A Piecewise-planar Method for Modelling Terrain and Finding Shortest Paths

Carsten Maple

Department of Computing and Information Systems, University of Luton, U.K.
carsten.maple@luton.ac.uk

Abstract

Much research has been undertaken in the area of modelling geographical data. In this paper we are concerned with modelling terrain using elevation data. Applications of terrain modelling include temporal changes to terrain, terrain visualisation and the analysis of terrain for features such as fault lines. One key application for terrain modelling is its use in optimal path planning for mobile robots over natural, nonhomogenous terrain. In this paper we describe methods for the representation of the topography in a digital terrain model (DTM). We describe a number of terrain modelling techniques and provide algorithms, implementations and results for finding optimal paths over nonhomogeneous terrain.

Keywords: Terrain modelling, path planning, planar approximations, geometric modelling, surface approximation, robot motion planning.

1. Introduction

The availability terrain data is increasing rapidly along with a massive growth of geographic information systems (GIS). There exist a number of both freely and commercially available resources for terrain data. Free sources of data include the United States Geological Society (USGS) and the National Aeronautics and Space Administration (NASA).

The USGS provides data in various formats: Digital Elevation Model (ASCII DEM); Spatial Data Transfer Standard (SDTS DEM); National Elevation Dataset (NED); and Global 30 Arc Second Elevation Data Set (GTOPO30 DEM). NASA provides files in the formats including Shuttle Radar Topography Mission (SRTM) and Mars Orbiter Laser Altimeter (MOLA). The National Geophysical Data Center (NGDC), part of the National Oceanic and Atmospheric Association (NOAA), runs a project that provides access to 30-arc-second (1-km) gridded, global DEMs. The project is named the Global Land One-kilometer Base Elevation (GLOBE) Project and is considered one of the best publicly available topographical data sources. The NGDC also provide data in the form of TerrainBase, a more general database over an 8 kilometer nominal grid.

The data with which we are primarily concerned is in the DEM (digital elevation model) format. This is now being superceded by SDTS but the basic DEM data can be extracted from STDS files. The reason for not using STDS is that they are highly complex and a single data set consists of over 20 files of data. DEM data are arrays of regularly spaced elevation values referenced horizontally either to a Universal Transverse Mercator (UTM) projection or to a geographic coordinate system. The grid cells are spaced at regular intervals along south to north profiles that are ordered from west to east. The elevations are given in decimal and whole units of meters and feet (with the exception of 1-degree DEM, which is given in whole meters only.)

The accuracy of DEM data is dependent upon the source and resolution of the data samples. Comparing linear interpolation elevations in the DEM with map location elevations gives the accuracy of DEM data. The root-mean-square error (RMSE) is used to describe the DEM accuracy.

1.1. Elevation Maps

The data for a DTM is most often given in an elevation map. There are essentially two major methods for storing mapped information - those that use vector-based storage and those that use raster-based storage. Vector-based storage involves storing points, lines and polygons and are highly accurate. Raster-based storage is more common and involves storing map features in raster or grid format. The location of features is generalised to a regular matrix of cells. Data is normally given over an equidistant or regular grid of elevation values. It contains the three dimensional coordinates of points on the ground. An elevation map can be expressed in a two-dimensional array. Each element of the array corresponds to a Cartesian coordinate in a plane. The entries in the

array each correspond to a height or elevation of the terrain or surface at a point of certain latitude and longitude. Typically the map array has a value stored in map[i][j] that corresponds to the elevation of the surface at (x_0 + i*dx, y_0 + j*dy), and hence there exists a point in three-dimensional space (x_0 + i*dx, y_0 + j*dy, map[i][j]) that lies on the surface. For regular or equidistant grids dx = dy. The approximation to the surface is not normally given in an elevation map. That is, the vertices in the approximation are given but the method of connecting these data points is not described. It is practice that this is left to the interpreter of the data, and they may choose a range of techniques including, for example, a piecewise planar approximation or a spline approximation.

1.2. Triangular Irregular Networks

The TIN model was developed over 30 years ago as a simple way to construct an approximation to a terrain or surface using a set of irregularly spaced points. Typically the TIN is adapted to the specific terrain, with many points in areas of rough terrain and few in smooth terrain.

The data points in a TIN points are usually connected by lines to form (non-overlapping) triangles giving a piecewise planar approximation to the surface or terrain. The approximation is continuous since each triangle is defined by the elevations of the three corner points. The TIN is not unique for a given set of data points as the connections between data points must be chosen. The model is attractive due to its simplicity and economy and works particularly well in areas with sharp breaks in slope. A special type of triangulation, Delaunay triangulation can often provide smooth approximations to a terrain.

