10-1-2012

Procedural generation of road networks for large virtual environments

Craig Martek

PROCEDURAL GENERATION OF ROAD NETWORKS FOR LARGE VIRTUAL ENVIRONMENTS

by

Craig Martek

Submitted to the faculty of

Rochester Institute of Technology

in partial fulfillment of the requirements for the degree of

Master of Science

B. Thomas Golisano College of Computing and Information Sciences

Department of Computer Science

Rochester Institute of Technology

October 2012

ROCHESTER INSTITUTE OF TECHNOLOGY

GRADUATE COMMITTEE APPROVAL

submitted by

Craig Martek

Date	Zack Butler, Ph.D., Chair
Date	Richard Zanibbi, Ph.D., Reader
Date	Roxanne Canosa, Ph.D., Observer

ABSTRACT

PROCEDURAL GENERATION OF ROAD NETWORKS

FOR LARGE VIRTUAL ENVIRONMENTS

Craig Martek

Department of Computer Science

Master of Science

Much work has been done in recent years involving the generation of virtual landscapes, namely for applications in video games and movies. Some recent video games feature worlds upwards of 400 mi^2, and they continue to expand. Manual design of these landscapes can be extremely time consuming. However, recent approaches have shown that reasonable scenes could potentially be generated procedurally with little to no input from a human designer. Many of the individual pieces necessary for generating a large, complex terrain populated with cities have been investigated, but efforts to piece them together have been minimal so far. A system is presented to build a reasonable hierarchical road network when provided with positional and size information about cities as well as a map of the terrain. Experiments are included to compare this system to existing real world road networks in the United States.

ACKNOWLEDGMENTS

First and foremost, Dr. Zack Butler and Dr. Richard Zanibbi are deserving of more thanks than I know how to give, for being more patient with me than should be reasonably expected of any human being.

I would also like to thank my family and friends, whose support helped me maintain my sanity throughout college.

Finally, I would like to thank my dog for giving me someone to bounce ideas off of, even if he never did understand me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Procedural worlds

As virtual worlds increase in size and complexity, generating them manually becomes very complicated and time-consuming. As shown in Fig. 1.1, video game maps have been expanding rapidly in recent years. Procedural world generation aims to solve this problem by creating these worlds with only minor external input. Terrain, buildings, city roads, and even entire cities have been generated in this fashion. Both the terrain [2] and city [3] levels of the procedural world generation problem have been thoroughly investigated.

Terrain generation can be divided into a number of subproblems. The land itself must be generated, along with water features and vegetation. Land is typically created by generating a heightmap, a map that contains an elevation value for each point. This generation is usually done with some form of fractal noise as a base. The models can be further improved by simulating erosion and other natural processes. Heightmaps in their basic form cannot represent some types of terrain features like caves, however. This is addressed in some

recent works like that of [4], which presents a volumetric structure for storing terrain data.

City generation is also a multiple stage problem. With many approaches, the first step is identifying where to place buildings. This is typically done by laying out a network of roads and identifying plots sectioned off by the network. In the most simple forms, building footprints can be placed in these lots and random heights used to generate monolithic structures. In some more complex methods, the buildings themselves can be procedurally generated in a number of fashions. Some approaches are even more complex like that of [5], which simulates a city's expansion over time.

1.2 Procedural roads

Specifically relevant to the global road network problem is the problem of generating networks of roads within a single city. In [6], tensor fields are interactively created and edited to design the road network. This is done in three stages. In the first, the user designs some base tensor fields with a drawing interface. Next, the street graph is created by using the hyperstreamlines of this tensor field. Finally, the graph is used to create a full three-dimensional model of the city. This method is able to create both grid and radial patterns, and can also generate roads following natural barriers.

A grammar-based system called an L-system is used in [7] to generate a city network of roads. L-systems are a recursive string rewriting system in which a set of rules is defined to modify some start string. In this approach L-systems are used to create a generic template for the road, which is then modified with respect to some goals such as population density and general overall patterns.

