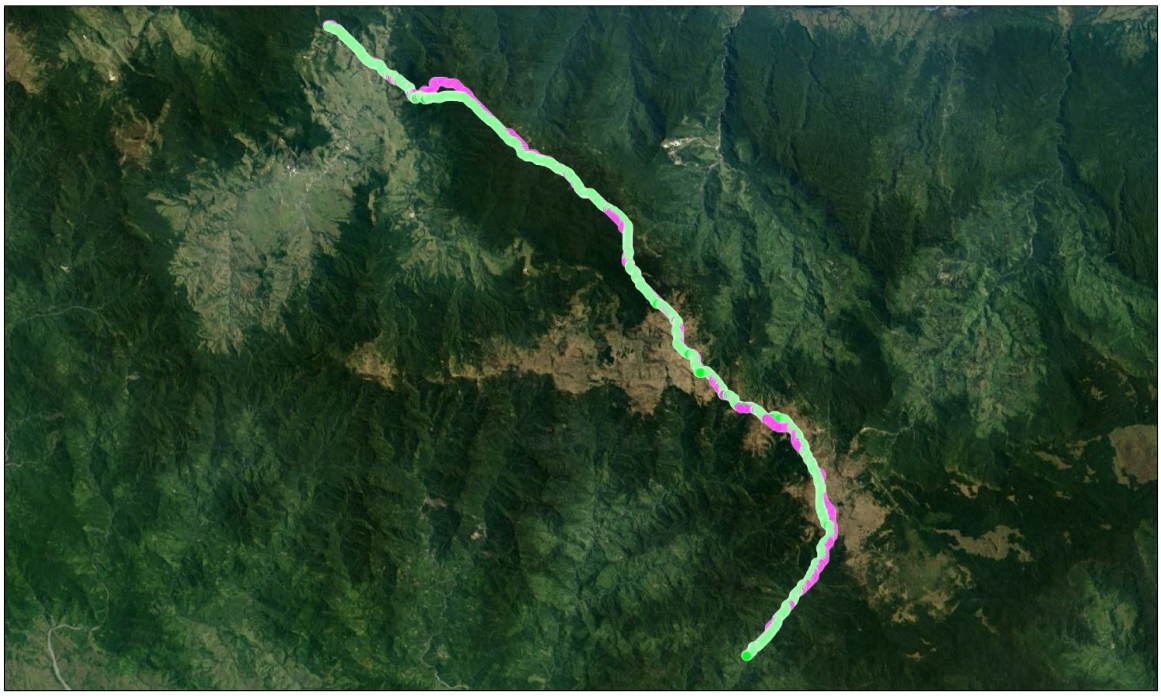


Figure 5.19: Low (pink) and high (green) resolution *mountain2* paths

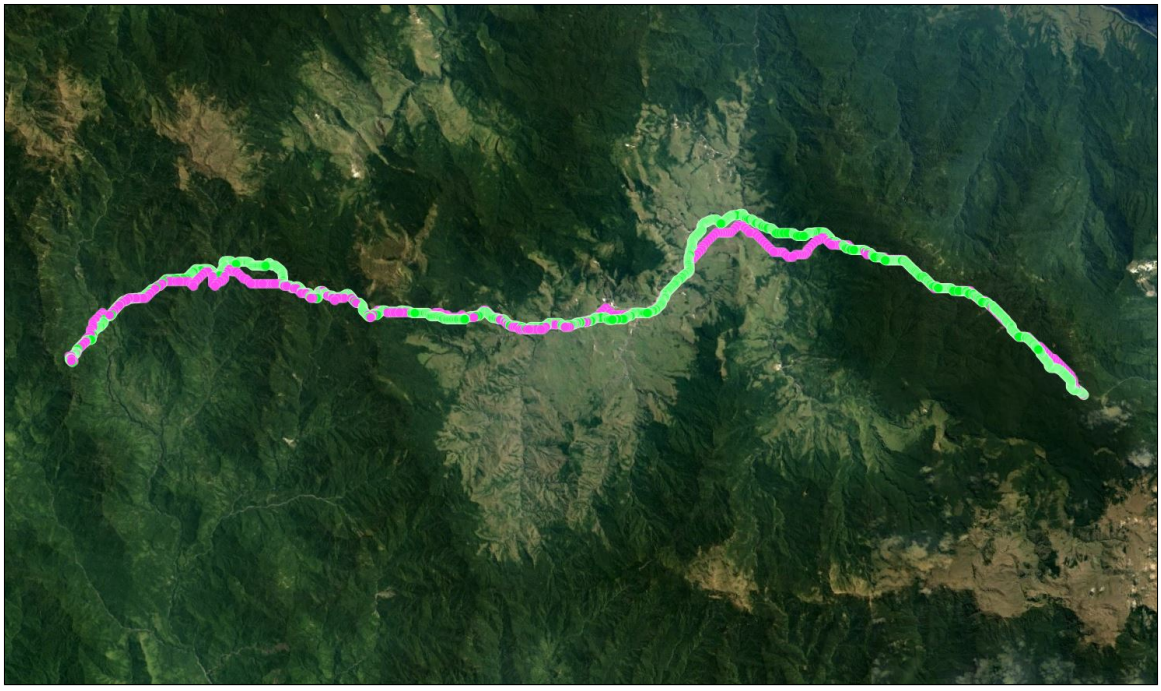


Figure 5.20: Low (pink) and high (green) resolution *mountain3* paths

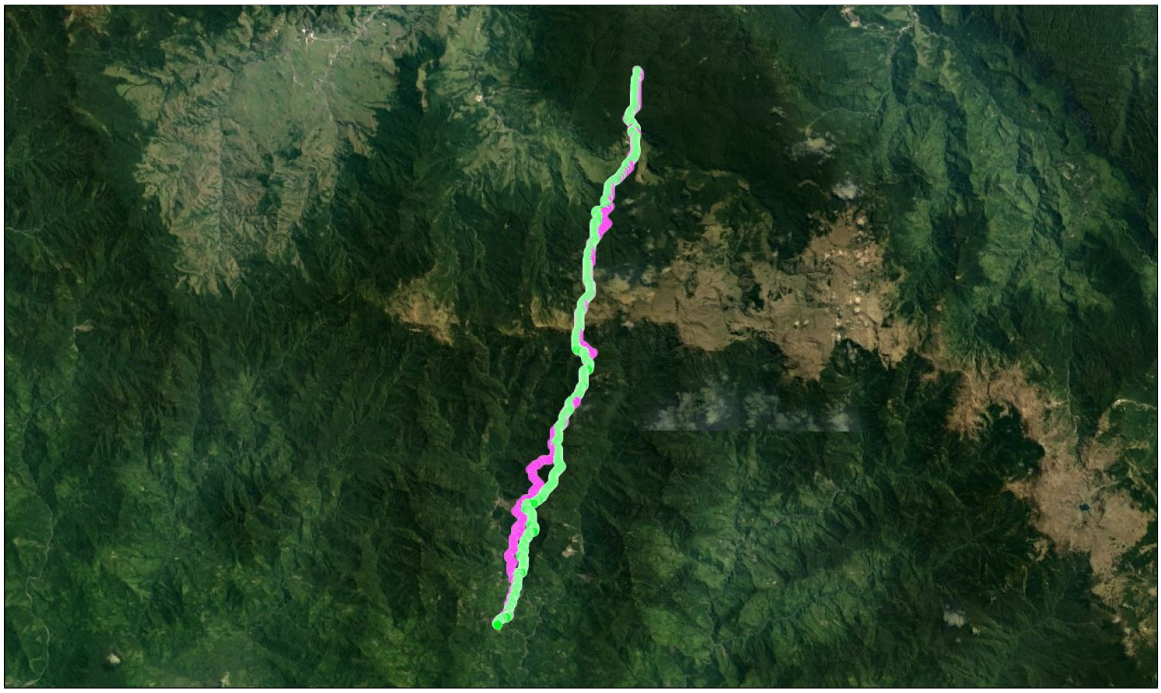


Figure 5.21: Low (pink) and high (green) resolution *mountain4* paths

Figure 5.22: Low (pink) and high (green) resolution *mountain5* paths

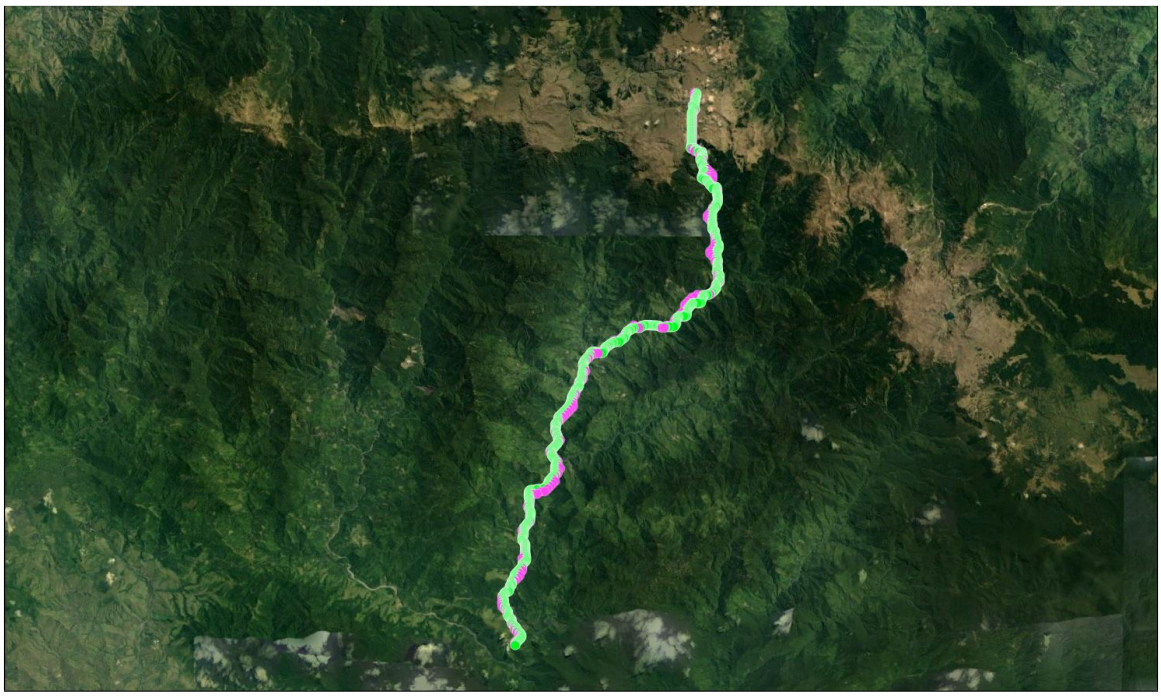


Figure 5.23: Low (pink) and high (green) resolution *mountain6* paths

Figure 5.24: Low (pink) and high (green) resolution *mountain7* paths

Chapter 6

CONCLUSIONS & FUTURE WORK

This work has presented a framework for a web-based collaborative tool to compute, view, and analyze least cost caloric paths with constraints across terrain. Novel to this work is the web platform in which to use the application as well as the resulting collaborative nature of the tool. Additionally this work presents a system of integrating user-drawn constraints into least caloric cost calculations so as to simulate sociopolitical or geographic boundaries in which travel is limited or not permitted. Also included is an error analysis comparing path error distance of 15 path pairs computed at low and high resolution DEM data, the results of which indicate that the average error seen in a path generated with low resolution data is 183.13m, or 83.22m when excluding the *bottomHill2* path pair outlier with extreme error distances.