If the data is given in the form of a TIN, it is typically given as a series of tuples of coordinates. Each tuple contains the three coordinates (in three-dimensional space) of the vertices in the TIN. Another data structure used to represent a TIN is that which presents a list of all vertices in the DTM and provides an adjacency list. The adjacency list is constructed using a binary relation ~ such that v1 ~ v2 □ there exists an edge in the DTM with v1 and v2 as endpoints. The two representations of the DTM are equally valid and there exists a one-to-one relationship between the two. To convert one representation to the other is a trivial operation.

2. The Boundary Cubes Method

Both TINs and elevation maps are useful in describing terrains, and recent work has seen the development of using boundary cubes to represent surfaces and terrains. This representation gives rise to a highly regularised graph that can be used for efficient path planning and a data structure that allows consideration of further terrain characteristics. In 1987 Lorensen and Cline [4] proposed the marching cubes algorithm, a surface generation technique used to produce 3D images given point data from 3-space. The method requires the data to be given on orthogonal grids over parallel planesThe distance between the planes is the same as the grid mesh size in each plane. Data is given at each of the vertices of every square in every slice. This pixel data is transformed into values of either 0 or 1; a 0 corresponding to a vertex lying outside the surface, a 1 corresponding to all other vertices. Marching cubes has been seen to have a wide variety of applications and has again been modified to combat domain specific problems, see for example [2], [3], [7], as well as been used as the basis for generating piecewise-planar representations of 3D objects, see [1], [5], for example.

The traditional data structure for this type of information is a standard three-dimensional array, where the value of the pixel, 0 or 1, at position (i, j, k) is stored in position [i, j, k] of the array. There is a simple mapping from a pixel-wise representation to a marching cube representation. Two consecutive slices, k and k-1 say, are used to define a layer of marching cubes. Four neighbouring pixels that form a square from each of the two slices comprise a cube. The marching cubes algorithm forms a piecewise-planar approximation to an object. The boundary of the object is assumed to pass exactly half way between a pixel that is exterior to the boundary of an object and a neighbouring pixel that is not. This determines the way in which the surface intersects a cube, if indeed it does. There are 8 vertices to each cube, each of which can be outside the surface of the object or not. There are, therefore, 256 different ways in which a cube can intersect (or not) the surface. In order to uniquely identify each of the 256 ways weighting value is assigned to each vertex of a cube.

Once a Marching Cubes approximation of a terrain has been obtained, it is possible to extract the boundary, see [6]. Extracting this boundary gives a piecewise-planar approximation to a surface. The representation of the approximation is in the form of a non-simple graph.

There exist simple algorithms to translate representations between the three discussed here. In this paper we will discuss how data from a TIN can be converted into an elevation map. Given an elevation map we then present the method for converting this data to a boundary cubes representation. Finally we discuss how the boundary cubes algorithm may be used for robot motion planning.

Given a triangular irregular network we can convert the data into an elevation map in a fairly straightforward manner. We assume without loss of generality the data is given as a series of tuples, each describing a triangle in the network.

This method effectively places an orthogonal grid over the TIN and changes the initial zero values into the corresponding height of the triangle it covers. This provides an elevation map for the terrain.

To change from an elevation map to a boundary cubes method, the task is simple if we first convert the data into a series of data points over parallel orthogonal grids. We already have an initial grid size in our elevation map.

The boundary cubes method uses more memory than an elevation map method which in turn uses more memory than a TIN. This is due to the fact that they are successively coarser approximations to the terrain.

3. Path Planning

There have been a number of works that examine robot motion planning by considering terrain maps, see for example [8], [9], and [10]. Algorithms for path planning rely on constructing a tree or graph representation of the terrain. A tree is a graph in which each node has only one parent. Many of the path planning algorithms are indeed modified version of general shortest path finding algorithms in a tree or graph. Both the TIN and Boundary Cubes approaches give rise to graph structures.

In this work we will describe a path planning algorithm for terrain approximated by a boundary cubes method. Once we have a boundary cubes representation, we effectively have a representation that forms a labelled digraph. The labelled digraph corresponds to a piecewise planar approximation to the terrain with some regularity not found in TINs. Each node in a boundary cubes representation graph has a value of between 1 and 254 that denotes the surface-cube interaction. Each cube has six faces and if the approximation of the surface at this cube intersects a face, then the surface passes through this face; this can be used to determine possible routes out of that cube. Path planning algorithms exist that can be used to find optimal routes through the graph. Optimal routes may be defined, for example, to be those of minimum Euclidean length, those with minimum number of bends or a combination of both.