The template is also checked by a set of constraints to ensure that no invalid roads are generated. This approach is able to effectively handle changes in the parameters, even the general pattern used to guide the generation.

The work presented in [8] first generates a set of primary roads, which are filled in with secondary roads to create city blocks in a manner similar to that of [7]. Each road is created with a high level graph representing control nodes, or a general path to which a road should adhere. A lower level graph is then computed that adapts to both the control nodes of the higher level graph and changes in the environment. In this work, these adaptations only take elevation into account. Once the major roads are created, secondary roads expand inwards from the containing cells using an L-system with a number of user-defined parameters. Some of the ideas used in these city-level approaches can be applied to large scale road networks, particularly the common distinction between major and minor roads.

Another road generation system is described in [9]. Instead of typical city roads, however, this method generates roads similar to those found in the informal settlements of South Africa. These settlements grow on unused land near industrial zones, and so roads tend to be placed where they are most beneficial. Voronoi diagrams are used to generate the major road pattern with centers that are random, regular, or generated by an L-system. L-systems are also investigated as minor road generators and compared to simple subdivision with noise. When compared to real world examples, the subdivision with noise seems to perform best as a minor road generation method. This appears to indicate that minor roads develop along simple patterns, or at least in this context.

Some of the approaches for road networks across larger terrain involve some

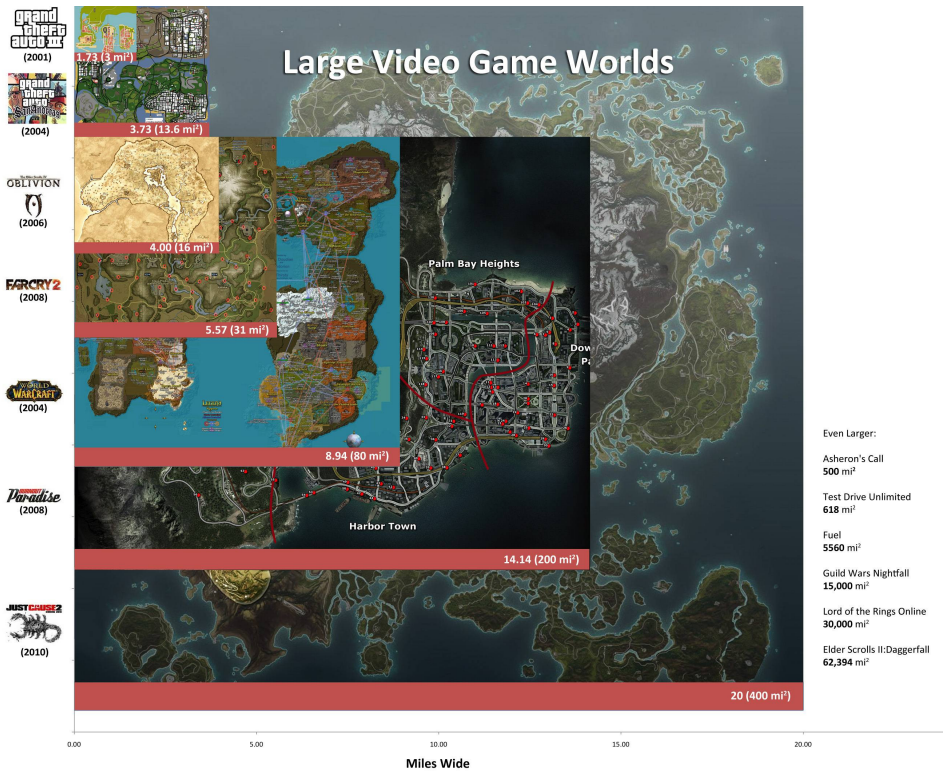


Figure 1.1: Comparison of large video game worlds from [1]

interaction with or significant input from a user. In [10], features and terrain are first sketched by a user and then fit procedurally into the surrounding map. The landscape and elevation are designed using a colored grid, while roads and other smaller features are created with standard drawing tools. Changes to the world around these features can cause the features themselves to be automatically modified. This system produces particularly good results with a moderate amount of direction.

