ENERGETIC PATH FINDING ACROSS MASSIVE TERRAIN 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

Andrew Tsui

June 2009

COMMITTEE MEMBERSHIP

TITLE:	Energetic Path Finding Across Massive Terrain Data
AUTHOR:	Andrew Tsui
DATE SUBMITTED:	June 2009
COMMITTEE CHAIR:	Zoë Wood, Ph.D.
COMMITTEE MEMBER:	Chris Clark, Ph.D.
COMMITTEE MEMBER:	Chris Buckalew, Ph.D.

Abstract

Energetic Path Finding Across Massive Terrain Data

Andrew Tsui

Before there were airplanes, cars, trains, boats, or bicycles, the primary means of transportation was on foot. Unfortunately, many of the trails used by ancient travelers have long since been abandoned. We present a software tool which can help visualize and predict where these forgotten trails might lie through the use of a human-centered cost metric. By comparing the paths generated by our software with known historical trails, we demonstrate how the tool can indicate likely trails used by ancient travelers. In addition, this new tool provides novel visualizations to better help the user understand alternate paths, effect of terrain, and nearby areas of interest. Such a tool could be used by archaeologists and historians to better visualize and understand the terrain and paths around sites of historical interest.

This thesis is a continuation of previous work, with emphasis on the ability to generate paths which traverse several thousand kilometers. To accomplish this, various graph simplification and path approximation algorithms are explored to construct a real-time path finding algorithm. To this end, we show that it is possible to restrict the search space for a path finding algorithm while not disrupting accuracy. Combined with a multi-threaded variant of Dijkstra's shortest path algorithm, we present a tool capable of traversing the contiguous US, a dataset containing over 19 billion datapoints, in under three hours on a 2.5 Ghz dual core processor. The tool is demonstrated on several examples which show the potential archaeological and historical applicability, and provide avenues for future improvements.

Acknowledgements

An abundance of thanks to Zoë Wood for her constant encouragement, contagious enthusiasm, and endless stream of ideas.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Before there were airplanes, cars, trains, boats, or bicycles, the primary means of transportation was on foot. Anthropologists, archaeologists, and historians have spent a great deal of time uncovering the routes taken by ancient travelers as they traversed between their villages, between trade-routes with long-distance neighbors, or as they foraged for food around their camp [21]. As a general rule of thumb, humans, through trial and error, are often quite proficient in finding the most efficient path of travel. However, some of these paths traverse great distances, making it near impossible to verify the actual efficiency.

This thesis is a continuation of previous work [45, 35] which was concerned with energetic paths (paths of least caloric cost when traveling on foot) across large terrain data sets. We provide tools and algorithms that can work with even larger data sets, for example the contiguous US, and provide an interactive 3D perspective viewing application. When working with such a large data set, simplification and approximation are necessary. This work explores various graph simplification and path approximation algorithms in order to create a real-time path finding algorithm for massive data. Furthermore, this work provides a more

efficient means of generating paths by restricting the search to a subset of the original data. We validate this approach by comparing the paths generated in this manner with paths generated on the full, unrestricted data. In addition, visualization techniques to compare potential paths, difficult terrain, and alternate destinations are explored.

The tools are written in C++ using the OpenGL graphics API and Berkeley Database. Satellite imagery is utilized to provide a 93,600 by 212,400 elevation and landcover data grid covering the contiguous US. The application is interactive and allows the user to select start and end locations, as well as one of several path computation algorithms. The available algorithms are Dijkstra's, Fast Dijkstra's (a multi-threaded variant of Dijkstra's introduced in this thesis), A*, and Single-Query Single Direction PRM (a Probabilistic Road Map algorithm). Most computations are divided into a global and detailed search phase. The global search phase identifies a rough path using a simplified dataset. This rough path is then used to significantly reduce the total memory and computational time required for the detailed search phase, which computes the exact path on the full dataset, by ensuring that only relevant areas of the terrain are searched.

Our results show that path computation over massive out-of-core datasets is possible. We conclude that using Dijkstra's algorithm (or the Fast Dijkstra variant) provides the best results in terms of accuracy, but can require additional memory. However, alternate methods such as Probabilistic Road Maps can be used to great effect in certain scenarios (for example, when generating very short paths).

1.1 Contributions

The contributions of this thesis are listed below:

- Tools for managing and performing energetic analysis on massive out-of-core data sets are presented. In particular, a restrictive tiling scheme is constructed which significantly reduces the search space without reducing accuracy.
- Energetic equations that account for terrain type, specifically the presence of water, are introduced.
- A comparison between multiple path computation algorithms in terms of runtime, memory usage, and accuracy is conducted.
- A multi-threaded bidirectional version of Dijkstra's shortest path algorithm that does not suffer any accuracy loss is presented.
- An interactive 3-D perspective viewing application is provided.
- Visualizations that allow the comparison of similar paths are introduced.

Chapter 2

Previous Work

This section contains descriptions of the previous work in this area.

2.1 Path Finding and Data Management

There has been significant work in the area of path finding algorithms. In general, these algorithms transform the data into a graph structure and iterate over the data starting at a specific node in the graph and making connections to other nodes in an expanding fashion until the destination is found. Dijkstra's [8] and Fast Marching [22] are two examples of a path finding algorithms that find optimal paths. A* [16] and Probabilistic Road Maps [19] are examples of path finding algorithms which may sacrifice accuracy for speed. The algorithms used in this paper are covered in detail in Chapter 3.

The Berkeley Database [32] is a mature, non-relational database developed at the University of California at Berkeley. It stores information as arbitrary key/value pairs, making it useful for applications which utilize large groups of

data, but are not concerned with creating relations between individual data. The Berkeley DB is covered in detail in Section 4.1.1.

2.2 Energetic Analyst

The Energetic Analyst tool developed by Brian Wood [45, 46] was designed to aid archeologist's in finding energetically optimal paths. Wood's work demonstrated the importance of using a human-centered, as opposed to distance-centered, metric for determining the routes of travel. Energetic Analyst contained visualization tools to display terrain described by digital elevation models (DEM's), as well as the path of least caloric cost between two points on the terrain found using Dijkstra's shortest path algorithm.

Energetic Analyst was constrained in several ways which made the analysis of large terrain datasets infeasible. All the data required for the path computation was stored in memory, thus severely limiting the size of the DEM that could be analyzed. In conjunction with this, only the ArcInfo ASCII DEM format was supported. While this format is simple and easy to parse, it consumes a large amount of space on disk and in memory, thus further limiting the potential for analyzing large DEM's.

2.3 Continuous Energetically Optimal Paths

Jason Rickwald continued Brian Wood's work by providing tools to compute continuously energetic optimal paths (CEP) across larger terrain datasets [35]. Rickwald utilized the Fast Marching Algorithm in his path computations. In essence, the Fast Marching Algorithm tracks an expanding wavefront as it moves

across a surface. For terrain data, this has the benefit of allowing paths to cross a grid face, rather than being constrained to the grid edges. In addition, Rickwald introduced a multi-threaded variant of Fast Marching, which significantly reduced the runtime but introduced a small error in the energetic path computation.

Rickwald addressed the memory limitations encountered by Wood by providing a mechanism for swapping terrain data between memory and disk. However, the same ArcInfo ASCII DEM format is used, again hindering the capability of analyzing very large datasets. For comparison a 3600x3600 ASCII DEM grid requires roughly 77.3 megabytes of disk space, whereas a binary HGT format containing the same information requires only 24.7 megabytes. In addition, finding paths between two widely spaced points using these tools required a significant amount of time. Rickwald tested his implementation on a dataset covering most of the state of Oregon and noted that a path computation across the state required most of the day.

Chapter 3

Algorithms

This section details the algorithms used in analyzing the terrain data.

3.1 Caloric Cost

The equations used in this work to determine the caloric cost of travel between two points are the same as for Energetic Analysis and CEP. These equations were determined and verified under various conditions [9, 11, 23, 33]. First, the metabolic rate for traveling between two points is calculated based on the physical parameters of the subject and the slope (grade) of travel. Equation (3.1) shows rate when traveling on a positive grade (uphill), while Equation (3.2) is for a negative grade:

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

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

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

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

Where:

MR	is the metabolic rate in watts
w	is the person's weight in kilograms
l	is the load carried in kilograms
v	is the velocity in meters per second
g	is the percent grade
n	is the terrain factor

Terrain factors are given by [23]:

1.0	Black Top Road / Treadmill
1.1	Dirt Road
1.2	Light Brush
1.5	Heavy Brush
1.8	Swampy Bog
2.1	Loose Sand
1.3+0.082*D	Snow, where D = depression depth in cm

For this work, the average velocity when traveling across open ground is $1.34112\frac{m}{s}$, or approximately $4.8\frac{km}{hr}$. When traversing water, it has been experimentally determined that swimming at $0.7\frac{m}{s}$ is roughly equivalent to running at $3.3\frac{m}{s}$ [34, 41].

Unfortunately, the above equations can potentially under-predict the caloric cost when traveling downhill at certain velocities. Thus, the above metabolic rate is compared to the metabolic rate while standing [15](SMR), with the larger being used to complete the calculation:

$$MR_{downhill} = max\{MR_{downhill_{3.2}}, SMR\} \tag{3.5}$$

$$SMR = 1.2 * BMR \tag{3.6}$$

$$BMR_{male} = 66 + (13.7 * w) + (5 * h) - (6.8 * a) \tag{3.7}$$

$$BMR_{female} = 655 + (9.6 * w) + (1.7 * h) - (4.7 * a) \tag{3.8}$$

Where:

MR	is the metabolic rate in watts
SMR	is the standing metabolic rate in watts
BMR	is the basal metabolic rate in watts
w	is the person's weight in kilograms
h	is the person's height in centimeters
a	is the person's age in years

The metabolic rate gives the amount of energy expended over time, thus it is necessary to obtain the approximate time required to travel between the two points:

$$T = \frac{dist_{total}}{v} \tag{3.9}$$

$$dist_{total} = \sqrt{dist_{lat}^2 + dist_{long}^2} \tag{3.10}$$

$$dist_{lat} = n|a_{lat} - b_{lat}| \tag{3.11}$$

$$dist_{long} = n * cos(a_{lat}) * |a_{long} - b_{long}| \tag{3.12}$$

Where:

a	is the starting location in latitude/longitude
b	is the destination in latitude/longitude
v	is the velocity in meters per second
T	is the time to travel between **a** and **b** in seconds

Finally, the caloric cost (CC) is calculated and converted to kilocalories using Equation (3.13):

$$CC = \frac{MR * T}{4184 \frac{J}{kilocalorie}} \tag{3.13}$$

3.2 Dijkstra and A^*

Dijkstra's shortest path algorithm [8] is a well known and extremely pervasive algorithm for determining paths of least cost between two points on a graph. In general, the algorithm uses a priority queue to expand the search in a wavefront propagation until the destination is found.

A* [16] is a slight modification of Dijkstra's algorithm. This algorithm utilizes an admissible heuristic function to estimate the cost from the current node to the destination node[1]. Thus, an appropriate heuristic can cause the algorithm to converge to the destination node in much shorter time than Dijkstra's by estimating the cost to reach the destination for each node and ignoring nodes of higher cost. For this work, the heuristic uses the Euclidean distance from the current node to the destination, combined with the metabolic rate associated with the corresponding grade, to estimate the total caloric cost. Unfortunately, this heuristic is not admissible in certain rare cases, which introduces a small error into the path computation. Thus, the paths provided using A* are not necessarily the most energetic.

3.3 Fast Dijkstra

There has been significant recent work on developing parallelized cost approximation algorithms [35, 42, 39] to utilize the increasing number of cores within standard processors, as well as new GPGPU architectures which are capable of running potentially thousands of threads simultaneously [14, 29]. However, the general problem with these algorithms is that they provide approximations of

[1] Admissible indicates that the heuristic must not overestimate the cost to the destination

optimal cost paths, thus sacrificing accuracy for speed.

This work presents a bidirectional implementation of Dijkstra's shortest path algorithm which uses two threads to capitalize on modern multi-core processors. This algorithm does not disrupt the optimality properties of Dijkstra's, thus providing optimal paths with a minimal amount of memory overhead. In essence, Dijkstra's algorithm is run separately in two threads with one thread calculating the cost from the start node to the destination node, while the second thread simultaneously calculates the cost from the destination to the start. The two threads meet roughly half-way to their respective goals where one thread is given priority and is responsible for combining the results of the two threads. Algorithms 1 and 2 contain the pseudo code for the 2-threaded version of Dijkstra's, called *Fast Dijkstra*. The initialization code has been removed for brevity while `FORWARD` and `BACKWARD` denote the two threads. A *tainted* node indicates that the `FORWARD` thread has visited, but not necessarily found an optimal path to, this node.

Algorithm 1 is very similar to the traditional Dijkstra's algorithm, with the exception of lines 5, 8, and 13. Line 5 indicates a new termination condition. It is assumed that the `FORWARD` thread will take precedence, and is therefore responsible for terminating the algorithm. However, the `FORWARD` thread should never encounter the destination node since the `BACKWARD` thread should have been computing paths from the destination. Thus, it is sufficient to check if this is the `FORWARD` thread and that it is not contained in `BACKWARD`'s set of unvisited nodes (i.e. `BACKWARD` has found a shortest path from the destination to this node). These two conditions imply that a shortest path has been found by `FORWARD` from the start node to this node, and a shortest path has been found by `BACKWARD` from the destination node to this node. This is explained in greater detail during the

Algorithm 1 The Fast Dijkstra Algorithm

Input: *source*, *target*

Output: Path from *source* to *target*

1: $Q_f, Q_b \leftarrow$ All nodes in graph
2: $T \leftarrow \emptyset$ {Set of tainted nodes}
3: **while** Q_d is not empty **do** {Q_d is the set for the direction}
4: $u \leftarrow$ vertex in Q_d with smallest cost
5: **if** FORWARD && $u \notin Q_b$ **then**
6: Break {Least cost path found}
7: **end if**
8: **if** BACKWARD && $u \in T$ **then**
9: Continue {FORWARD results take precedence}
10: **end if**
11: Remove u from Q_d
12: **for all** neighbors v of u **do** {Where $v \in Q_d$}
13: **if** BACKWARD && $v \in T$ **then**
14: Continue {FORWARD results take precedence}
15: **end if**
16: $UPDATE(u, v)$
17: **end for**
18: **end while**
19: $path \leftarrow path(source, u) + path(u, target)$
20: **return** *path*

discussion of Algorithm 2 below.

Lines 8 and 13 of Algorithm 1 indicate that the BACKWARD thread should not continue to analyze any nodes that the FORWARD thread has begun analyzing. This effectively means that the FORWARD thread overwrites all of BACKWARD's unfinished[2] computations.

If the two threads have not yet encountered any nodes visited by the other thread, then the update step described in Algorithm 2 proceeds as it normally would in the standard Dijkstra's algorithm. It should be noted that lines 2 and 15 differ only in the order in which the *cost* function analyzes the two nodes. This is to account for bidirectional graphs, which is important in this work as the caloric cost of traveling uphill differs from traveling downhill as discussed in Section 3.1. The reversed order means that BACKWARD actually calculates paths from a node to the destination, not from the destination to the node (as this would create incorrect paths as all uphill travel would be calculated as downhill and vice versa).

Lines 3 through 13 handle the case when the two threads begin to analyze nodes seen by the other thread (i.e. where the search space of the two threads overlap). Line 3 indicates that special consideration is only taken if BACKWARD has found a shortest path to node v from the destination. If BACKWARD has updated the cost to reach v, but not yet found a shortest path to it, then FORWARD overrides any of those results (lines 11 and 13). However, if BACKWARD has found a shortest path to v, then we want save those results (lines 5 and 6). Hereafter, the cost to reach v is augmented with the cost to reach the destination from v as determined by the BACKWARD thread. Essentially, this says that if BACKWARD

[2]Unfinished means that the thread may have assigned a cost to reach a node, but has not declared that the shortest path to that node has been found.