The basic step of exploring a graph is generating a successor of a node. A node is said to have been *expanded* when all its successors have been generated. Nodes that have been generated and that are awaiting expansion are called *open*. Nodes that have already been expanded are called *closed.*

The most common technique for finding minimal paths through a graph is Dijkstra's shortest path algorithm. The algorithm operates by visiting nodes or vertices in the graph beginning at the start vertex. It then considers the closest not yet examined vertex adding its vertices to the set of vertices to be examined. The algorithm repeatedly runs this operation, expanding outwards from the start vertex until it encounters the goal. The algorithm is guaranteed to find a (possibly non-unique) optimal path.

The best-first search algorithm is also widely used. Whereas Dijkstra's algorithm begins at the start vertex and expands out, thereby minimising the distance away from the start, the best-first algorithm attempts to minimise the distance to the goal from the current vertex. It uses an estimate or heuristic to estimate the distance a vertex is from the goal.

3.1. A-Star Search (A*)

The A* search algorithm is a "greedy" graph-search algorithm that uses a heuristic to find the shortest path between two graph vertices. It combines the better elements of Dijkstra's algorithm and the best-first search. "Greedy" search algorithms do not usually guarantee that an optimal path will be found. A* however is guaranteed to find the shortest path between the start and goal vertices provided an admissible heuristic is used. An admissible heuristic is one that never overestimates the distance to the goal vertex. The A* algorithm keeps a priority queue of vertices yet to be processed and a list of vertices for which a, possibly non-optimal, path has already been determined. These lists are normally referred to as the open and closed list respectively. The main element within the algorithm is a while loop that tests whether there are further vertices in the open list awaiting processing. While this condition is true the algorithm fetches the vertex having the lowest priority from those in the open list. The measure of priority is determined by the cost, cx, incurred in getting to a particular vertex, vx from the start vertex vs, and an estimate of the "distance" to the goal vertex, vg, as measured by the heuristic, h. Hence we can define an estimated cost $f = c(vx) + h(vg)$. In the absence of a heuristic, the A* algorithm operates a Dijkstra's algorithm. The algorithm guarantees finding an optimal path if one exists. Each time a vertex, vcurrent, is removed from the open list, all vertices adjoining vcurrent are added to the open list. Checking the values of the vertices in the open and closed lists determines whether a shorter path to vcurrent has already been determined. If a shorter, or equal, path already exists, vcurrent is discarded and the next

vertex on the open list is examined. If a shorter path does not exist, vcurrent is added to the closed list. Each time a vertex is added to the closed list, information about the parent vertex is added to vcurrent so that a path can be subsequently reconstructed. There are two possible ways in which the algorithm can terminate. Either vg has been reached and the shortest path has been found or the open list is found to be empty before vg has been reached and therefore no path exists.

The 'cost' of each move is normally derived solely from some previously estimated distance between two vertices. When determining the 'best' path across a given terrain the shortest, in terms of distance, path may not necessarily be optimal for the given application. An acceptable route for a wheeled vehicle such as a robot would be unlikely to contain steep inclines for example. The 'difficulty' in traversing particular types of terrain needs to be considered in the determination of an 'optimal' route. Measuring 'difficulty' in a consistent and quantifiable way requires additional data and additional, possibly non-trivial, calculations, but is made easier in the framework of boundary cubes. Each cube value can be associated with a cost that is relevant for that surface-cube interaction. Finding an appropriate cost function depends on the study at hand and is beyond the scope of this paper. In this paper we use a cost function that is based upon the cost of an elevation of a distance *h* vertically is three times as costly as travelling a distance of *h* horizontally.

3.2. Heuristics

Heuristics are principles for deciding which course of action should be chosen from a set of possibilities in order to be most effective in achieving some goal. A heuristic, *h(n)*, is said to be *consistent* or *monotone*, if it satisfies the triangle inequality $h(n) \leq w(n, n') + h(n')$ for every node n and every successor n' of n, where $w(n, n')$ is the weight of the edge from n to n'.

Defining the heuristics may not be simple in all applications; however, in applications such as robot motion planning simple heuristics can be easily defined. An example of an appropriate heuristic would be the Euclidean distance from the current vertex (x_n, y_n, z_n) to the goal vertex (x_g, y_g, z_g). The Euclidean distance, or L_2 norm is given by $\sqrt{[(x_n - x_g)^2 + (y_n - y_g)^2 + (z_n - z_g)^2]}$. The algorithm does not guarantee a return of an optimal path, but is generally faster than Dijkstra's algorithm due to the reduced number of vertices in the graph that require expansion.

A common heuristic is the Manhattan or city block heuristic. This is effectively the L_1 norm and is given by $|x_n - x_g| + |y_n - y_g| + |z_n - z_g|$. Another common heuristic is the maximum distance heuristic, derived from the L_∞ norm and is given by the maximum value of the set $\{|x_n - x_g|, |y_n - y_g|, |z_n - z_g|\}$.