In [11], the roads and rivers are generated from vector data provided to the system. The world can be edited by the user, which in turn modifies these underlying vector structures. Quadtrees form the underlying storage for this approach, making updates to the map efficient. A template based system is devised in [12], in which the user can divide the map into regions and for each choose what template will be used to generate the contained roads. The system attempts to split each of the regions into approximately equal chunks based on the population density of the area.

Generating an optimal single road across three-dimensional terrain can generally be considered a weighted shortest path problem, where weights can include aspects such as slope, terrain features, and population density. The cost is usually considered to be anisotropic, meaning that it is dependent on both position and direction. The problem is also generally considered on a continuous domain and has to be discretized in some fashion before it can be solved. The approach proposed in [13] first constructs a uniform grid out of the map and computes discrete paths across this grid. Local shifts are then performed on this path, as the original path may not be an optimal one.

A three-dimensional version of the anisotropic shortest path problem is considered in [14]. The approach here considers a polyhedral surface divided

into triangles, with each triangle having some weight. The cost function used is simply the distance traveled across a face multiplied by the triangle's weight. Instead of a simple grid, the triangles are discretized based on Steiner points. The graph used for generating the shortest path is based on these Steiner points. This is a particularly attractive solution for shortest path across virtual terrain, which is typically represented and drawn as a set of triangles.

Many different approaches to procedurally generating roads for a single city have been developed, but this is only one piece of the world generation problem. Modern video games can contain a multitude of cities, all of which should be connected by a network of roads. It also stands to reason that other procedural worlds with a set of cities should have a road network connecting them. Generation of road networks from the ground up has not been thoroughly investigated, as in real-world applications many of the roads already exist. Some research does exist, however, such as the work presented in [15] which aims to generate a network of rural roads. To do this, a greedy randomized adaptive search procedure, or GRASP, is used on the obvious underlying graph representation. The system seeks to maximize both connectivity and efficiency with this algorithm. However, this only produces the towns that should be linked, and would require more complete calculation of the shortest paths between each pair of towns.

Another prevalent issue with constructing road networks is optimizing their traffic flow. On a good road network, passengers should be able to get from their origin to their destination with minimal traffic. One study of freeway congestion pinpointed inefficient operation as the root cause of congestion issues [16]. Ramp metering, the process of controlling entry onto freeways, is one popular answer to this problem. This method was combined with vari-

able speed limits in [17] and showed decent improvements over networks with no flow control systems. Flow control can also be used to evaluate a road network. The capacity reliability approach presented in [18] uses a method based on network flow to define a performance measure that includes travel time reliability, connectivity reliability, and uncertainty analysis. This particular problem is out of the scope of this work, but integration with procedural modeling methods could certainly improve the effectiveness of the results.

The problem of road generation for virtual environments is a relatively recent problem, so a new system is presented here to create these networks. A recently devised shortest path algorithm is combined with minimum spanning trees and rapidly expanding random trees, along with the use of a population metric. This system produces road networks that appear more realistic and correspond to roads in real world examples more closely than other methods like the template system in [12].

Chapter 2

Shortest Path Algorithm

The problem of procedurally generating reasonable roads over given terrain has been recently investigated in [19]. This paper presents a method for generating the shortest viable road between two arbitrary points of some input scene. Many conditions must be satisfied for a road to be viable, so a simple shortest path algorithm is not sufficient. Here an anisotropic shortest path algorithm is devised, meaning that the cost function used depends on the first (p') and second (p'') derivatives in addition to position (p). In the continuous sense, this cost function is represented as:

$$C(\rho) = \int_0^T c(p(t), p'(t), p''(t))dt \tag{2.1}$$

where ρ is a function mapping $[0, T]$ to a path between the two specified points. Because of the complexity of this problem, a discrete approximation is used instead. The region is divided into a grid of points in which one point is connected to any other point within some distance instead of just those adjacent to it. This mask, M_k is defined as the set of all points $(i, j) \in [-k, k]^2, GCD(i, j) = 1$ connected to the origin (0,0). Once this graph is built, an A^* search is employed to find the actual shortest path. In this search, the

heuristic used is the straight line distance between the two points.