Algorithm 2 The Fast Dijkstra Algorithm Update Step

Input: u {node to expand from}, v {node to expand to}

Output: Updated path from u to v

if FORWARD **then**
  $alt \leftarrow cost[u] + cost(u, v)$
  if $v \notin Q_b$ **then** {If BACKWARD has found a shortest path}
    if $v \notin T$ **then** {First time FORWARD has seen v}
      $sPrev[v] \leftarrow prev[v]$ {Save the previous from BACKWARD}
      $sCost[v] \leftarrow cost[v]$ {Save the cost from BACKWARD}
    end if
    $alt \leftarrow alt + sCost[v]$ {Cost is to get to the target}
  end if
  if $v \notin T$ **then**
    $cost[v] \leftarrow \infty$ {Force the relaxation}
  end if
  $T \leftarrow T \cup v$ {Taint v}
else {This is BACKWARD}
  $alt \leftarrow cost[u] + cost(v, u)$ {Bidirectional graph}
end if
if $alt < cost[v]$ **then** {$Relax(u, v)$}
  $cost[v] \leftarrow alt$
  $prev[v] \leftarrow u$
end if
return

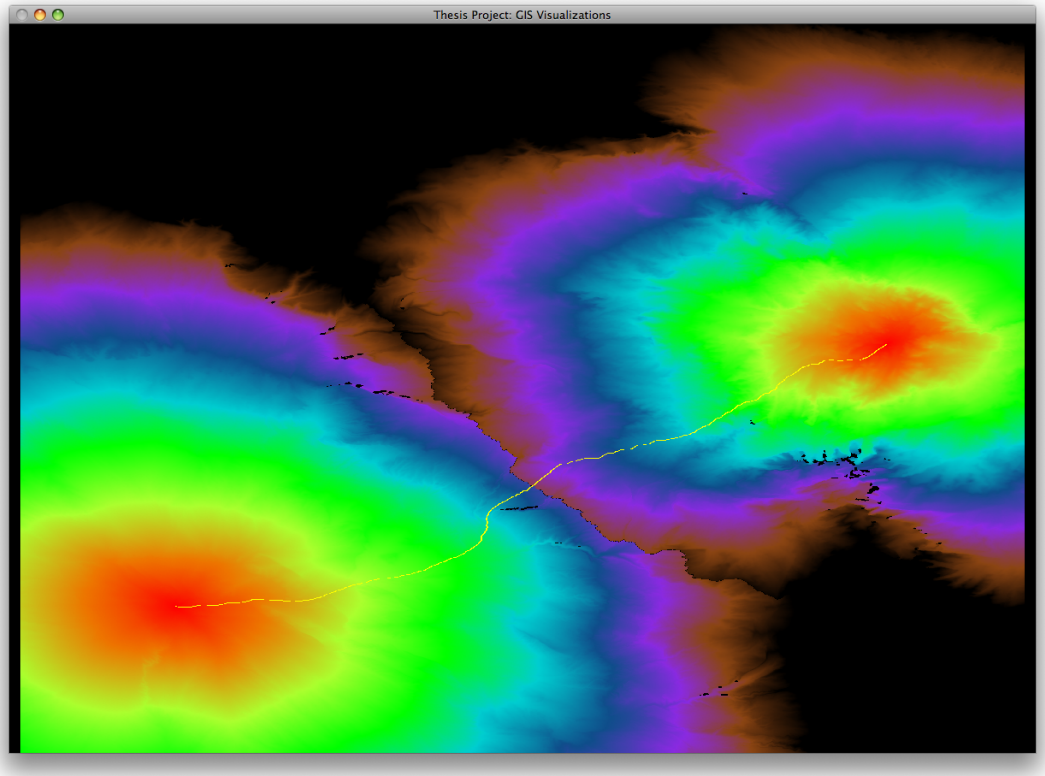


Figure 3.1: This shows the areas searched by each thread during a run of the Fast Dijkstra algorithm. Notice that one thread gains priority when the search areas overlap each other.

has found a shortest path from v to the destination, then once `FORWARD` finds the shortest path from the start node to v, the optimal substructure property of Dijkstra's algorithm tells us we have found the shortest path from the start to the destination that goes through v. Therefore, line 5 in Algorithm 1 terminates the algorithm when the node that offers the 'best' connection between the paths found by both threads is found.

3.4 Probabilistic Road Maps

A Probabilistic Road Map (PRM) [19, 18, 25, 26] is defined as being a discrete representation of a continuous configuration space generated by randomly sampling the free configurations of a search space and connecting those points in a graph. Essentially, the algorithm randomly samples the search space until a path can be constructed from the start to the destination. These algorithms are designed for speed at the cost of accuracy. However, due to the probabilistic nature of the algorithms, it is possible to randomly produce an optimal path in a fraction of the time of other algorithms.

PRM's have had success in finding paths in spaces that are difficult to model, or have many degrees of freedom. The later point is the reason for the application of PRM's to the task of finding energetic paths across massive data. At any given point, a path can diverge in eight different directions (see Figure 4.1). Using PRM's we hope to eliminate the need to search many of these directions, thus greatly decreasing the necessary computation time.

The specific PRM algorithm used in this work is the Single-Query Single Directional PRM (SQPRM) [19]. The idea is to grow a tree type path in random directions until the destination is found. However, since SQPRM's expand in a

random fashion, it may require a large amount of time to randomly select and connect the destination node to the graph. Thus, the algorithm terminates when a node is examined that is in the *endgame region*, meaning that it is sufficiently close to the destination node. The pseudo code for the basic algorithm is given in Algorithm 3 with an illustrative example of how SQPRM would be run on a DEM grid shown in Figure 3.2.

Node selection (line 3) is important to ensure that a large portion of the search space is examined in a short amount of time. A poor choice for node selection, such as giving all nodes an equal probability of being chosen, often results in the searched nodes being closely packed together instead of spread throughout the search space. This is termed *clustering*. To alleviate this problem, candidate nodes (those in T) can be weighted with their node density (the percentage of neighbors that are also in T). This allows us to pick nodes with a probability proportional to the inverse of node density, thus forcing faster coverage of the search space. Figure 3.3 shows the effect of clustering and the results after a weighting scheme is applied.

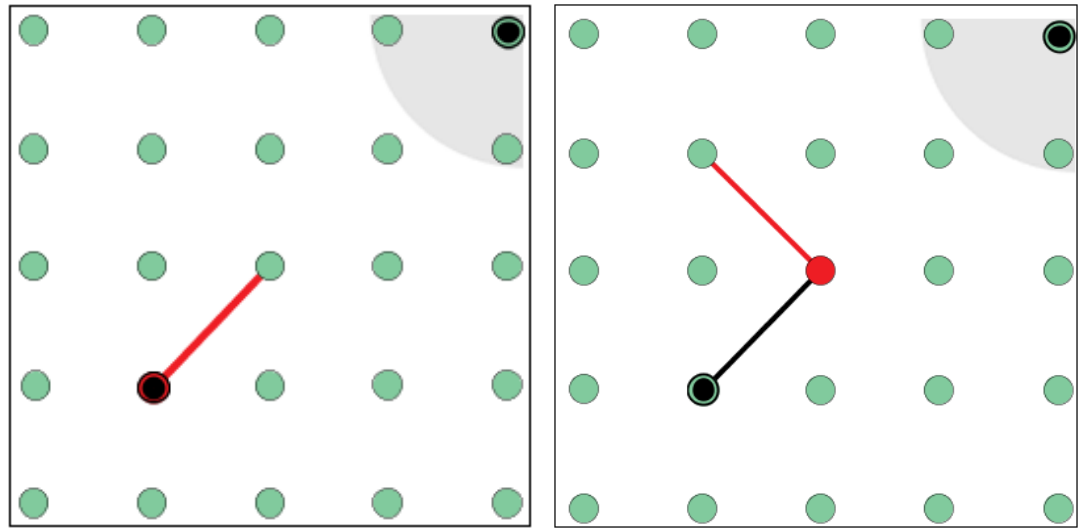
An efficient and highly effective method of node weighting is given in [20]. First, the entire search space is discretized into *cells*. A *cell* is considered *occupied* if there exists a node within the cell that is part of the current tree. Thus, node selection occurs by randomly selecting from the the set of *occupied* cells, then randomly selecting a node within that cell. Figure 3.4 illustrates this method.

Algorithm 3 Single-Query Single Directional PRM

Input: *start*, *endgame region*

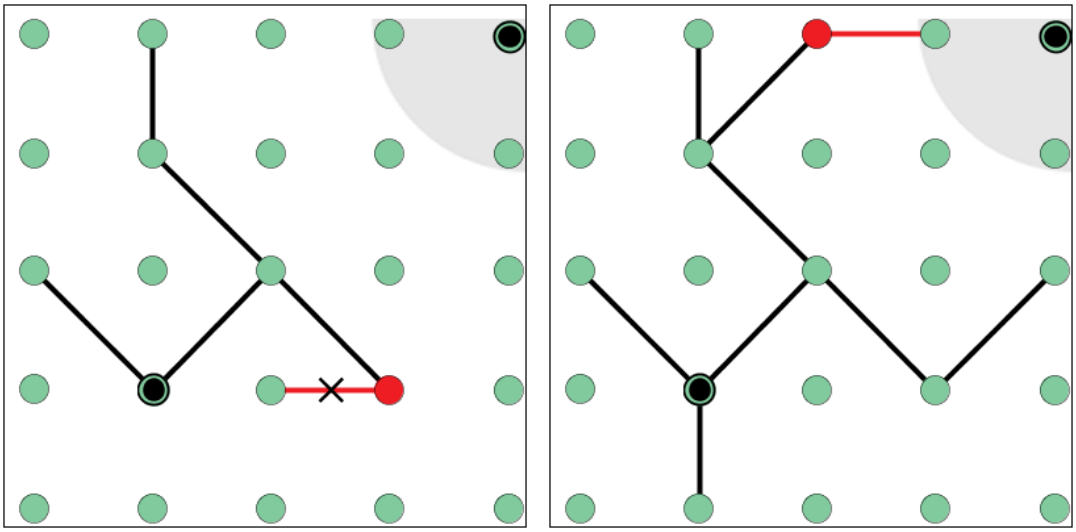
Output: Path from *start* to *endgame region*

1: Insert the start node into T
2: **loop**
3: Pick a node n from T with probability $\pi_T(n)$
4: Generate a new node n' near n
5: **if** n' is reachable from n **then**
6: Add n' to T
7: **if** $n' \in$ endgame region **then**
8: **return** path
9: **end if**
10: **end if**
11: **if** Stopping criteria is met **then**
12: **return**
13: **end if**
14: **end loop**



(a) Iteration 1

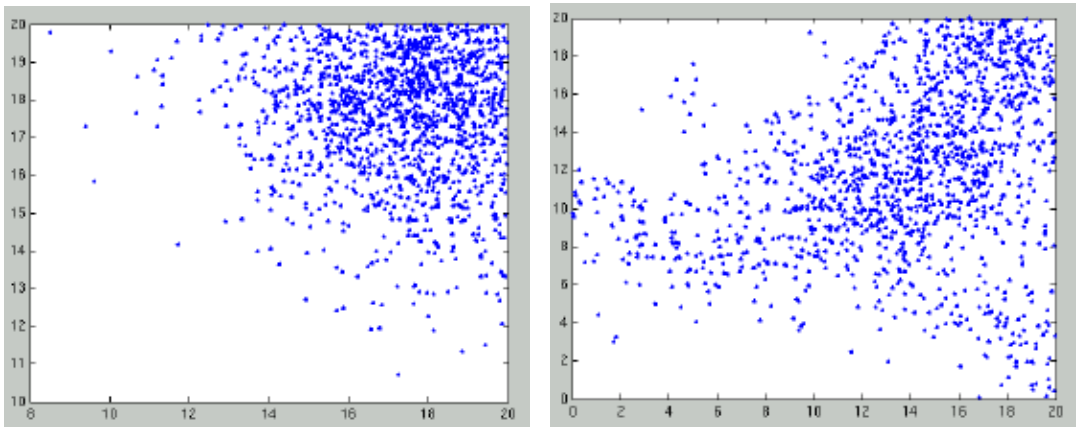
(b) Iteration 2



(c) Iteration 6

(d) Iteration 10

Figure 3.2: An example showing the Single-Query Single Directional PRM algorithm at various iterations. The double circles represent the start and end nodes, the grey area represents the endgame region, and the red node and line indicate the selected node and expansion for the iteration. In (c), the X indicates that the edge is rejected based on some criteria (step 5 in Algorithm 3).



(a) Not Weighted (b) Weighted

Figure 3.3: The impact of node selection for PRM's. The figures show the searched nodes after a given amount of time. (a) shows the effect of clustering while (b) shows the effect of a weighting scheme. Image courtesy of [6].

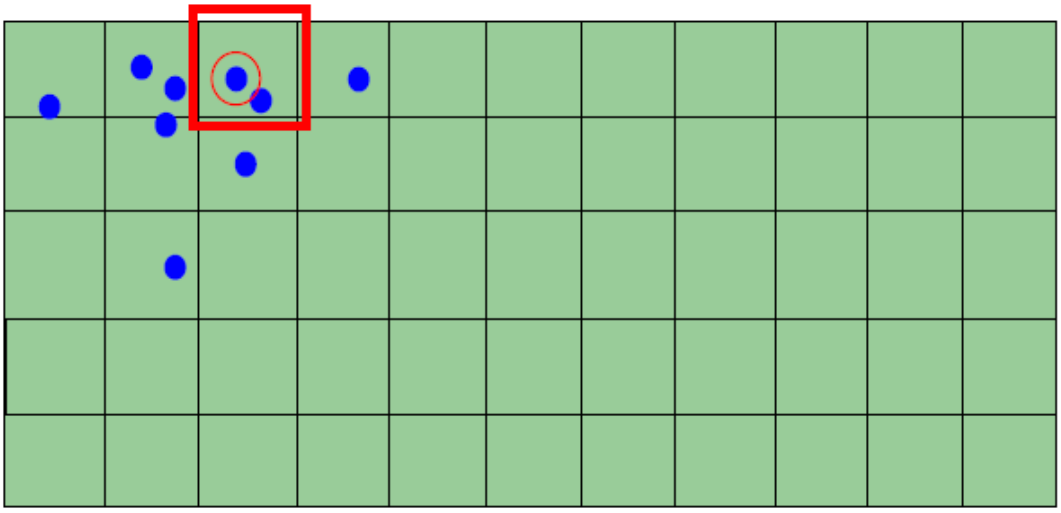


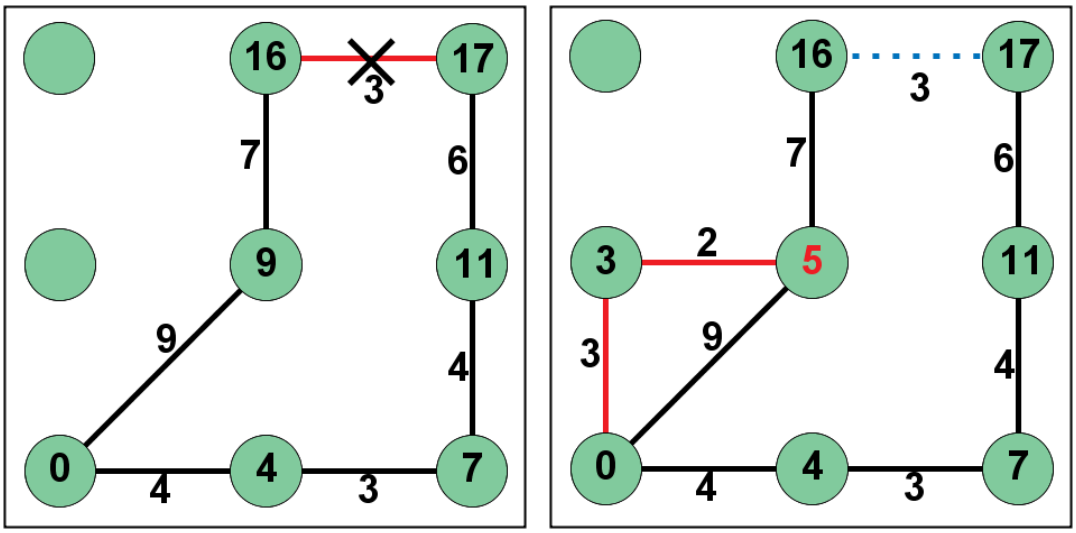
Figure 3.4: The Kindel Method for selecting a node to expand. The blue dots indicate candidate nodes while the red outline indicates the cell and node that are selected. Image courtesy of [6].

3.4.1 Difficulties

The PRM class of algorithms are designed to quickly construct a traversable path in a large search space, but are not concerned with the actual efficiency of the path. Thus, if a potential edge is not accepted (step 5 in Algorithm 3), it can be assumed that that edge will never be added to the graph. However, in the context of energetic paths where a criteria for acceptance may be the total cost of the path up to that point, it is possible that a previously rejected edge may become viable at a later iteration. This situation is detailed in Figure 3.5 and shows that an edge may be erroneously rejected due to out-dated information.

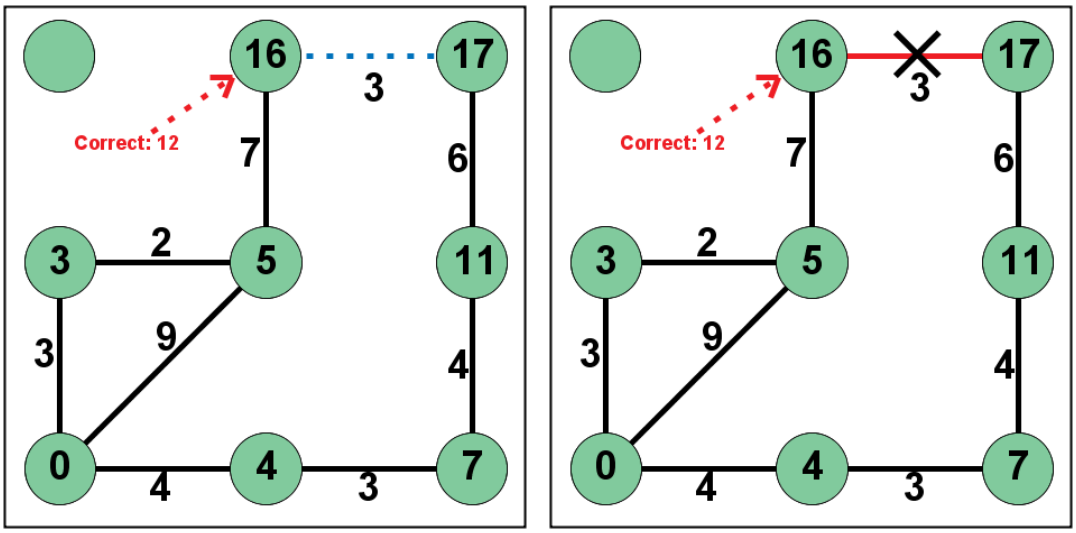
This problem can be alleviated by propagating an updated cost to all related (connected through a sequence of edges) nodes along the path whenever a node is updated. Thus, if a previously rejected edge is reconsidered, the most recent information will be available. However, due to the size of the search space, and the potentially large number of connections related to a given node, experimentation with this method has shown a considerable decrease in efficiency.