There are many areas in which to pursue future work from this thesis. One such area is creating packages of DEM data and associated satellite imagery to examine other regions of the Earth to expand on the current region in Papua New Guinea. A new feature could be to allow users to upload their own packages of DEM data and associated satellite imagery. Sourcing the DEM data and satellite images in an alternative way would also be an interesting way to make this process of creating packages of new data regions easier, faster, and more automated.

There is work to be done in the display of the paths in the 3D scene. Currently the paths look very large relative to the terrain and a future improvement would be to scale the width of the 3D paths by the zoom level of the camera. Additionally, alternative ways to display high resolution paths over low resolution sub-sampled terrain data could be explored. One such technique would be to show the 3D terrain data with some transparency, and the 3D paths at their full resolution at their original

elevations, rather than the adjusted path data elevations. This way paths could be examined at their full resolution accurate locations.

Furthermore, the 3D scene navigation can be improved. In order to continue the collaborative nature of the web application, the 3D view is a part of the system state maintained at the server level and published out to clients so that researchers can all examine the same 3D view of the terrain at the same time. However, another way to reduce the lag associated with constant updating of the 3D view would be to eliminate the 3D view from the system state. This would allow each user to freely navigate the 3D scene without worrying about disrupting their collaborator's 3D view. Perhaps a good hybrid of these approaches would be to have a "Publish View" button in the 3D scene so that each user could freely move about, but when all collaborators want to ensure they are all examining the same area, one user could publish her view, updating the other users with the same 3D view. A similar potential feature would be a mode to indicate whether a client is in "Shared Viewing" or "Local Viewing" mode, so that users could easily switch back and forth.

Allowing different formats in which to export paths would be useful. Currently only the .gpx file format is supported, but .kml and .kmz are the file types used by Google Earth and Google Maps, thus these would be easy file formats to use cross-platform.