4. Results

An A* algorithm written in C++ has been developed. The input to the algorithm is digital elevation model of a terrain, a start point and an end point. The heuristic used in the results presented here is the Manhattan or city-block heuristic. Data was taken from the USGS (for the city of Folsom, California) and a marching cubes approximation formed. The binary file used in the tests has a file size of 15,725,270 bytes. The nodes (boundary cubes) can be classified by their branching degree (the number of adjacent cubes) as in Table 1.

Branching degree	Number of nodes
0	0
1	0
2	17
3	132920
4	406485
5	96726
6	18832
Total	654980

Table 1: Characteristics of the boundary cubes approximation to elevation data for the city of Folsom, CA.

5. Conclusions

In this paper we have discussed methods for terrain approximation that can be used to find shortest paths. An A* algorithm has been implemented and produces results in a very short time frame when using the Manhattan heuristic (a graph of approximately 5×10^6 nodes in 2.3 seconds). The boundary cubes algorithm has an underlying structure that allows the number of nodes expanded to be much smaller than using otyher techniques, such as Dijkstra's algorithm.

Further work on this project includes the application of the algorithm to a large set of real examples using data obtained from the USGS. An analysis of the algorithm is currently underway that compares the method to other known techniques for path planning over heterogeneous terrain.

6. References

[1] Hannaford, G., Maple, C., 2003, Determining Candidate Binding Site Locations Using Conserved Features, in *Proceedings of IEEE International Conference on Information Visualisation* (IV2003).

[2] Heiden, W., Goetze, T., Brickmann, J., 1993, Fast generation of Molecular-Surfaces from 3D Data Fields with an Enhanced Marching Cube Algorithm, *Journal of Computational Chemistry*, 14, 2, 246-250.

[3] Levitt, D.G., Banaszak, L.J., 1992, Pocket - A Computer-Graphics method for Identifying And Displaying Protein Cavities and their Surrounding Amino-acids, *Journal of Molecular Graphics*, 10, 4, 229-234.

[4] Lorensen,W.E., Cline, H.E., 1987, Marching Cubes: A High Resolution 3D surface Construction Algorithm, *Computer Graphics*, 21, 4, 163-169.

[5] Maple, C., 2003, Using the Boundary Cubes Algorithm for Terrain Modelling and Robot Motion Planning, in *Proceedings of 9th IEEE International Conference on Methods and Models in Automation and Robotics*, (MMAR2003)

[6] Maple, C., Wang, Y., 2004, A Three-dimensional Object Similarity Test Using Graph Matching Techniques, in *Proceedings of IEEE International Conference on Information Visualisation*, (IV2004)

[7] Nagae, T., Agui, T., Nagahshi, H., 1993, Orientable closed Surface Construction from Volume Data, *IEICE Transactions on Information Systems*, 76, 2, 269-273.

[8] Pai, D.K., Reissell, L.-M., (1998) Multiresolution Rough Terrain Motion Planning, IEEE Transactions on Robotics and Automation, **14**, 19-33.

[9] Petzold, I., et al, Network Planning using Geomorphology, in Proceedings of 9th ACM international symposium on Advances in Geographic Information Systems (GIS2001), 167 – 172.

[10] Sun, X., Reif, J., (2003) On Energy-minimizing Paths on Terrains for a Mobile Robot, accepted for publication in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2003).

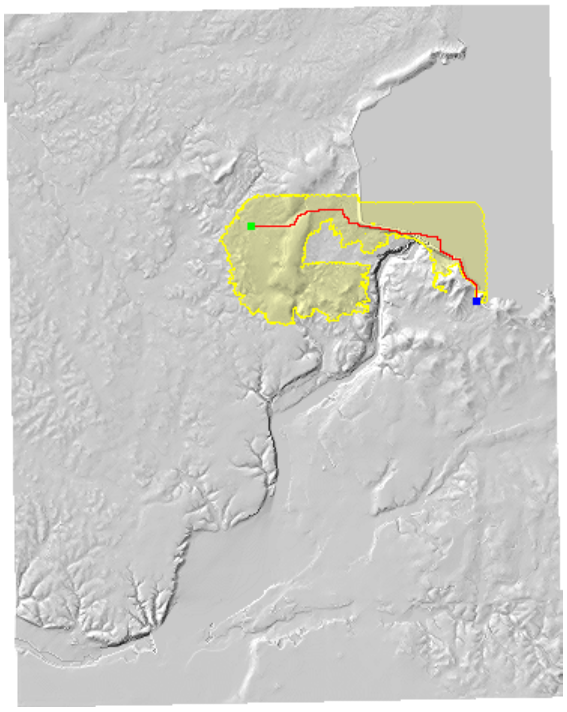

Fig. 1: A* algorithm with Manhattan (city-block) heuristic (30973 nodes expanded in 2.30 seconds)