Therefore, the approach taken in this work is to allow a node to be selected and expanded multiple (possibly unlimited) times, with the cost to reach directly connected (as opposed to all related) nodes being updated when appropriate. Thus, any change to a nodes cost may be slowly propagated as the algorithm progresses. While this does not completely eliminate out-dated information from being used to make decisions, it does provide a means to reevaluate certain edges and allows the algorithm to be applied to problems where path efficiency is of concern. However, the number of times the node is allowed to be updated can significantly impact the efficiency of the algorithm. This is shown in detail in Chapter 5.



(a) Edge Rejected

(b) Additional Edges Reduce Cost



(c) Related Nodes Not Updated

(d) Edge Erroneously Rejected

Figure 3.5: An example showing the difficulty in applying the Single-Query Single Directional PRM algorithm to energetic path finding. The number inside the node indicates the current cost to reach that node, while the numbers on the edges are the cost of traversing that edge. In (a), an edge is rejected based on the cost being greater than a previous path. After several iterations, (b) shows new paths which reduce the cost of reaching another node. However, the cost is not propagated to all connected nodes, as shown in (c). If selected again, the originally rejected edge should be accepted based on the updated costs, but is again rejected since the cost was not propagated, as shown in (d).

Chapter 4

Implementation

This section contains the implementation details for this work.

4.1 Data Sources

Graphical Information Systems (GIS) is a general term applied to software that captures and manages data related to geographical locations. This work utilizes two types of GIS data, Digital Elevation Models (DEM's) and Landcover data.

DEM's describe the elevation of a given terrain in a regularly interspersed grid pattern in various resolutions and formats. For any particular DEM, there are a variety of ways in which that data may have been collected which may affect their accuracy and completeness. These techniques include photogrammetric mapping [30], Light Detection and Ranging (LIDAR) [4], and Interferometric Synthetic Aperture Radar (InSAR) [3]. DEM data is available from a wide variety of sources including the US Geological Survey [40], Lakes Environmental [24], and

various other online sources [43, 28].

The data used for this work was obtained during the Shuttle Radar Topography Mission (SRTM) [13]. This mission used an InSAR array on-board the Space Shuttle Endeavor during an 11 day mission in February of 2000. Using data from this mission provides several benefits. First, elevation data was collected on a global scale, providing a consistent data source and format allowing seamless integration of large areas. In addition, several passes were made over most areas, allowing for corrections and higher resolution results. Finally, the data has been rigorously validated for accuracy [1]. The obtained data comes in a binary HGT format at 30 meter resolution (1 arcsecond) and is available through FTP [38]. The area selected for this work covers the entirety of the contiguous US and contains 19,880,640,000 data points. For the remainder of this paper, this data is referred to as the *full dataset*. Figure 4.1 illustrates how a DEM grid is converted to a graph for use in the path finding algorithms.

In addition to elevation data, this work incorporates landcover data from the National Land-Cover Database (NLCD) created by the Multi-Resolution Land Characteristics Consortium [17, 27]. This dataset provides land cover classification into 16 different categories (i.e. open water, forested, and wetlands) in the same resolution as the SRTM data.

Finally, cartographic boundary coordinates were obtained from the U.S. Census Bureau [2].

4.1.1 Data Management

The Berkeley Database [32] was chosen in order to facilitate the management of the acquired data, as well as any data generated during the path computations.

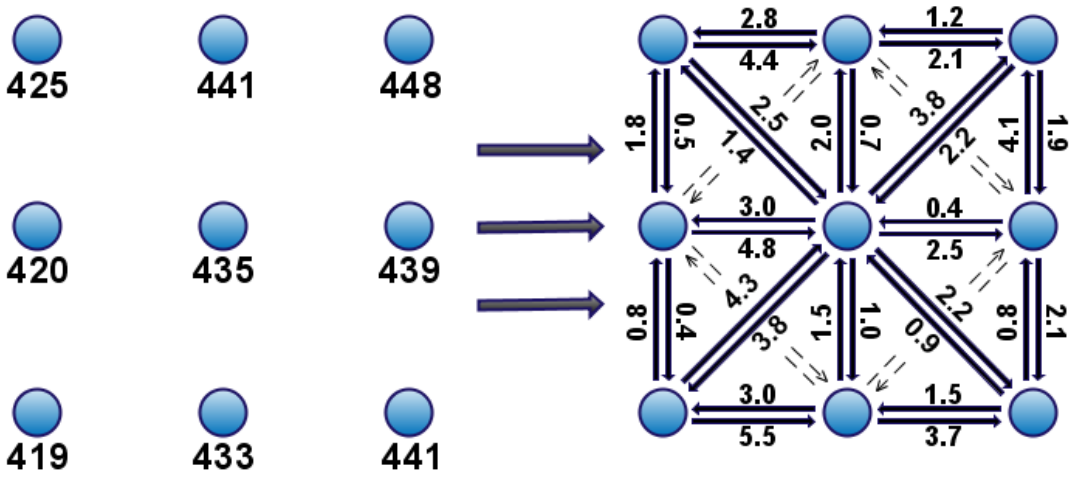


Figure 4.1: A DEM grid (left) is converted into a weighted graph (right) for use in the path algorithms. The numbers on the left indicate elevations, whereas the numbers on the right are caloric costs as determined in Section 3.1

The Berkeley DB is not a relational database, but rather a transactional database that stores binary data chunks using simple key/value pairs. It creates a local DB, thus avoiding the performance overhead of client/server database solutions.

The `gisDB` tool performs the actions necessary to load all the required data into a Berkeley DB. This includes both the terrain and landcover datasets. In addition, `gisDB` functions as a wrapper to the Berkeley DB API, allowing other tools to access the data with ease. In order to expedite the transfer of information between memory and disk, all information is stored in a compact binary format. Terrain and landcover data require 3 bytes total per data point, while data necessary for path computation require 5 bytes per data point (4 bytes for the cost to reach the node and 1 byte for various flags). To save space, the path computation information is only stored during a particular run, and erased as soon as it is no longer needed. The terrain and landcover data is stored in groups of nearest data points, as opposed to sequential order. This allows a certain degree of prefetching

since if one node is requested, it is likely that nodes near it will soon be required.

4.2 Simplification

A goal of this thesis was to provide a 3-D perspective viewing application capable of rendering a representation of the entire dataset. As the dataset comprises over 19 billion data points, it would be impossible to display the exact data in a real-time fashion. Thus, it was necessary to develop tools to provide a highly simplified, yet acceptably accurate, representation of the dataset.

This is accomplished through the `gisSimp` tool. The dataset was first broken down into 434*1,000 non-overlapping rectangular *clusters*[1]. The tool then utilizes `gisDB` to individually load each *cluster* and performs a simplification on those data points to obtain a single representative data point. For the terrain data, two simplified datasets were created; one which represents the average of all the points in the *cluster*, and one which represents the median data point. For the landcover data, the simplified dataset represents the classification that appears most often in each *cluster*.

4.3 Interaction

The `gisProj` tool is used as the entry point into the main application. It is responsible for displaying the simplified datasets generated by `gisSimp` using OpenGL [31], as well as handling all user interaction. Figure 4.2 shows the simplified terrain dataset generated by `gisSimp` and rendered by `gisProj`.

[1]These dimensions were chosen as they maintain the approximate latitude to longitude ratio across the US.

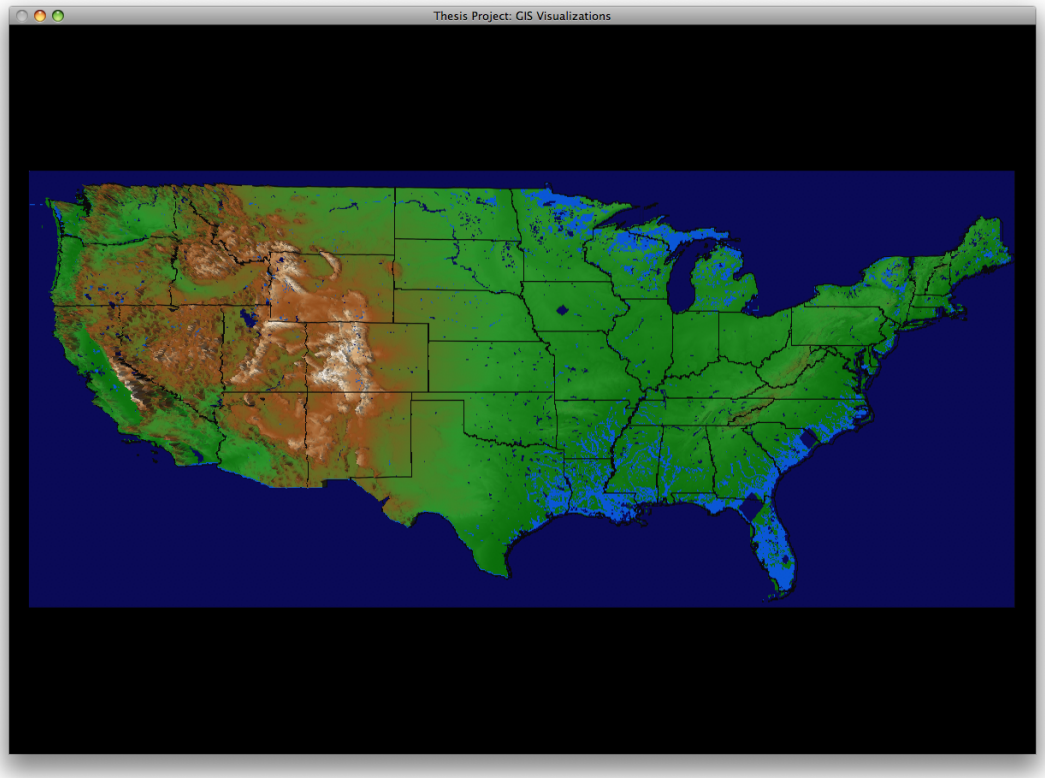
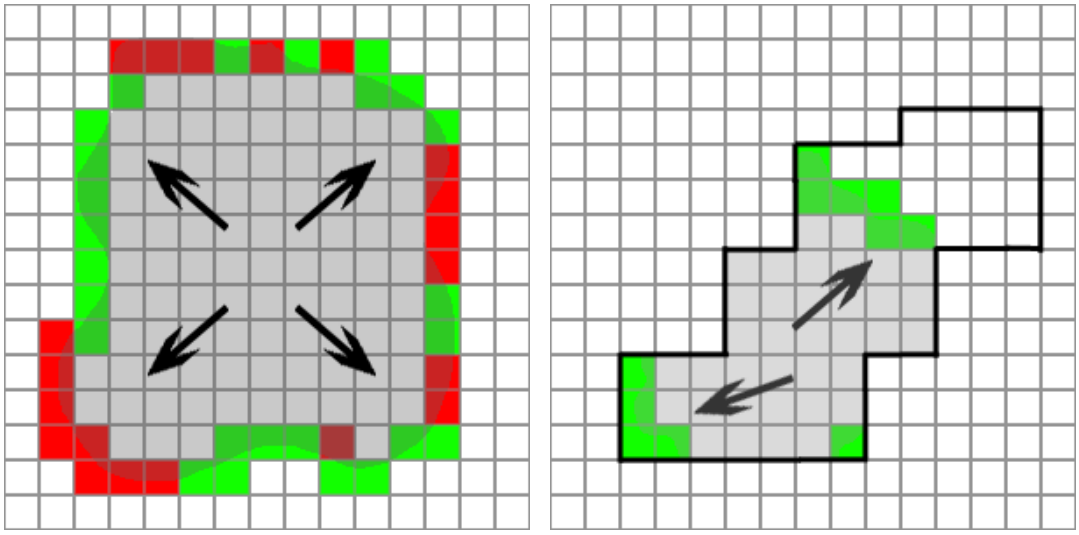


Figure 4.2: Display of the simplified terrain dataset. The color of the terrain is based on its elevation, with the exception of areas of water (blue) and wetlands (light blue).

The `gisMesh` and `gisTileManager` tools are designed to handle the analysis of the simplified and full datasets respectively. `gisMesh` takes as input the simplified dataset generated by `gisSimp` and constructs an internal graph representation of the dataset. Path computation occurs when the user interacts with `gisProj` to select two arbitrary points on the displayed terrain. The latitude and longitude of these two points is then passed to `gisMesh`, which performs a variant of Dijkstra's shortest path algorithm that is optimized for the particular graph structure. The resulting path serves as an approximation to the actual path, and is passed back to `gisProj` and overlayed onto the simplified terrain.

The `gisTileManager` tool is used to perform path analysis on the full dataset. Since the dataset is too large to fit in memory, `gisDB` is utilized to swap portions of the dataset that are not currently being used with those being actively searched into memory. `gisTileManager` is capable of performing each of the algorithms discussed in Chapter 3.

Previous work had difficulty analyzing large datasets due to the large search space and limited memory available. To address this problem, we present a restrictive tiling scheme in which the search space is drastically reduced based on an approximation of the actual path. First, the full dataset is divided into into several hundred *tiles*. Next, an approximate path is generated on the simplified dataset using `gisMesh`. We determine which *tiles* the approximate path crosses over, and use this information to restrict the algorithms contained in `gisTileManager` to only searching within these *tiles*. For robustness, a configurable *buffer* can be set so that if the path falls near the boundary of a *tile*, the neighboring *tiles* can also be searched. Through several experiments, its appears that this method is successful in limiting the search space without compromising the accuracy of the resulting path. This is a very important feature of our imple-



(a) Unrestricted (b) Restricted

Figure 4.3: The impact of restricting the search space. The green tiles indicate those currently in memory. The red tiles do not currently fit in memory, but are needed to continue the computation. (a) shows an unrestricted run where a large amount of swapping will occur, whereas (b) shows a restricted search space. Notice that when the wavefront encounters an edge, those tiles can be removed from memory permanently. This allows all tiles on the wavefront to be in memory, avoiding costly swaps.

mentation as it prevents a large amount of data swapping that would occur if the path computation was allowed to run un-restricted on the full dataset. Figure 4.3 illustrates the importance of this feature.

Figure 4.4 shows the application part-way through a path computation on the full dataset along with which *tiles* are in the current search space. Figure 4.5 illustrates a path found on the simplified dataset along with the corresponding path found on the full dataset.

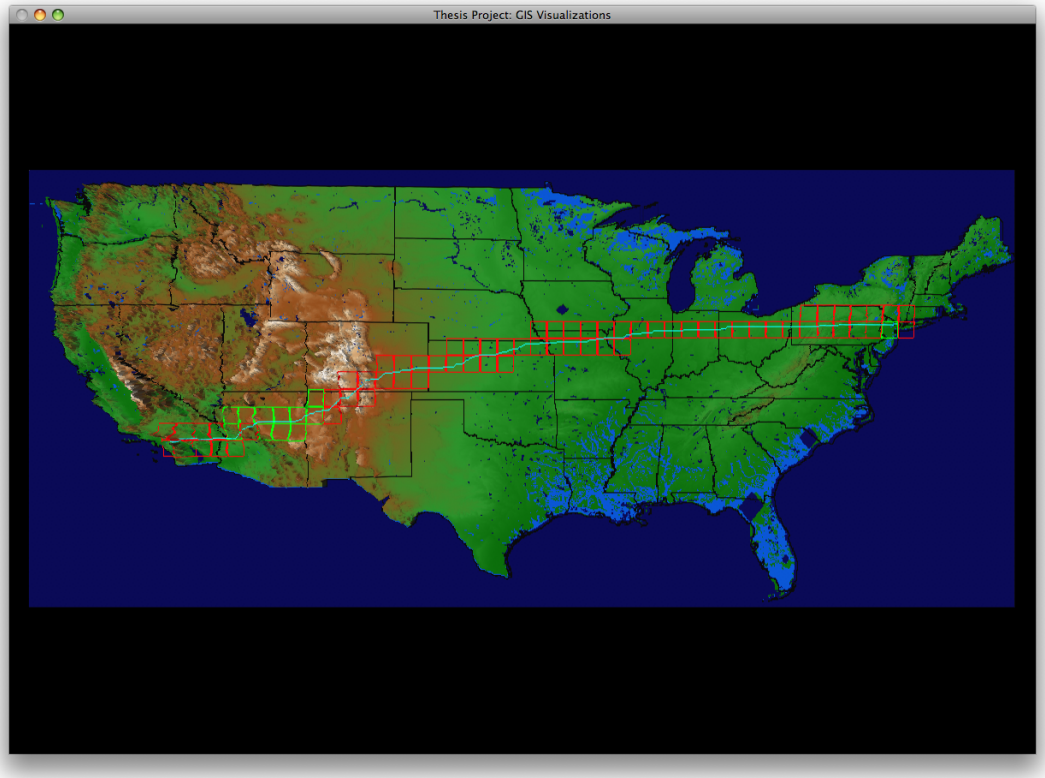


Figure 4.4: Display of an in progress path computation on the full dataset. The red squares indicate tiles that are within the search space while green tiles are currently loaded in memory. The simplified path (cyan) used to determine which tiles to search is also shown.