A new feature to improve the constraints in the system could be to allow users to upload constraints in a .gpx, .kml, or .kmz file format. This way users have more control over the precision and accuracy of the constraints in the system. Additionally, constraints could be further visualized in the 3D view. Currently all constraints have a total cost of some arbitrary large value. Interesting further work would be to analyze the cost of constraints required in order to force paths to avoid the constraint regions fully. Then this analysis could be used to develop some metric which could

dynamically be used to set this cost value per constraint, rather than the current static value that every constraint shares.

In order to further utilize the collaborative nature of the system while making the system more usable, an account system could be implemented to add a layer of security and to ensure that only authorized users can change the state of the system. Another collaborative feature would be markers or cursors that users could place on the 3D terrain or in the 2D map to indicate a location of interest.

Alternative path finding algorithms could be explored to improve performance such as those used in the work of Rickwald and Tsui [22, 24, 25].

Additional work in the error analysis of data resolutions would be interesting future work. A larger population of path pairs (than the current 15) would be useful in order to further examine trends in this data. This analysis could be continued by examining two different occurrences of error. The first is when the two paths seem to be traveling along generally the same route, but due to the resolution difference, the paths do differ slightly. This can be identified by relatively small error magnitudes and some overlapping path points in both paths. The second type of error is when the two paths differ greatly, possibly due to some inflection point where the path algorithm was faced with two options for path directions and due to the caloric sums at this point being similar, one resolution chooses one path and the other resolution chooses the other, quite different path. This can be seen in *bottomhill2*. It would be interesting to do further analysis to try and identify when these two different types of path error occur.

One feature that could help users analyze and visualize paths better would be to show paths with a thickness or width. This width would represent the confidence of the path at that path point, which would be a measure of how much the path could deviate while still staying within a prescribed amount of caloric expenditure or

kilocalories. For example, if a path was traversing a valley and a path could traverse within a width of 10 meters of the path line while the final kilocalorie sum of the path would still be within some epsilon of error, then the path would be displayed with some proportional width in this valley region to indicate this 10m width.

BIBLIOGRAPHY

[1] Apache commons imaging: a pure-java image library, 2017. https://commons.apache.org/proper/commons-imaging/.

[2] Arcgis online. http://www.esri.com/software/arcgis/arcgisonline. Accessed: 2017-06-06.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[4] A. Duggan and M. Haisman. Prediction of the metabolic cost of walking with and without loads. *Ergonomics*, 35(4):417–426, 1992.

[5] Y. Epstein, L. Stroschein, and K. Pandolf. Predicting metabolic cost of running with and without backpack loads. *European journal of applied physiology and occupational physiology*, 56(5):495–500, 1987.

[6] ESA. Copernicus scientific data hub. https://scihub.copernicus.eu/dhus/#/home. Accessed: 2016-11-30.

[7] ESA. Science toolbox exploitation platform: Snap. http://step.esa.int/main/toolboxes/snap/. Accessed: 2016-11-30.

[8] ESA. Sentinel-1 sar overview. https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/overview. Accessed: 2017-05-31.

[9] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo, and J. Blat. 3d graphics on the web: A survey. *Computers & Graphics*, 41:43–61, 2014.

[10] A. Ferretti, K. Fletcher, A. spatiale européenne, E. S. Research, and T. Centre. *InSAR Principles: Guidelines for SAR Interferometry Processing and Interpretation*. ESA TM. ESA, 2007.

[11] Google earth. https://www.google.com/earth/. Accessed: 2017-05-28.

[12] Google maps. https://www.google.com/maps/about/. Accessed: 2017-06-06.

[13] Gpx: The gps exchange format. http://www.topografix.com/gpx_for_users.asp. Accessed: 2017-04-25.

[14] Gwt project, 2017. http://www.gwtproject.org/.

[15] J. A. Harris and F. G. Benedict. A biometric study of human basal metabolism. *Proceedings of the National Academy of Sciences*, 4(12):370–373, 1918.

[16] Hiking Science. Calculate personalized calories burned. http://hikingscience.blogspot.com/p/calculate-calories-burned_22.html. Accessed: 2017-05-25.

[17] Hudson Hikers. How many calories did you burn on your last hike? https://www.hudsonhikers.org/calories.html. Accessed: 2017-05-25.

[18] Khronos. Webgl specifications. http://www.khronos.org/registry/webgl/specs/latest/. Accessed: 2017-05-27.

[19] J. Knapik, K. Reynolds, and E. Harman. Soldier load carriage: historical, physiological, biomechanical, and medical aspects. *Military medicine*, 169(1):45, 2004.