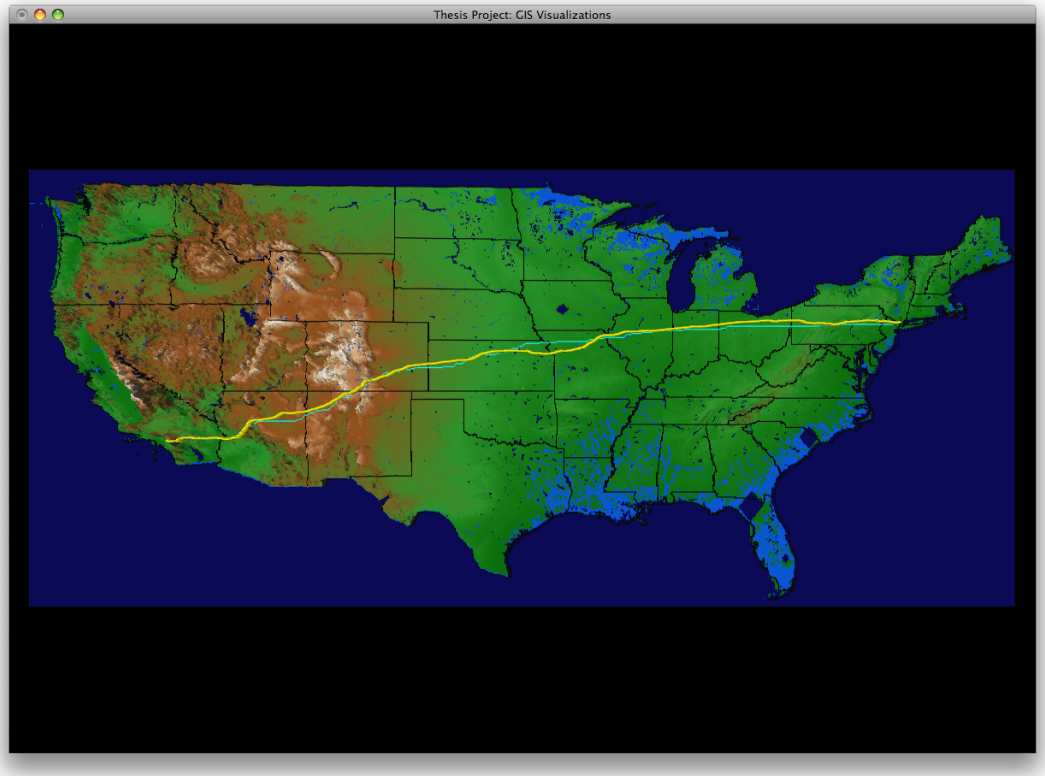


Figure 4.5: Display of the path across the simplified dataset (cyan) with the corresponding path across the restricted dataset (yellow). This path is from Los Angeles to New York City.

4.4 Visualizations

Each *tile* can be viewed individually, thus allowing a more detailed representation of a particular portion of the dataset. In addition to showing the fine-grained details of a particular path, several visualizations generated by `gisTileManager` are available which allow further analysis of the generated paths.

Figure 4.6 illustrates the *landcover effect* visualization, which shows how incorporating landcover data can affect the generated paths.

Figure 4.7 illustrates the *directional difference* visualization, which shows how the energetic path can differ based on which direction you are traveling. Note that it is often the case that both directions use the same path. Thus, this visualization is only available in rare circumstances.

Figures 4.8 and 4.9 illustrates the *nearby paths* visualization, which shows potential paths that end near the original destination. This is useful for observing nearby paths that may be traversed if the original path is blocked or undesirable in some way.

Figures 4.10 and 4.11 illustrates the *alternate destinations* visualization, which shows potential destinations that could be reached for the same caloric cost as the original destination. Note that this visualization is only available if the path was constructed using Dijkstra's shortest path algorithm as it is the only algorithm which provides accurate caloric costs in a single-source multiple-destination manner.

Figure 4.12 illustrates the *caloric radius* visualization, which shows areas that can be reached from a particular location for a range of caloric costs. This can be used to determine the area in which a person could walk in a given amount

of time, or how far a particular tribe could forage for food. For the latter case, this provides a better estimate than straight distance as the terrain can greatly effect the reachable areas.

Figures 4.13 and 4.14 illustrates the *divergent paths* visualization, which shows areas that could be reached while traversing the original path for a nominal caloric cost. This is useful for showing potential areas of travel should the original path be blocked, areas that could be explored with minimal extra energy expenditure, or areas that could represent possible camp locations when on a long journey.

Figure 4.15 illustrates the *gradient* visualization, showing how the cost to reach any point on the *tile* increases radiating out from the start. This is useful for estimating the cost to travel to any location within the *tile*, as well as displaying how the caloric cost is impacted by certain terrain (such as steep mountains or water).

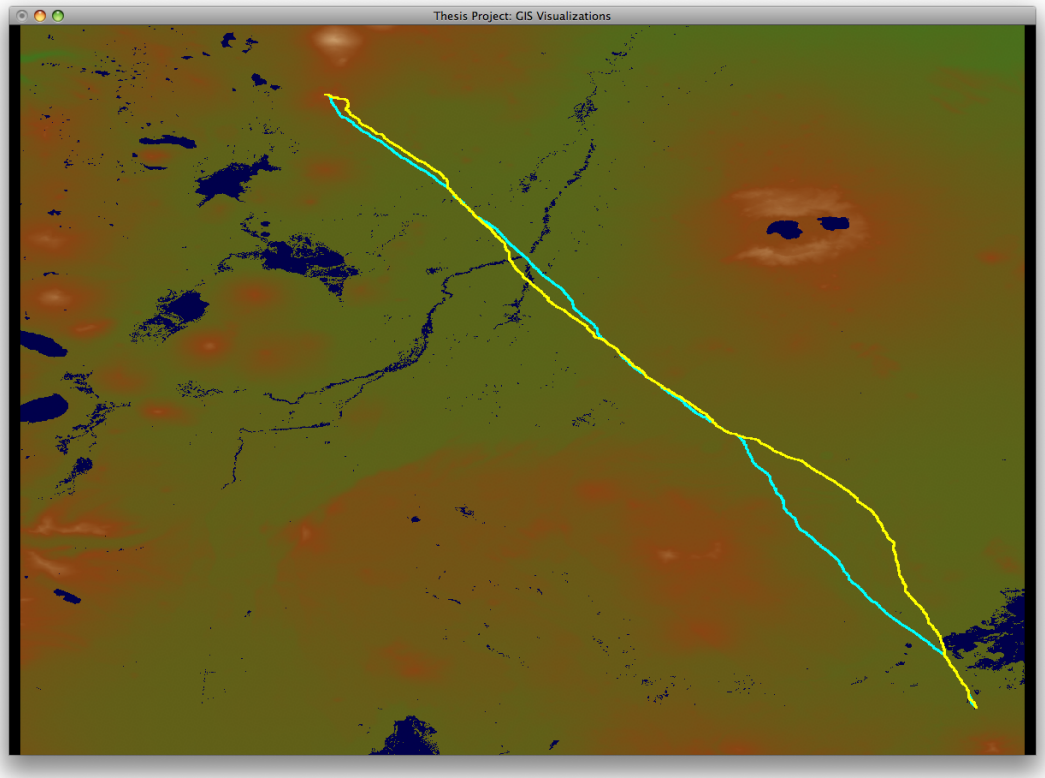


Figure 4.6: The landcover effect visualization. The yellow path takes into account the specific landcover of the terrain, while the cyan path treats all terrain as grasslands. Note that the terrain is colored based on its elevation, not its landcover.

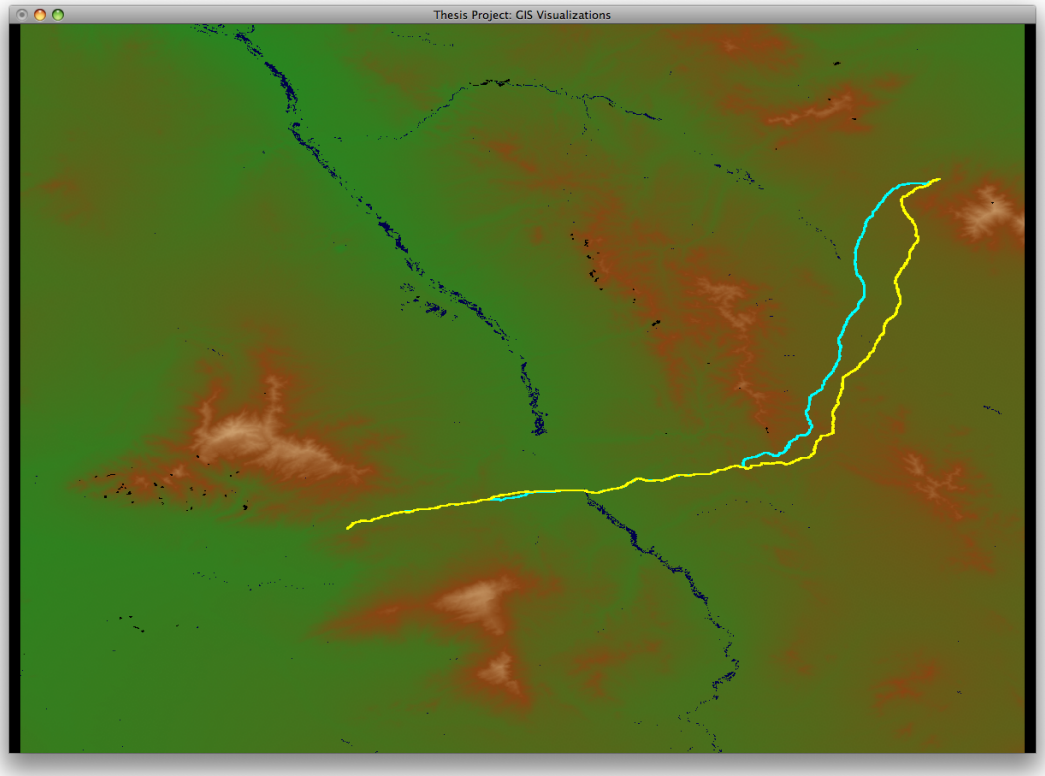


Figure 4.7: The directional difference visualization. The yellow path is the most energetic when traveling left to right, while the cyan path is the most energetic when traveling right to left.

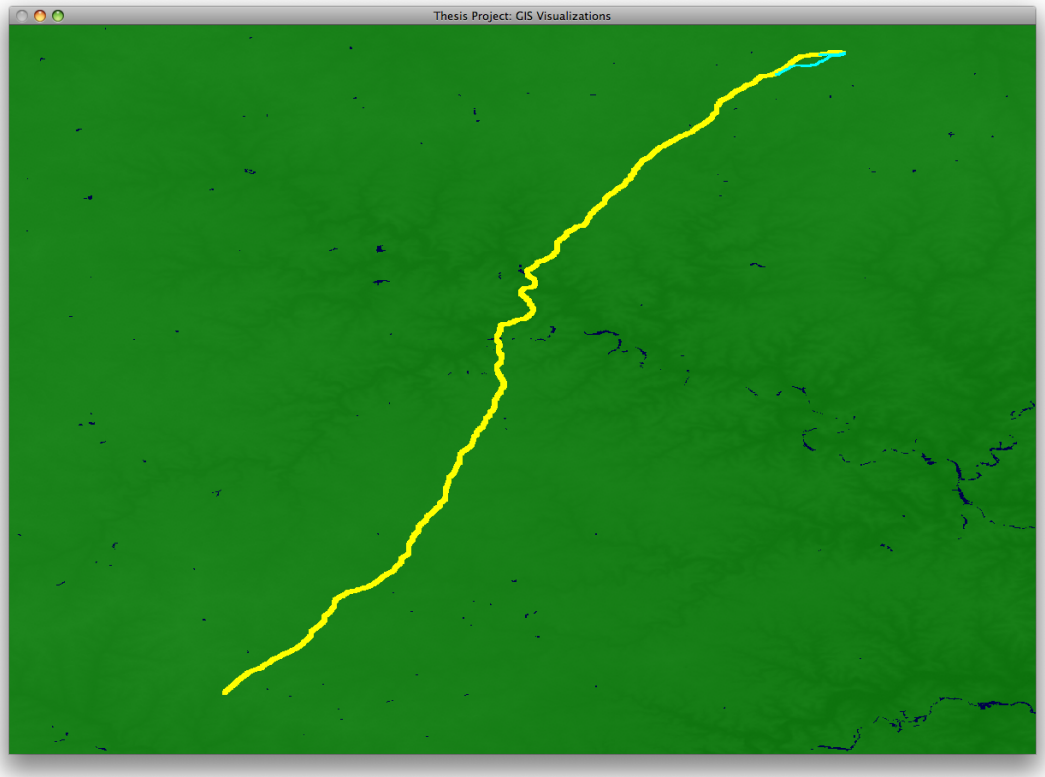


Figure 4.8: The nearby paths visualization. The yellow line indicates the original path, while the cyan line indicates a path from a destination that is very close to the original destination. On flat terrain, the nearby paths are very similar to the original. This is due to the actual length of the path having a higher impact than the minimal elevation changes.

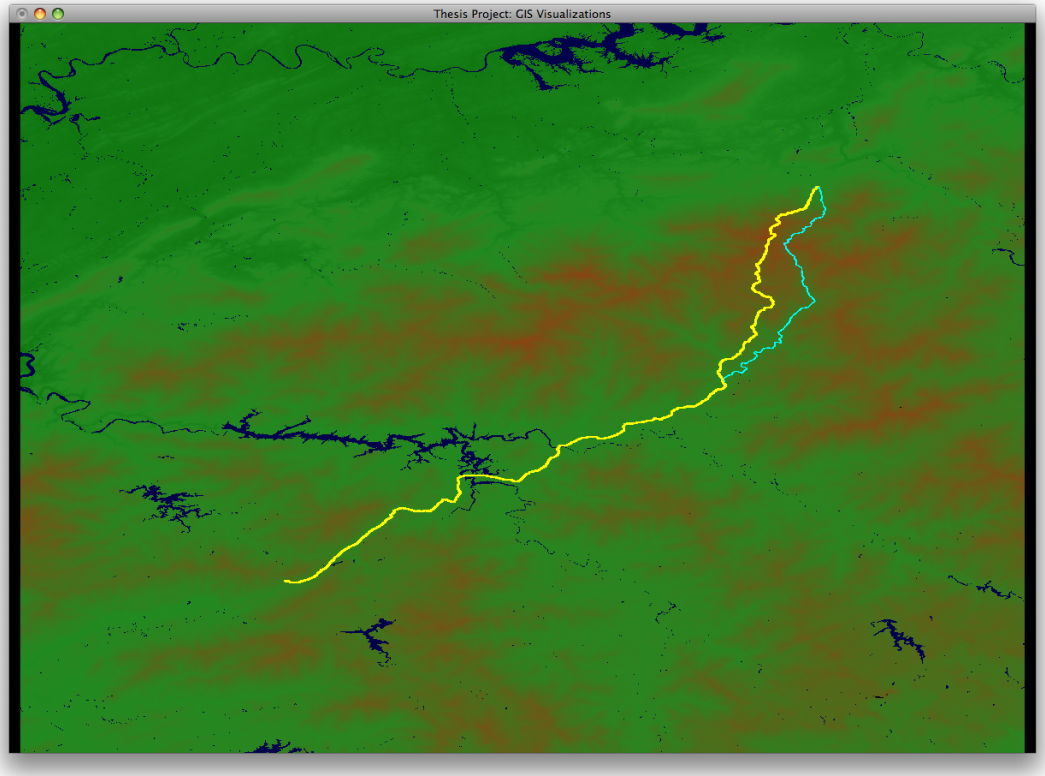


Figure 4.9: Another nearby paths visualization. The yellow line indicates the original path, while the cyan line indicates a path from a destination that is very close to the original destination. Compared to Figure 4.8, mountainous terrain may contain several distinct paths to avoid natural obstacles.

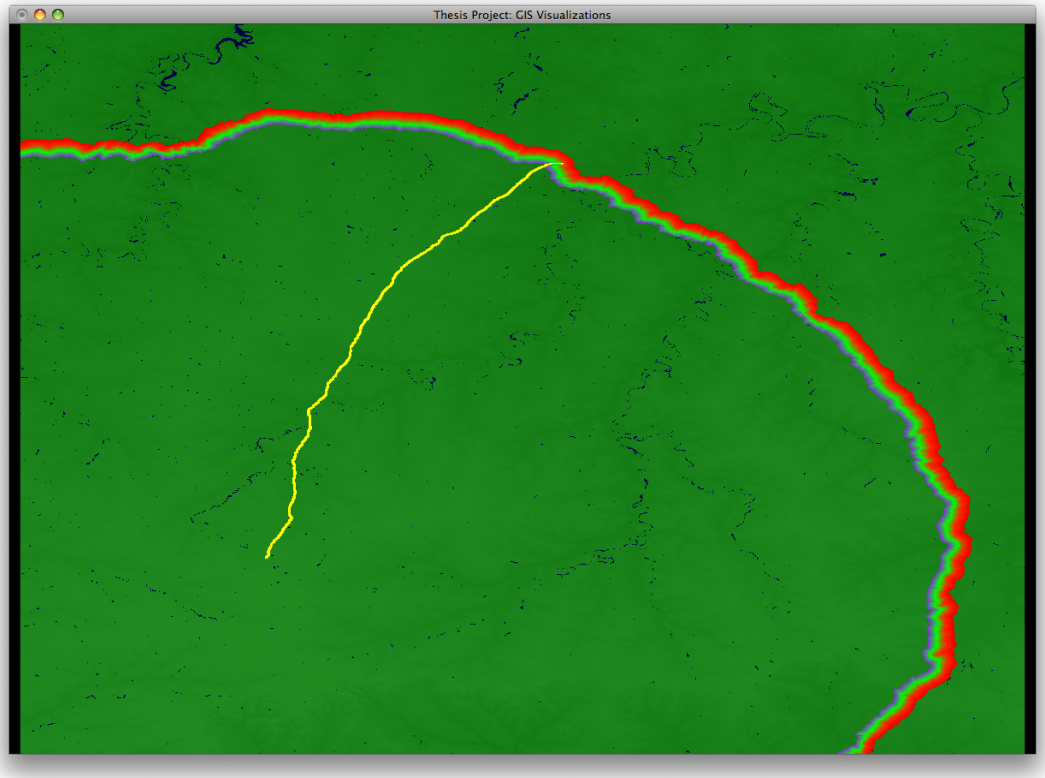


Figure 4.10: The alternate destinations visualization. The colored area represents destinations that can be reached from the start with a similar cost to the original destination. The color is scaled between violet (-150 kilocalories of original), green (same cost as original), and red (150 kilocalories of original).

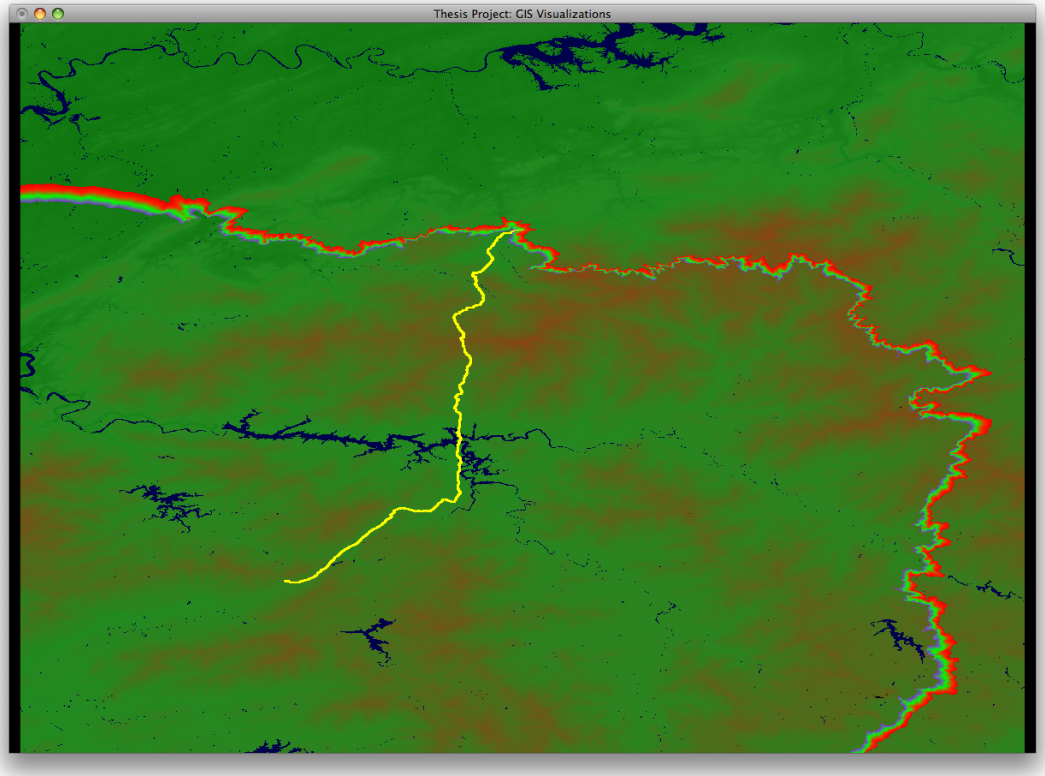


Figure 4.11: Another alternate destinations visualization. Compared to Figure 4.10, notice that mountainous terrain can greatly effect the overall distance that can be traversed for a similar cost to the original destination.

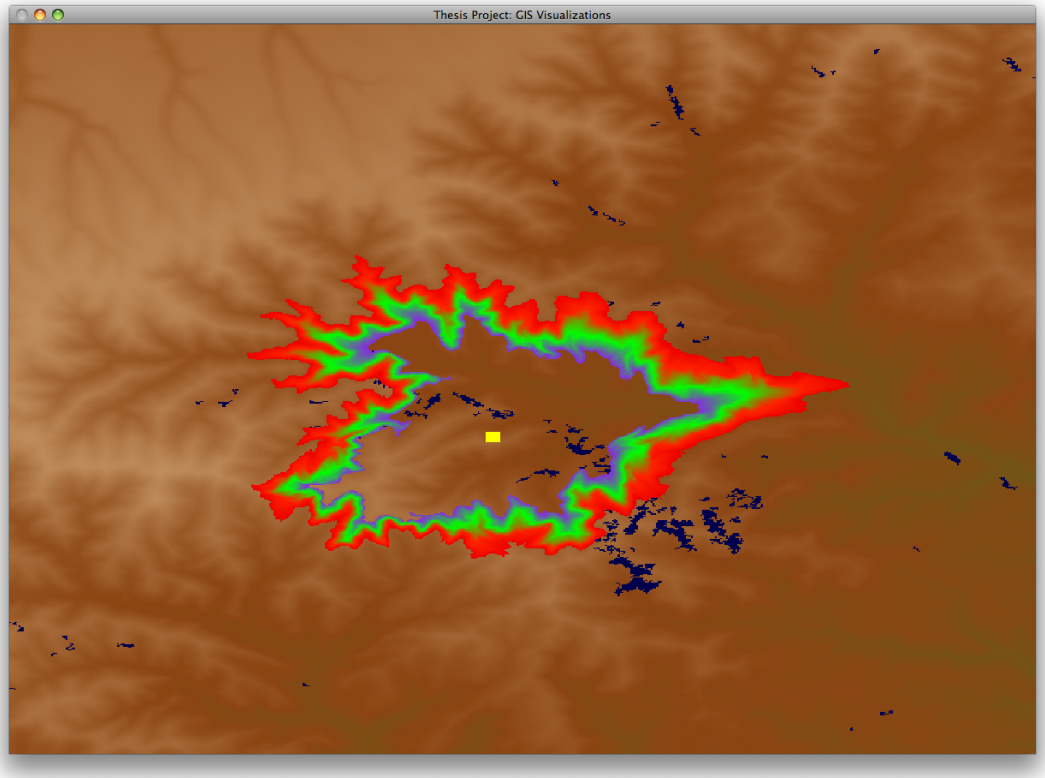


Figure 4.12: The caloric radius visualization. The colored area represents destinations that can be reached from the location for a particular cost. The color is scaled between violet (850 kilocalories), green (1000 kilocalories), and red (1150 kilocalories).

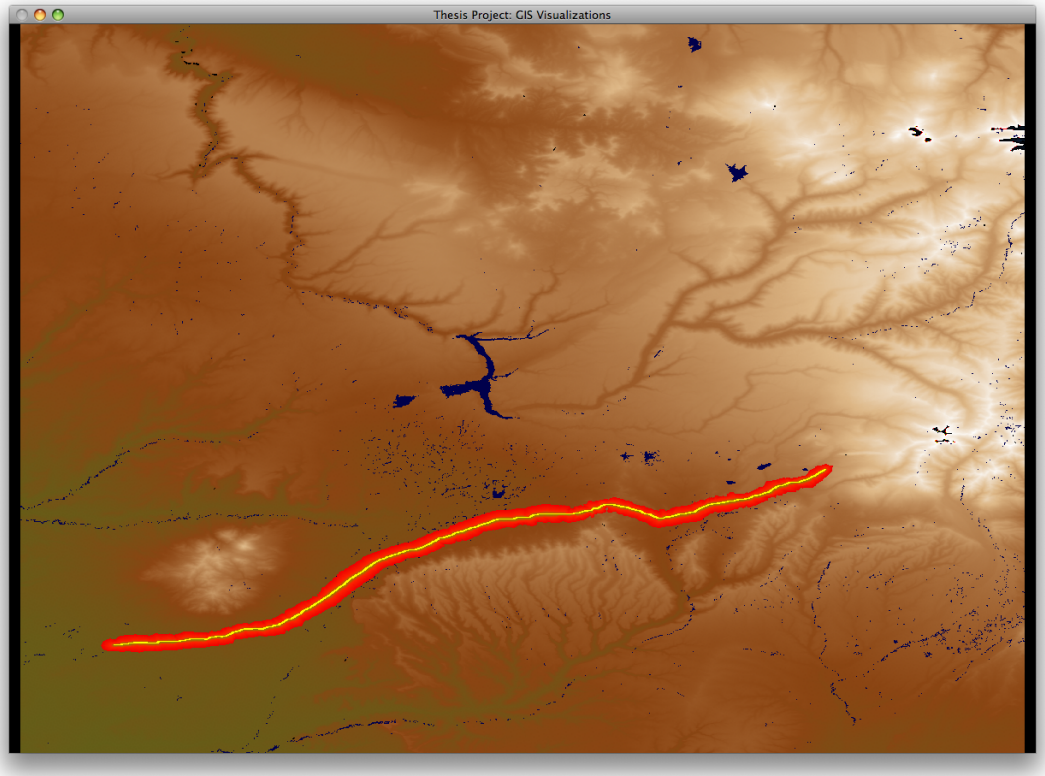


Figure 4.13: The divergent paths visualization. The green areas represent areas that can be reached within a 50 kilocalorie expenditure. The red areas can be reached within a 150 kilocalorie expenditure.

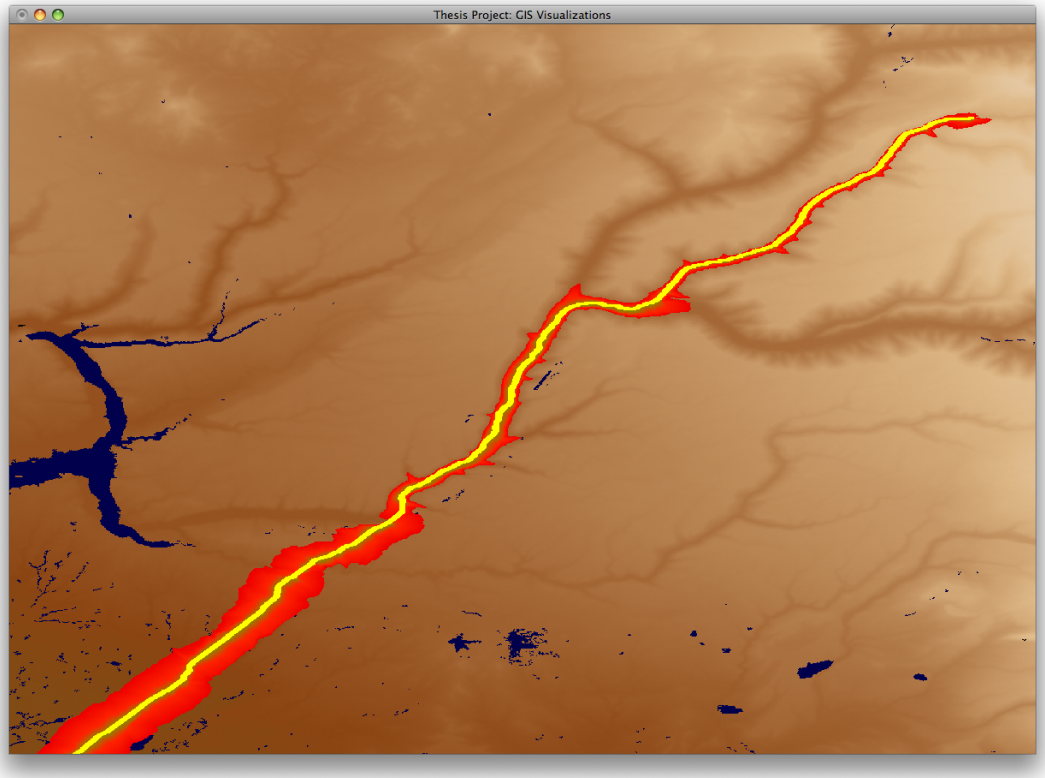


Figure 4.14: Another divergent paths visualization. Using the right half of Figure 4.13 as a terrain reference, notice that a larger area can be reached when the path falls on more open terrain (lower left quadrant), whereas the reachable area drastically decreases as it traces through a valley (upper right quadrant).

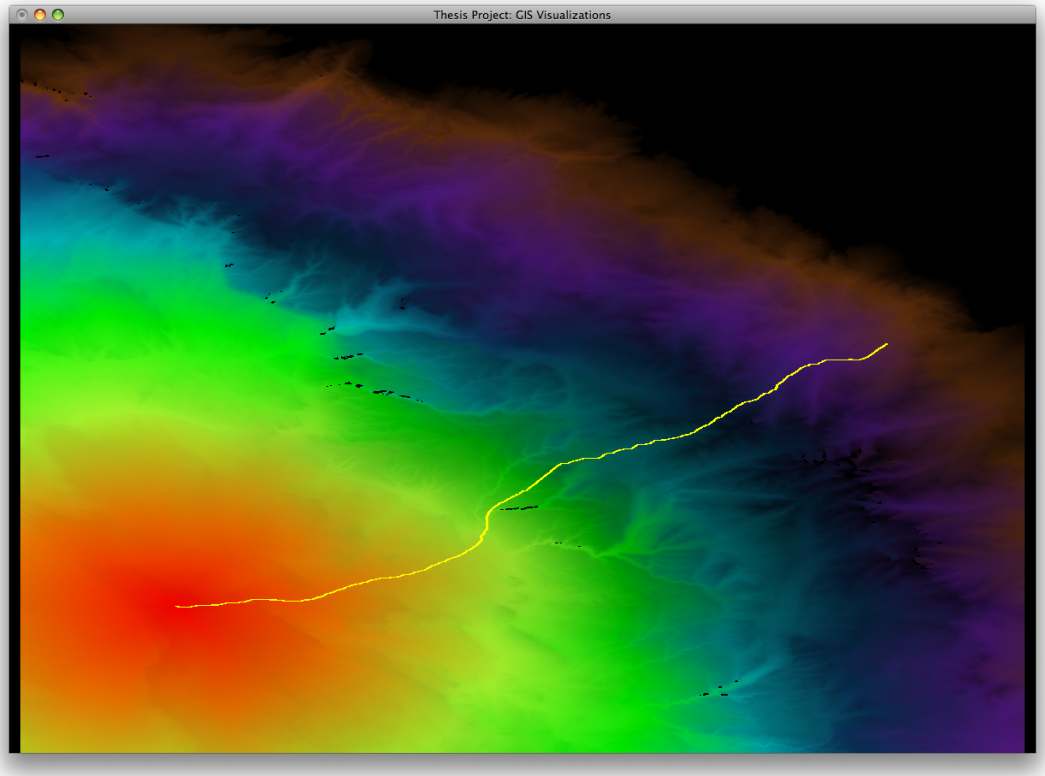


Figure 4.15: The gradient visualization. The colors are scaled based on their percentage of the final cost in 15% increments.

Chapter 5

Results

To demonstrate the tools effectiveness, a number of paths were constructed which tested their ability to accurately and efficiently analyze massive terrain datasets. Several representative results are presented here[1]. Table 5.1 describes each path finding method that was used. For each result, the error percentage is based on the difference in cost between the indicated method and the cost obtained from running Dijkstra's shortest path algorithm unrestricted on the full dataset (marked with a * in the results table). The 'Nodes' column indicates the number of data points that were analyzed by the particular method. All costs are in kilocalories.

Note that the results occasionally show a slight difference in cost between Dijkstra's shortest path algorithm and the Fast Dijkstra variant. The paths generated from each method were checked and in all cases the two paths were found to be identical. Thus, we attribute the cost difference to be caused by floating point rounding differences during the caloric cost calculations.

[1] All results were obtained on a 2.5 Ghz Intel Core 2 Duo MacBook Pro running OS X 10.5.6 with 4 GB 667 MHz DDR2 SDRAM and a 5400 RPM Fujitsu hard drive.

Also note that the results provided for the PRM algorithms are for a specific run. Due to the probabilistic nature of the algorithms, each run will likely produce a different result.

$Dijk$	The standard Dijkstra's shortest path algorithm.
$Dijk_{Fast}$	The multi-threaded variant of Dijkstra's shortest path algorithm as described in section 3.3.
A^*	The standard A* algorithm. The heuristic used is the caloric cost of traveling from the current node directly to the destination node.
PRM	The Probabilistic Road Map algorithm as described in section 3.4. Referring to Algorithm 3, the following conditions are imposed: • The Kindel method is used to select the node n to expand (line 3) • For increased speed, n is expanded four times, with each expansion node n' occurring eight data points away from n in a random direction (line 4) • The endgame region contains all nodes within 128 data points of the destination (line 7) • A node can be selected for expansion an unlimited number of times
PRM_{Fast}	A variant of PRM designed for additional speed over accuracy. All conditions of PRM are used with the exception that a node can only be selected for expansion twice. In general, this will lower the accuracy of the path by reducing the number of nodes searched, as well as preventing expensive paths from being replaced by more efficient ones. See Section 3.4.1 and Figure 3.5 for details.
$Dijk \rightarrow \alpha$	α can be any of the above methods. Indicates that $Dijk$ was run on the simplified dataset to determine an approximate path ξ, followed by α on the full dataset, but restricted to the tiles crossed by ξ.
Buffer	Applies to $Dijk \rightarrow \alpha$. 'Without Buffer' indicates that only the tiles crossed by an approximate path are searched on the full dataset. 'With Buffer' indicates that the tiles crossed by an approximate path, as well as neighboring tiles when the path falls near a tiles border, are searched on the full dataset.