[20] K. B. Pandolf, B. Givoni, and R. Goldman. Predicting energy expenditure with loads while standing or walking very slowly. Technical report, DTIC Document, 1976.

[21] Parallax. Parallax google web toolkit 3d library, 2017. http://parallax3d.org/.

[22] J. Rickwald. Continuous energetically optimal paths across large digital elevation data sets. Master's thesis, California Polytechnic State University, San Luis Obispo, 2007.

[23] Stanford Radar Interferometry Research Group. Snaphu: Statistical-cost, network-flow algorithm for phase unwrapping. https://web.stanford.edu/group/radar/softwareandlinks/sw/snaphu/. Accessed: 2016-12-01.

[24] A. Tsui. Energetic path finding across massive terrain data. Master's thesis, California Polytechnic State University, San Luis Obispo, 2009.

[25] A. Tsui and Z. Wood. Energetic path finding across massive terrain data. *Advances in Visual Computing*, pages 879–888, 2009.

[26] About web3d consortium. http://www.web3d.org/about. Accessed: 2017-05-27.

[27] D. White and S. Surface-Evans. *Least Cost Analysis of Social Landscapes: Archaeological Case Studies*. University of Utah Press, 2012.

[28] B. M. Wood. Energetic analyst : software for the visualization and analysis of pedestrian travel over three dimensional terrain, and its archaeological applications. Master's thesis, California Polytechnic State University, San Luis Obispo, 2004.

[29] B. M. Wood and Z. J. Wood. Energetically optimal travel across terrain: visualizations and a new metric of geographic distance with anthropological applications. In *Electronic Imaging 2006*, pages 60600F–60600F. International Society for Optics and Photonics, 2006.

[30] G. Zipf. *Human behavior and the principle of least effort: an introduction to human ecology.* Addison-Wesley Press, 1949.

APPENDIX

The following are Brian Wood's responses to questions seeking feedback from his use of the web application.

Please describe how effectively you are able to use this tool (rate on 1-5 scale, with 1 being not at all and 5 being very effectively)

My rating: 5, very effective. The interface is quite easy to use, and if someone has experience with Google Earth, then the interface takes 1 minute or less to learn. It is quite an advanced UI, I will note, given that it runs in a browser and not as a stand-alone application. In this regard, it shows impressive software engineerings skills, and utilization of some of the most advanced features of HTML5. I can imagine that a short, 5 minute YouTube video could accompany this tool, explaining each feature of the interface, and then users could easily start using all this application's features. Even without such a video, a user could figure out what is going on without much trouble just because the panes, the buttons, the tables, and the labels are all intuitively organized and named.

Please describe how it was using this tool

I used this tool by first navigating in the 3D map area to get a sense of the overall area that is loaded into the system (NE Papua New Guinea). I then opened Google Earth and selected two villages that appeared to be in this area, but were separated by many kilometers and a great deal of terrain. I wrote down the lat and lon values for these villages. I then pushed the "Add Path" button in the Caloric Path tool. I entered the name of the path, the starting coordinates, and the end coordinates. This all went very smoothly. I then selected the Wood&Wood algorithm and hit compute. The

computation procedure was immediately listed as having commenced, and it gave me a readout of its progress. This progress read-out is much improved from the first iteration of the tool that I had seen. It was accurate, and quite fast, given the length of the path. The computation of the path was then finished, and the path displayed on the 3D map automatically. I then zoomed in and out of the 3D view in order to see where the path traveled. Navigating in the 3D window was very smooth – and again, much improved over the first version I had used. After looking at the path and looking at the resulting caloric value of the path, I then hit the "export selected" button. This exported a GPX file with the name of the path as the title of the file. I then opened this file in Google Earth, making sure to select "Make LineStrings" option. After viewing the route in Google Earth, I scanned the path that the route took in greater detail, and noted that the computed path seemed to be much more intelligent than would a 'as the crow flies' calculation – this was satisfying. The route that was calculated follows several river valleys and it avoids moving into the highlands, just as one would hope. After doing this calculation, I then clicked on the "information" button in the top of the 3D interface. This allowed me to see some of the info regarding the different cost algorithms. This was useful. I was then finished using the tool.

Please describe how you would use this tool for your research

I would use this tool in order to: 1) compare real routes of travel to predicted routes of travel, under different cost-function assumptions; 2) to 'retrodict' where people might have travelled in the past, under different cost function assumptions; 3) to compute a general measure of geographic distance which is appropriate to pedestrian travel. This would be used

to predict the flow of objects through trade, ideas or linguistic markers through social contact, or genes through gene-flow in spatially structured populations. There are many potential anthropological applications!