Table 5.1: Path Finding Methods

5.1 California Indian Trails

This test was selected to demonstrate the potential application to the field of Archeology and Anthropology. Historical records show evidence of healthy trade relations among many of the California Indian tribes. However, the actual trails used for these trades is somewhat of a mystery. Work by James Davis [7] has provided several plausible routes between the numerous tribes, as shown in Figure 5.1.

A trail was plotted between two Indian tribes, one located within a valley between two mountain ranges[2], and the other located near the coast[3]. The distance between these two tribes necessitates searching most of southern California. Figure 5.2 displays the energetic path found by the tools, while Table 5.2 shows the runtime required for the different algorithms.

As can be seen, using the simplified dataset to get an approximate path and restricting the search space (with a small buffer) on the detailed dataset can still produce a perfectly accurate path. Notice that the required time is drastically reduced for both the $Dijkstra's$ and $Fast\ Dijkstra$ variant, but with no error in the path. In addition, A^* provides a path with very little error while requiring even less time than the $Fast\ Dijkstra$ algorithm. However, PRM took a substantially longer amount of time to complete and provided a highly inaccurate path. This does not conclusively determine the inappropriateness of PRM as the probabilistic nature of it means it may occasionally require a longer runtime. In addition, the specific implementation of PRM can drastically change the results. This can be seen in PRM_{Fast}, which completes very quickly, but provides a path with a much higher energy requirement.

[2] 37° 06' N, 118° 25' W
[3] 34° 11' N, 119° 6' W

Unrestricted

Method	Nodes	Runtime	Memory	Cost	Error
$^*Dijk$	265,024,564	1h 21m 3s	2.36 GB	42,455.5	0.00%
PRM_{Fast}	7,888,454	42m 3s	2.66 GB	91,207	114.83%

With Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	92,548,510	23m 37s	1.35 GB	42,455.5	0.00%
$Dijk \rightarrow Dijk_{Fast}$	100,371,403	14m 26s	1.51 GB	42,455.2	0.00%
$Dijk \rightarrow A^*$	39,576,328	11m 31s	1.35 GB	42,880.5	1.00%
$Dijk \rightarrow PRM$	77,423,841	42m 3s	1.19 GB	54,417.4	28.17%
$Dijk \rightarrow PRM_{Fast}$	2,665,417	46s	1.19 GB	76,938.2	81.22%

Without Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	54,998,209	13m 10s	919 MB	48,748.9	14.82%
$Dijk \rightarrow Dijk_{Fast}$	60,236,478	8m 12s	997 MB	48,748.4	14.82%
$Dijk \rightarrow A^*$	42,019,677	11m 22s	923 MB	49,175.5	15.82%
$Dijk \rightarrow PRM$	38,949,512	22m 15s	758 MB	67,133.6	58.12%
$Dijk \rightarrow PRM_{Fast}$	1,579,735	30s	911 MB	91,187.1	114.78%

Table 5.2: Results for path computation of a California Indian Trail.

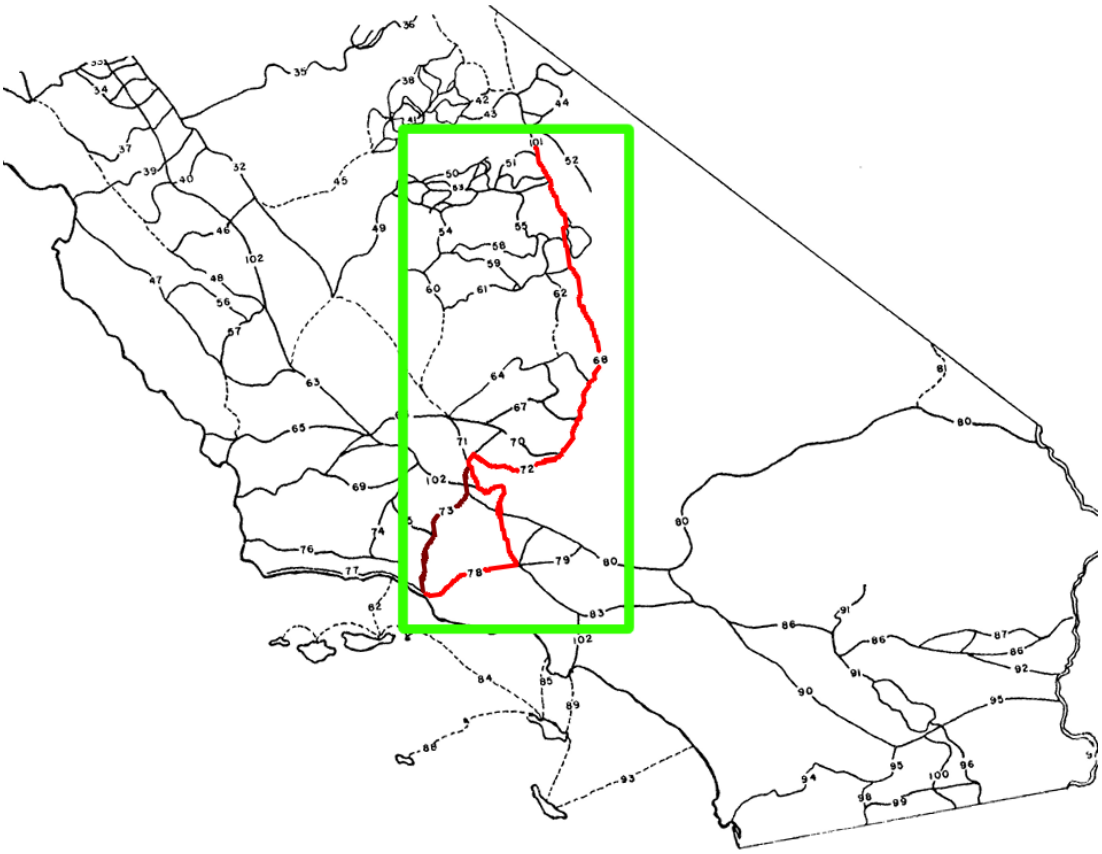


Figure 5.1: A map of California Indian trails by James Davis [7]. The green rectangle indicates the area shown in 5.2 while the red and brown paths indicate possible corresponding trails.

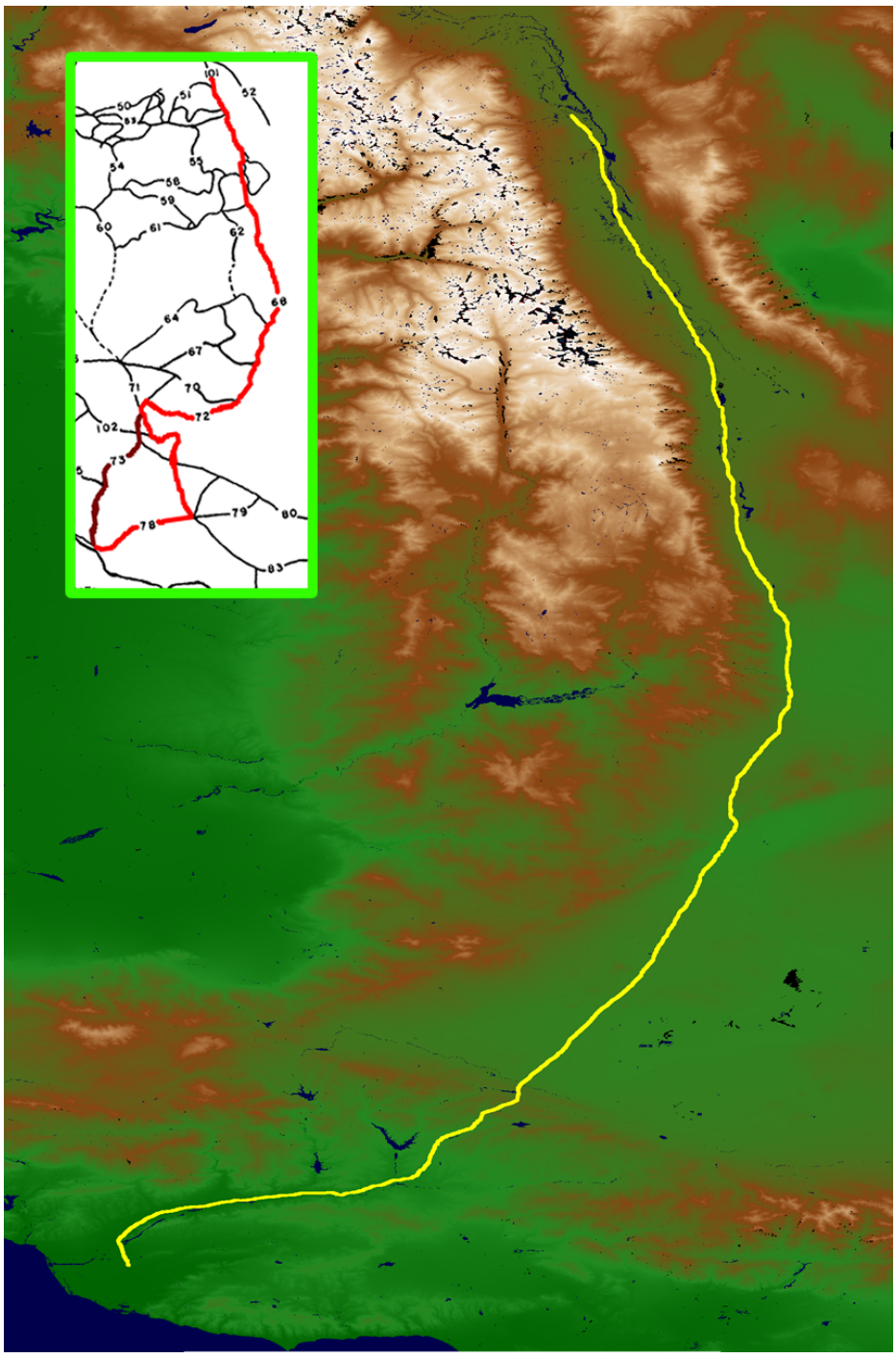


Figure 5.2: An energetic path found by the application, possibly corresponding to the trails shown in Figure 5.1.

5.2 Oregon Trail

This test was selected to allow a comparison against previous work while demonstrating a historical application. The Oregon Trail is a well known trail taken by settlers journeying to the western United States in the mid 1800's. Despite the name, the trail actually splits in several places, allowing travel to various places of settlement in what is now known as California, Oregon, and Washington.

This test concentrates on a section of the Oregon Trail that crosses Oregon itself. The trail begins at Old Fort Boise in Idaho[4] and ends at Oregon City in Oregon[5]. Figures 5.3 and 5.4 show the paths generated on the simplified and full datasets respectively, while Table 5.3 presents the results.

Rickwald used this trail extensively when testing his implementation. While no timing information is given, he did note that constructing a path across Oregon required most of the day. By contrast, the restrictive tiling scheme presented in this work significantly reduces the amount of time required while still producing accurate paths. The results are comparable to those of the California Indian Trail in that both the *Dijkstra* and *Fast Dijkstra* methods produced accurate paths even with a highly restricted search space (less than 1% error when run without a buffer). PRM_{Fast} again requires very little time, but produces highly inaccurate paths. It is interesting to note that A^* had a longer runtime than *Dijkstra* in both the buffered and unbuffered cases. We hypothesize that the heuristic, which ignored the presence of water, may have caused the algorithm to explore paths which required many river crossings before discovering more energetically optimal terrain.

[4]Old Fort Boise: 43° 37' N, 116° 11' W
[5]Oregon City: 45° 21' N, 122° 35' W

Unrestricted

Method	Nodes	Runtime	Memory	Cost	Error
$^*Dijk$	505,896,251	2h 41m 50s	2.34 GB	74,653.2	0.00%
PRM_{Fast}	16,601,736	28m 22s	2.68 GB	209,445	180.55%

With Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	184,985,532	46m 5s	1.80 GB	74,653.2	0.00%
$Dijk \rightarrow Dijk_{Fast}$	181,351,914	26m 9s	1.95 GB	74,655.7	0.00%
$Dijk \rightarrow A^*$	164,865,746	47m 54s	1.95 GB	75,776.5	1.50%
$Dijk \rightarrow PRM$	164,464,736	2h 22m 29s	1.94 GB	97,402	30.47%
$Dijk \rightarrow PRM_{Fast}$	5,635,817	3m 10s	2.00 GB	131,743	76.47%

Without Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	110,063,682	26m 43s	1.50 GB	75,046.5	0.53%
$Dijk \rightarrow Dijk_{Fast}$	116,559,602	15m 30s	1.51 GB	75,046.1	0.53%
$Dijk \rightarrow A^*$	101,173,730	27m 15s	1.50 GB	75,605.8	1.28%
$Dijk \rightarrow PRM$	143,160,284	2h 13m 50s	1.34 GB	98,104.2	31.41%
$Dijk \rightarrow PRM_{Fast}$	3,702,639	3m	1.55 GB	118,478	58.70%

Table 5.3: Results for path computation between Old Fort Boise (ID) and Oregon City (OR).

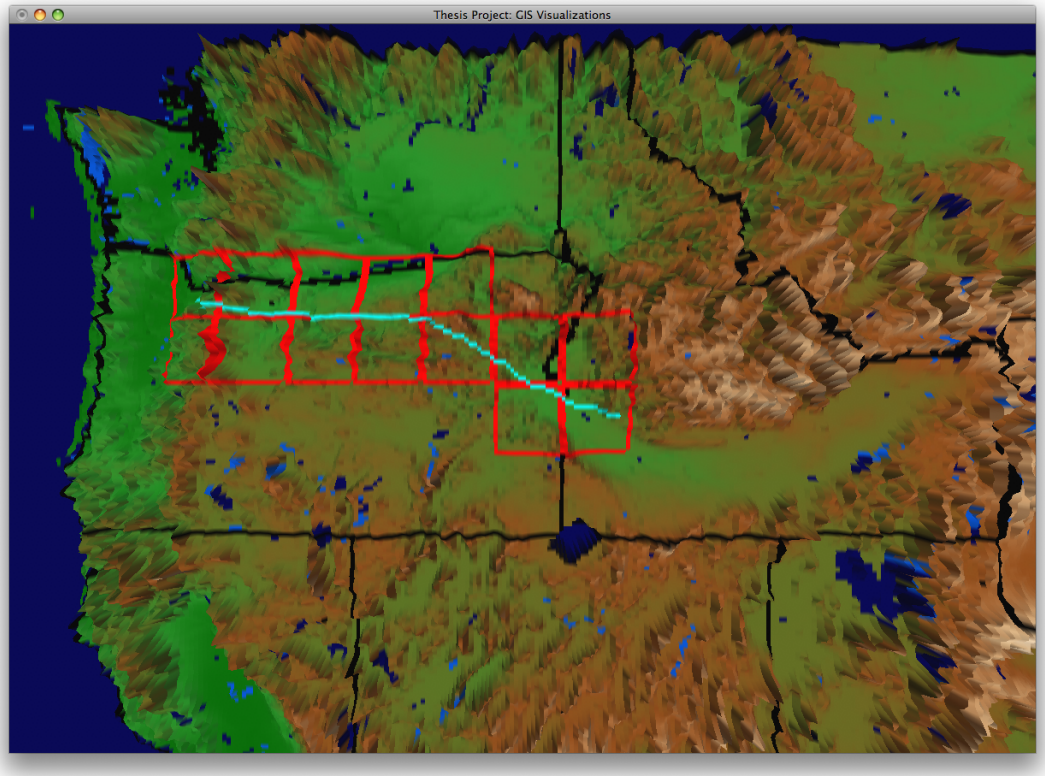


Figure 5.3: The estimated Oregon Trail produced on the simplified dataset. The red boxes indicate the search space for the algorithms run on the full dataset.

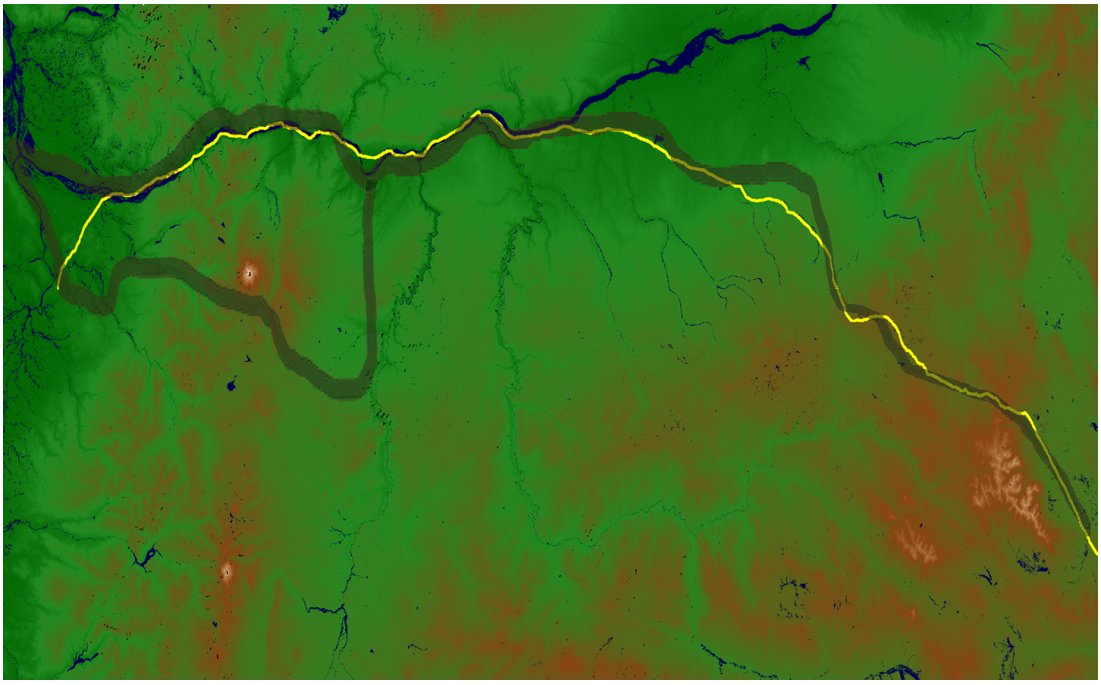


Figure 5.4: An energetic path overlayed with an estimate of the actual historic Oregon Trail [5].

5.3 Moundbuilders

This test was designed to exercise the tools on a path computation over a significant distance. During the 1700's, early settlers of the Eastern United States encountered large mounds of earth that could not have been made by natural means. These mounds were often constructed in perfectly symmetrical shapes, or into exquisite designs. Excavation of hundreds of mounds has revealed that most were constructed as burial chambers, often filled with artifacts depicting the occupants social status or victories in battle. However, some of the mounds contained no human remains, but instead may have had a spiritual purpose (see Figure 5.5). While it was apparent that these mounds were constructed by human hands, the responsible culture remained a point of mystery and debate for over 100 years. Called the Moundbuilders, it is now generally agreed that they consisted of early native American Indians, though from which tribes they came remains unknown [36].

Mounds constructed by the Moundbuilders have been found all along the central and eastern United States. Of particular interest to this work are the mounds found near Hopewell in modern Ohio. Excavation from these mounds have revealed flint from the Rocky Mountains, shells from the Gulf of Mexico, and other artifacts of distant origin [12]. This indicates that the inhabitants engaged in extremely long-distance trade, an impressive feat for a civilization that was long gone by the time European settlers arrived.

While it is known that trade occurred between villages separated by great distances, the actual trade routes remain somewhat of a mystery. Figure 5.6 demonstrates an energetic path that may have been used between two prominent

Figure 5.5: The Serpent Mound found in Southern Ohio. The triangular head appears to be devouring an egg shaped object, while its tail ends in a triple coil over 1,000 feet away. Image courtesy of [10].

Moundbuilder villages; Moundville, Alabama[6] and Hopewell, Ohio[7]. The results are presented in Table 5.4.

This is the longest path presented in this section, as evidenced by the increased runtimes and nodes searched for each method. Notice that, compared to the results for the previous paths, *PRM* is able to construct a relatively accurate path in a small fraction of the time required by *Dijkstra* when run unrestricted on the full dataset. However, the unbuffered *Fast Dijkstra* is able to produce a far more accurate path while consuming considerably less time and memory.

[6]Moundville: 32° 59' N, 87° 37' W
[7]Hopewell: 39° 58' N, 82° 10' W

Unrestricted

Method	Nodes	Runtime	Memory	Cost	Error
$^*Dijk$	735,795,927	3h 18m 39s	2.36 GB	107,723	0.00%
PRM_{Fast}	27,330,219	31m 6s	2.83 GB	145,731	35.28%

With Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	252,449,787	1h 14m 59s	1.94 GB	107,723	0.00%
$Dijk \rightarrow Dijk_{Fast}$	285,154,704	37m 15s	1.95 GB	107,726	0.00%
$Dijk \rightarrow A^*$	230,861,134	1h 12m 18s	1.91 GB	112,149	4.11%
$Dijk \rightarrow PRM$	185,307,75	2h 24m 31s	1.94 GB	123,435	14.59%
$Dijk \rightarrow PRM_{Fast}$	8,897,884	7m 12s	2.00 GB	155,447	44.30%

Without Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	158,805,328	36m 32s	1.94 GB	109,036	1.22%
$Dijk \rightarrow Dijk_{Fast}$	168,498,376	22m 2s	1.95 GB	109,039	1.22%
$Dijk \rightarrow A^*$	147,167,803	42m 26s	1.98 GB	112,376	4.32%
$Dijk \rightarrow PRM$	119,739,506	1h 33m 17s	1.79 GB	126,408	17.35%
$Dijk \rightarrow PRM_{Fast}$	5,479,137	3m 56s	2.00 GB	172,630	60.25%

Table 5.4: Results for path computation between Moundville (AL) and Hopewell (OH).

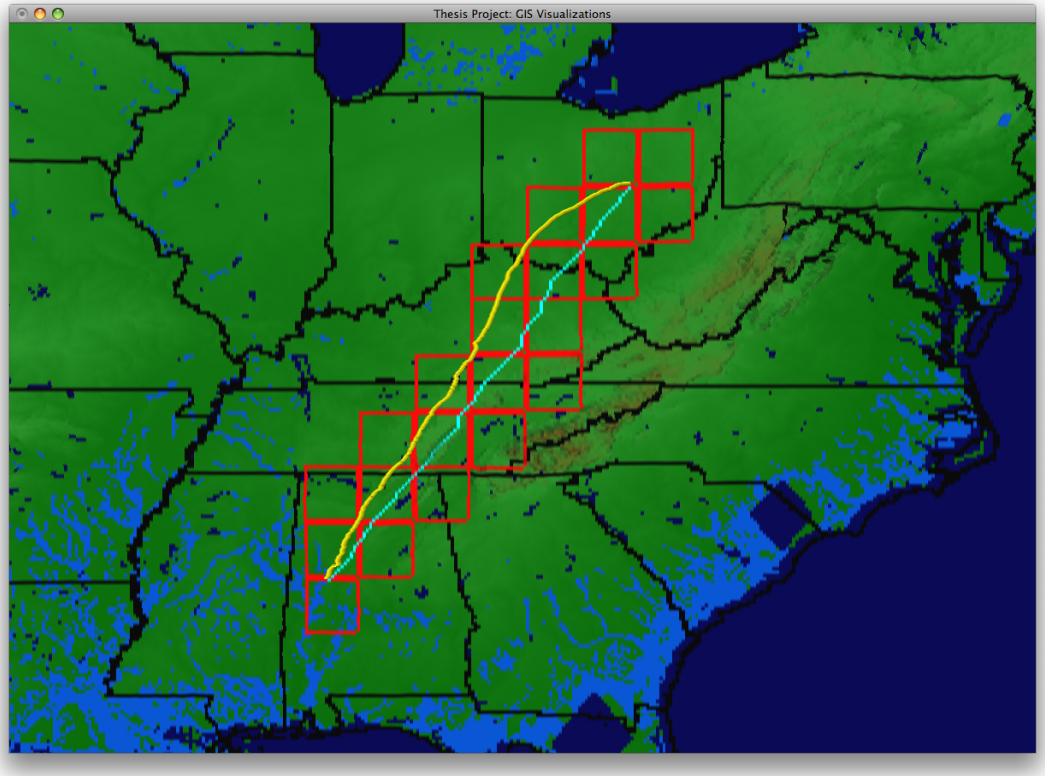


Figure 5.6: A possible trade route utilized by the Moundbuilders. Note that while the path from the simplified dataset (cyan) is quite different than the path from the full dataset (yellow), it is sufficient to determine which tiles in the full dataset should be searched.

5.4 California Archaeological Sites

This test was designed to demonstrate the caloric radius visualization. In the archaeological community, a traditional means of determining a tribes hunting and foraging grounds is to draw a circle on a map, focused on the tribes camp, with a radius of several kilometers. However, this method does not account for the terrain type. For example, a hunting party can travel much further on flat terrain (thus extending the radius) as opposed to very rocky or sloped terrain. Thus, a better metric for determining a tribes hunting ground may be a caloric measure.

Figures 5.7 and 5.8 show archaeological sites in southern California designated CA-SLO-2 and CA-SLO-9 respectively[8]. For context, the body of water seen at the top of each image is Morro Bay. The radius shows the area for which the tribes located at these sites may have searched for food. Notice that when the elevation remains level, such as along the coast or through valleys, a much greater distance can be reached. Conversely, the extra energy required when the traveler must leave the valley in order to reach a higher point of elevation can significantly reduce the distance that can be traveled.

[8]Note that these images show a much closer view of the terrain compared to the other images in this document.

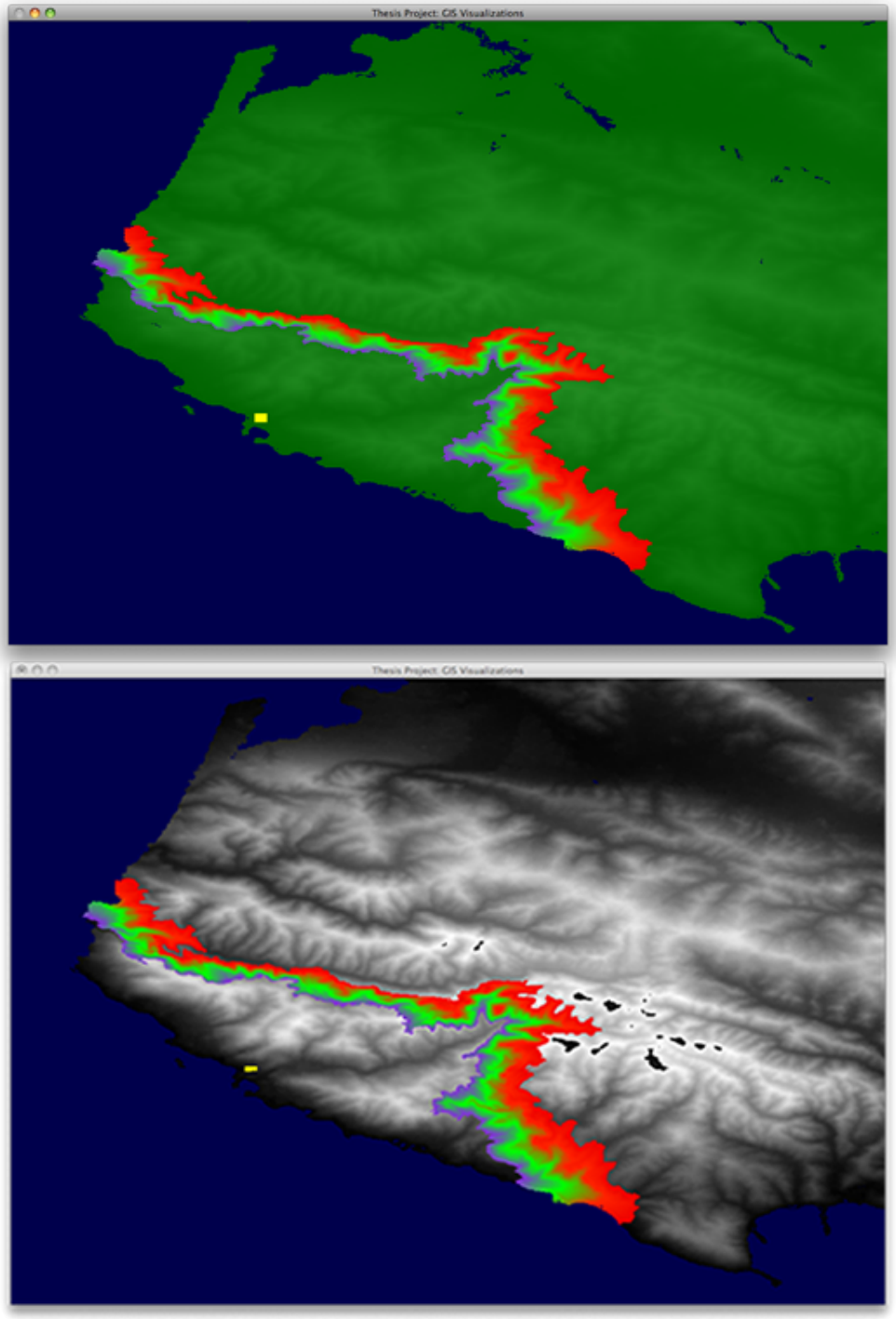


Figure 5.7: The CA-SLO-2 archaeological site with the caloric radius visualization. The color is scaled between violet (850 kilocalories), green (1000 kilocalories), and red (1150 kilocalories).

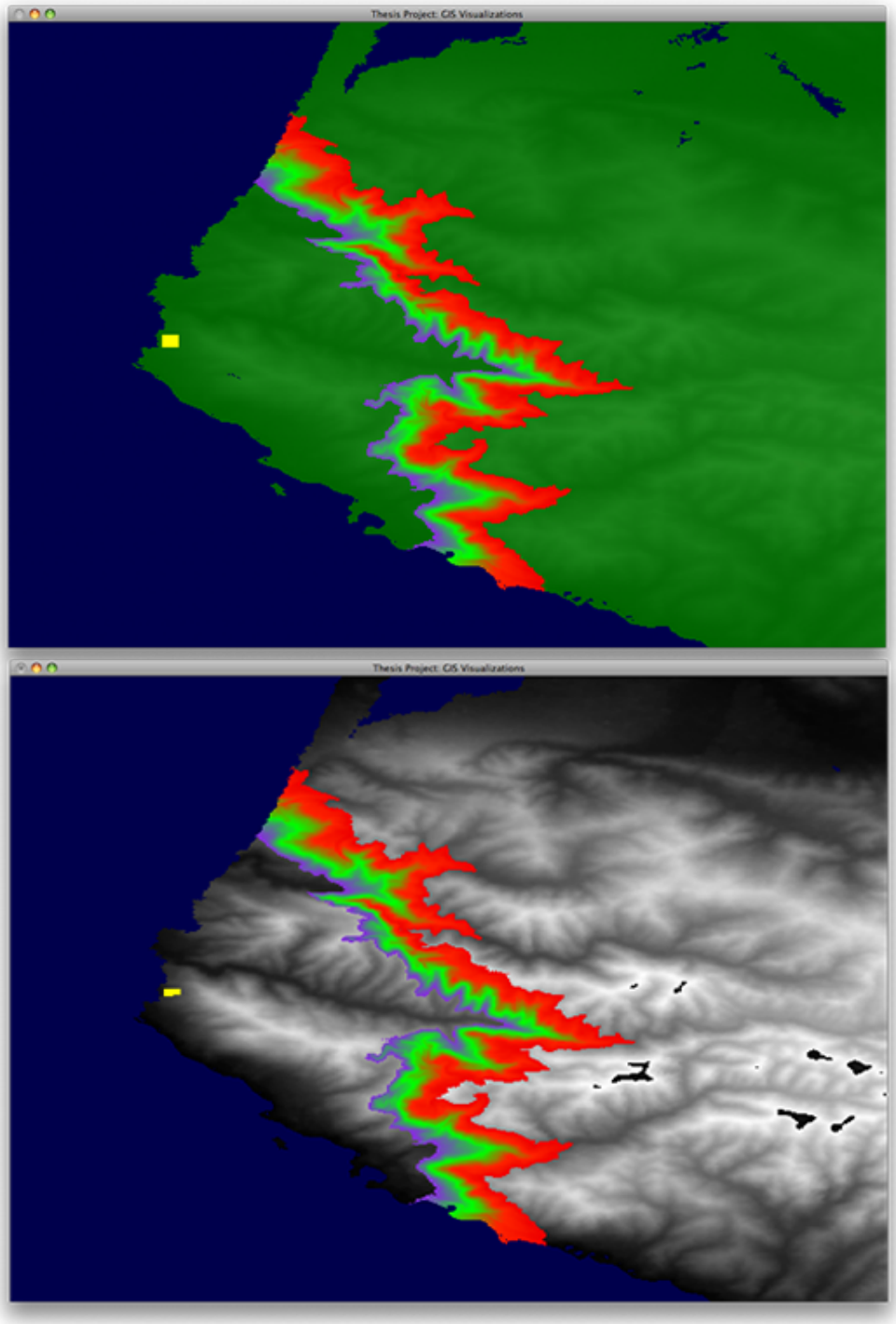


Figure 5.8: The CA-SLO-9 archaeological site with the caloric radius visualization. The color is scaled between violet (850 kilocalories), green (1000 kilocalories), and red (1150 kilocalories).

Chapter 6

Conclusions and Future Work

This work has presented a set of tools that can be used to analyze massive terrain datasets. A key contribution is our use of a simplified dataset and restrictive tiling scheme to vastly reduce the search space while maintaining accuracy. We have also introduced a multi-threaded version of the popular Dijkstra's shortest path algorithm, which drastically reduces the time necessary to construct energetic paths. Moreover, we have introduced the emerging concept of Probabilistic Road Maps to energetic path finding. Finally, we have incorporated several different visualizations that allow for explorations of alternative paths, difficult terrain, and alternate destinations.

We have demonstrated the efficiency and possible historical applications of our tools by performing several experiments to construct energetic paths across large distances using a variety of algorithms. Using our multi-threaded Dijkstra variant combined with the restrictive tiling scheme, we are able to construct an accurate energetic path across the United States in under three hours. This shows a large improvement over previous work in which a path across the state of Oregon required most of a day.

While we have explored several different path algorithms, our results show that using a simplified dataset to obtain a general path, followed by Dijkstra's algorithm (or the Fast Dijkstra variant) provides the best results. While methods such as PRM can provide a path in a much shorter time, the accuracy is generally quite low. In addition, only Dijkstra's algorithm provides the data necessary to construct several of our alternate visualizations, such as the alternate destinations visualization.

There are many possible avenues for future work including interface modifications, improved and additional datasets, and algorithm adjustments. For the interface, the current method of interacting with the tools is manipulated through a series of keyboard shortcuts. It would be necessary to construct an intuitive graphical user interface before this tool would be viable for use in the archaeological or historical community.

For dataset improvements, there are indications that the dataset used to test these tools contained several missing areas, as evident by the seemingly large body of water located near the Florida panhandle. A more complete dataset would improve the accuracy when traveling through certain areas. Similarly, alternate datasets could be obtained to explore other parts of the world. Trails such as those used during the Klondike Gold Rush in Alaska, or the Aboriginal trails of Australia would be very interesting to analyze. In addition, while the dataset simplification methods used provided paths accurate enough to correctly determine which tiles to search on the full dataset (when combined with a small buffer), more advanced simplification methods may provide more accurate paths, thus further reducing the search space.

For algorithm adjustments, there is evidence that higher altitudes may affect the caloric requirements and metabolic rate of humans [37, 44]. It would be

interesting to include an altitude factor when determining energetic paths in these environments. In addition, the caloric equations used in this work do not account for the physical limitations of humans when traversing certain terrain. For instance, it is possible to obtain a caloric cost for traveling down a very steep cliff, but in practice it may not be feasible for a human to traverse this path without specialized equipment. Including these limitations may provide more realistic paths in certain situations. Finally, while the implementations of PRM in this work provided less than satisfactory results, there are still many implementations and optimizations that could be used to improve the viability of these methods. Several possibilities include taking advantage of the fast runtime by running PRM multiple times and presenting the best result, a bidirectional approach for further speed increases, or running PRM once and using Dijkstra's algorithm between interspersed nodes on the path to obtain locally optimal paths.

Bibliography

[1] J. Brown, C.G., K. Sarabandi, and L. Pierce. Validation of the shuttle radar topography mission height data. *Geoscience and Remote Sensing, IEEE Transactions on*, 43(8):1707–1715, Aug. 2005.

[2] U. C. Bureau. Cartographic boundary files, April 2009. http://www.census.gov/geo/www/cob/st2000.html.

[3] R. Burgmann, P. Rosen, and E. Fielding. Synthetic aperture radar interferometry to measure earth's surface topography and its deformation. *Annual Review of Earth and Planetary Sciences*, 28:169–209, 2000.

[4] J. B. Campbell. *Introduction to Remote Sensing*. The Guilford Press, New York, NY, USA, 2002.

[5] H. O. City. End of the oregon trail interpretive center, April 2009. http://www.historicoregoncity.org/HOC/index.php?view=article&id=57:oregon-trail-maps.

[6] C. Clark. *Dynamic Robot Networks: A Coordination Platform for Multi-Robot Systems*. PhD thesis, Stanford University, Palo Alto, California, May 2004.

[7] J. T. Davis. *Trade Routes and Economic Exchange Among the Indians of California*. University of California Archaeological Survey, CA, USA, 1961.

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

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

[10] B. S. Encyclopedia. Mound builders: Serpent mound, May 2009. http://student.britannica.com/elementary/art-88204.

[11] Y. Epstein, L. A. Stroschein, and K. B. Pandol. Predicting metabolic cost of running with and without backpack loads. volume 53, pages 495–500. Springer Berlin / Heidelberg, 1987.

[12] B. M. Fagan. *From Black Land to Fifth Sun: The Science of Sacred Sites*. Addison-Wesley, Reading, MA, USA, 1998.

[13] T. G. Farr. The shuttle radar topography mission. *Reviews of Geophysics*, 45, 2004.

[14] T. R. Halfhill. Parallel processing with cuda. *Microprocessor Report*, pages 1–8, 2008.

[15] J. A. Harris and F. G. Benedict. *A biometric study of basal metabolism in man*. Cornell University, Mann Library, Ithaca, New York, 1919.

[16] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[17] C. Homer, C. Huang, L. Yang, B. Wylie, and M. Coan. Development of a 2001 national land-cover database for the united states. *Photogrammetric Engineering and Remote Sensing*, 70(7):829–840, July 2004.

[18] D. Hsu. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *In Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4420–4426, 2003.

[19] D. Hsu, J. claude Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *International Journal of Computational Geometry and Applications*, pages 2719–2726, 1997.

[20] D. Hsu, R. Kindel, J. claude Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles, 2000.

[21] T. Jones. Personal Correspondence, May 2009.

[22] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA*, pages 8431–8435, 1998.

[23] J. Knapik. Physiological, biomechanical and medical aspects of soldier load carriage. 2001.

[24] Lakes. Lakes environmental digital terrain data, April 2009. http://www.weblakes.com/lakesdem.html.

[25] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR: 98-11, Department of Computer Science, Iowa State University, 1998.

[26] S. M. Lavalle. Rapidly-exploring random trees: Progress and prospects, 2000.

[27] MRLC. Multi-resolution land characteristics consortium, April 2009. http://www.mrlc.gov/.

[28] NGDC. National geophysical data center, April 2009. http://www.ngdc.noaa.gov/.

[29] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, pages 1–14, New York, NY, USA, 2008. ACM.

[30] U. A. C. of Engineers. *Photogrammetric Mapping*. American Society of Civil Engineers, Reston, VA, USA, 1996.

[31] OpenGL. Opengl.org the industry's foundation for high performance graphics, April 2009. http://www.opengl.org/.

[32] Oracle. Berkeley db, April 2009. http://www.oracle.com/technology/products/berkeley-db/db/.

[33] K. B. Pandolf, B. Givoni, and R. F. Goldman. Predicting energy expenditure with loads while standing or walking very slowly. *Journal of Applied Physiology*, 43(4):577–581, 1977.

[34] P. Prampero, D. R. Pendergast, D. W. Wilson, and D. W. Rennie. Energetics of swimming in man. *Journal of Applied Physiology*, 37(1):1–5, 1974.

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

[36] H. C. Shetrone. *The Mound-Builders*. Kennikat Press, Inc., 1964.

[37] K. K. Srivastava and R. Kumar. Human nutrition in cold and high terrestrial altitudes. *International Journal of Biometeorology*, 36(1):10–13, March 1992.

[38] SRTM. Ftp server, April 2009. ftp://e0srp01u.ecs.nasa.gov/.

[39] M. C. Tugurlan and B. Bourdin. Distributed fast marching methods. In *MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–1, New York, NY, USA, 2008. ACM.

[40] USGS. United states geological survey, April 2009. http://www.usgs.gov/.

[41] P. Webb. *Bioastronautics Data Book*. NASA, USA, 1964.

[42] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph.*, 27(4):1–16, 2008.

[43] WebGIS. Geographic information system resource, April 2009. http://www.webgis.com/terraindata.html.

[44] B. Whitten, R. Burlington, M. Posiviata, C. Sidel, and G. Beecher. Amino acid catabolism in environmental extremes: effect of high altitude and calories. *Am J Physiol*, 218(5):1346–1350, 1970.

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

[46] B. M. Wood and Z. J. Wood. Energetically optimal travel across terrain: visualizations and a new metric of geographic distance with anthropological applications. volume 6060, page 60600F. SPIE, 2006.

Appendix A

More results

The following tables and figures show additional results. The 'Nodes' column indicates the number of data points that were analyzed by the particular method. The error percentage is based on the difference in cost between the indicated method and the cost obtained from running Dijkstra's shortest path algorithm unconstrained on the full dataset (marked with a * in the results table). All costs are in kilocalories. See Table 5.1 for an explanation of the various methods.

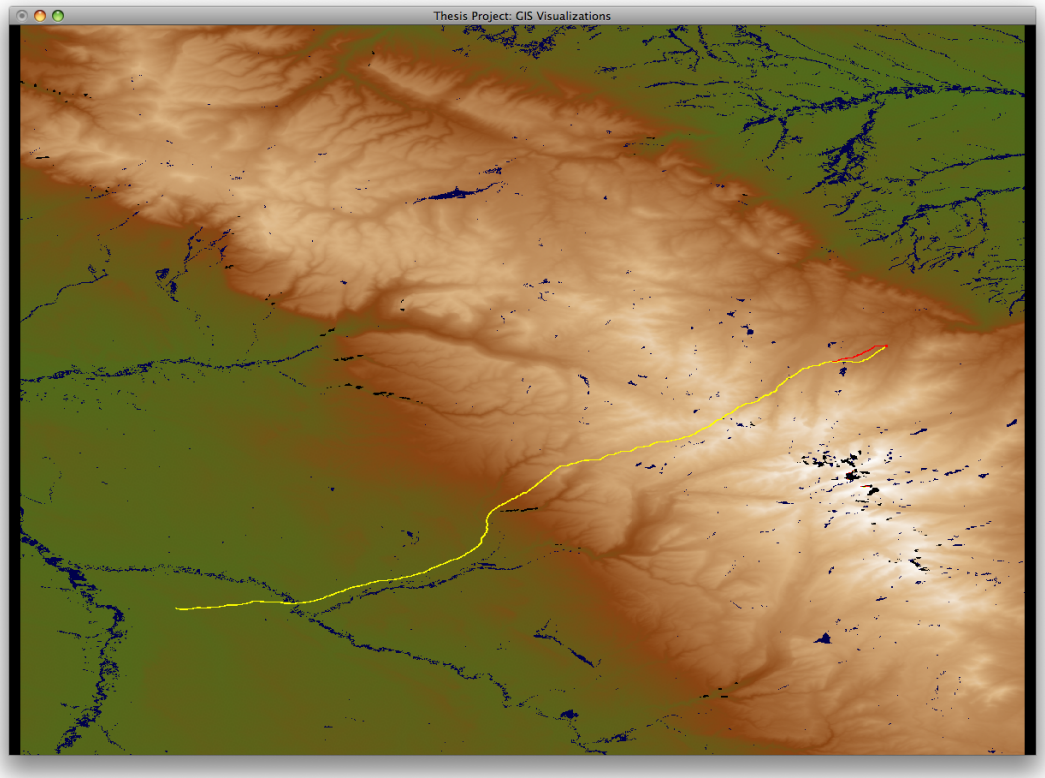


Figure A.1: An energetic path for the single tile test. The red line indicates an alternate path that has an ending nearby the original destination.

Method	Nodes	Runtime	Memory	Cost	Error
$*Dijk$	33,387,296	8m 27s	1.20 GB	11,048.4	0.00%
PRM	15,442,241	4m 15s	758 MB	12,094.8	9.47%
$Dijk \rightarrow Dijk$	9,625,255	2m 9s	305 MB	11,048.4	0.00%
$Dijk \rightarrow Dijk_{Fast}$	12,574,429	1m 33s	313 MB	11,048.4	0.00%
$Dijk \rightarrow A^*$	5,497,456	1m 20s	305 MB	11,175.1	1.14%
$Dijk \rightarrow PRM$	3,589,619	55s	298 MB	11,883.1	7.55%

Table A.1: (44° 12' N, 107° 51' W) to (44° 34' N, 107° 8' W)

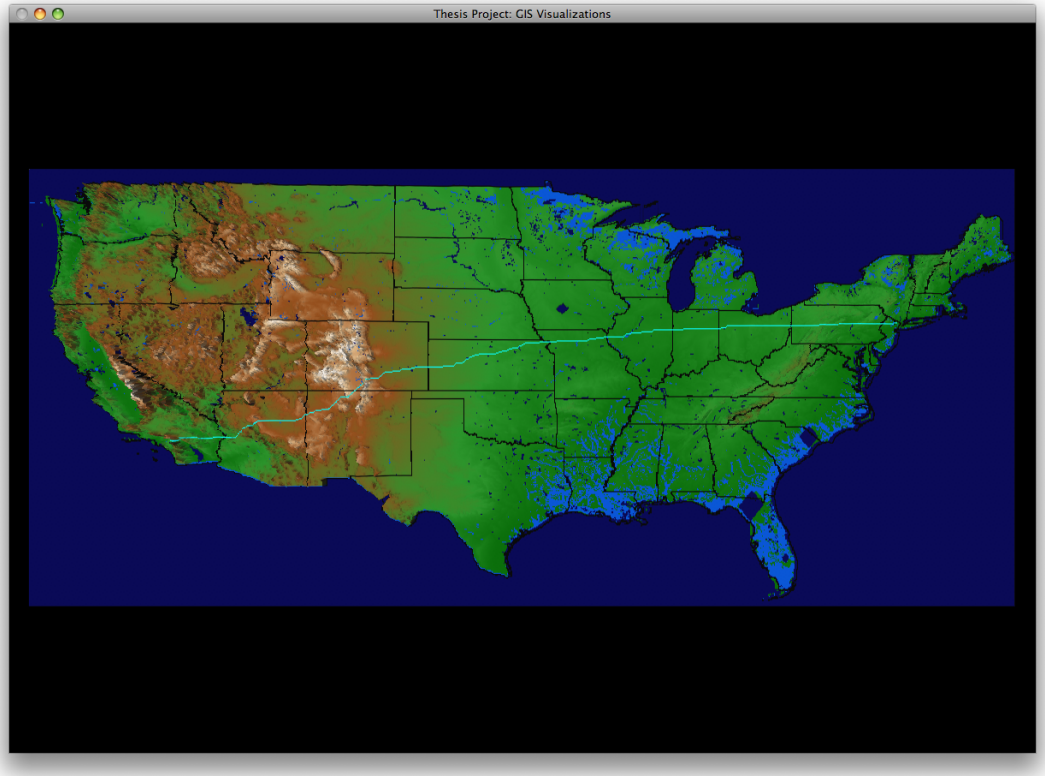


Figure A.2: An energetic path from Los Angeles to New York City found on the simplified dataset.

With Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	983,413,860	4h 18m 10s	1.90 GB	450,699	0.98%
$Dijk \rightarrow Dijk_{Fast}$	989,423,828	2h 33m 31s	1.94 GB	450,559	0.95%

Without Buffer

Method	Nodes	Runtime	Memory	Cost	Error
$Dijk \rightarrow Dijk$	654,000,000	2h 37m 16s	1.88 GB	455,122	0.00%
$Dijk \rightarrow Dijk_{Fast}$	655,200,408	1h 32m	1.95 GB	454,994	0.00%

Table A.2: Results for path computation between Los Angeles (33° 56' N, 118° 24' W) and New York City (40° 47' N, 73° 58' W)

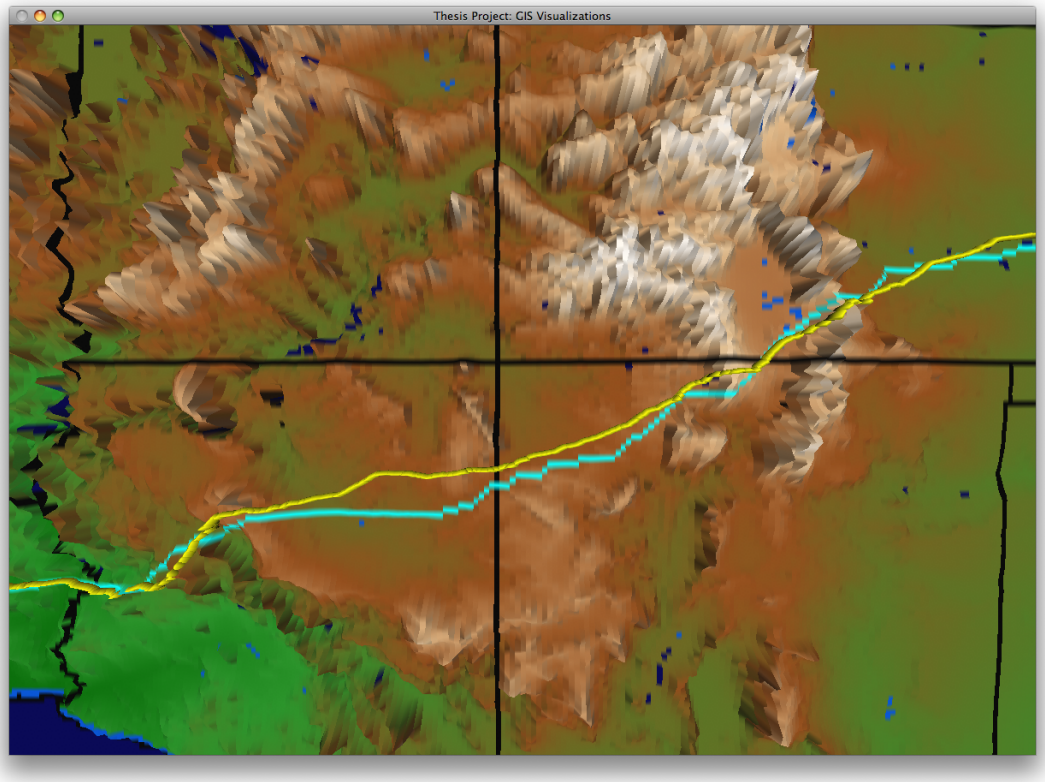


Figure A.3: A closer view of the simplified (cyan) and detailed (yellow) paths from Los Angeles to New York City as they cross a mountainous region.