The cost functions defined in [19] are also quite useful. The overall cost function is defined as:

$$c(p, p', p'') = \sum_{i=0}^{i=n-1} \mu_i \circ \kappa_i(p, p', p'') \tag{2.2}$$

where κ is a set of functions evaluating different properties of the terrain and μ is a set of functions that weight the influences of each aspect of the evaluation functions. The characteristics used in this approach are slope, water depth, curvature, and amount of vegetation. For example, the curvature function could be defined as the radius of curvature at a point p:

$$c(p, p', p'') = \frac{|1 + p'^2|^{3/2}}{|p''|} \tag{2.3}$$

The transfer function of this function could simply be the inverse, indicating that smaller radii implicate a higher cost for the road. Other parameters are even simpler; slope could be defined as the change in slope between two points, and vegetation as the density of vegetation at a particular point. Transfer functions for these could be as straightforward as a linear function with some maximum value, as higher numbers for either of these parameters indicate higher cost for that particular point. Additional functions allow this system to handle bridges and tunnels.

Once the shortest path is found, it is first segmented into piecewise clothoid splines in order to smooth it out. These splines are then segmented and labeled as road, tunnel, or bridge to indicate the type of graphics that should be used. Finally, any terrain modifications necessary to create these surfaces, such as leveling ground or removing vegetation, are performed and the roads themselves are created by modeling them as clothoid splines. A clothoid spline,

also known as a Euler spiral, is simply a curve for which the curvature is linearly proportional to its arclength. These splines are particularly useful for modeling roads, as they help to eliminate problems caused by centripetal force. The points generated as a path for each road are used as the control points for that road's spline, creating a realistic and efficient road.

Chapter 3

Approach

3.1 Overview

The system constructed here builds a network of major and minor roads given at minimum some set of cities and an elevation map. This provides a virtual terrain with a convincing set of paths that could be used to create three-dimensional road geometry. Water, foliage, and population information can also be included if it is desired that they have some influence on the shortest path algorithm.

The purpose of any road network is to provide an efficient means of transportation between population centers at a low cost. Although roads vary in size and type, they can be divided at a high level into two categories: major roads and minor roads. Interstates and large highways are good examples of major roads. Their primary goal is to connect the larger concentrations of population, namely large cities. Minor roads are the other highways and smaller roads. Their purpose is to connect to the smaller cities and towns and feed the major roads.

3.2 Shortest Path

This method is built primarily off of the discretized anisotropic shortest path algorithm presented in [19]. For the large scale problem of generating a road network, parts of this can be simplified. A single pixel could represent a very large candidate area, so curvature constraints are not necessary. Instead of user defined scaling functions, a simple linear scale is used for each of the other cost functions based on a maximum value specified by the user. Population was added as an additional cost function. To achieve an attracting effect on a road, the population at a point can reduce the cost of a road going through it.

One additional modification was made in order to allow for parallel searches. A thread-safe variable containing the minimum cost value for a road in the current search is maintained along with the A^* search so as to prune any roads that might overshoot that cost. When adding a new point to the priority queue, if the overall cost to that point is greater than the minimum cost value, the cost associated with that point is set to infinity. This effectively ignores any path that might include that point. If the search returns without reaching the goal point, no path exists below that cost requirement.

3.3 Major Roads

Major roads are generally more costly, and therefore tend to connect only major cities. In this approach, a modified version of Prim's algorithm is used to build the minimum spanning tree across the major cities. The parallelized algorithm works as follows:

1. Select a random major city as the start of the tree.

2. Calculate the shortest path from every city not currently into the tree to a city in the tree. The minimum cost variable is used here to keep track of the lowest cost road found so far, in order to avoid unnecessary searches.

3. Add the road and city with the shortest path into the tree.

4. Repeat steps 2 and 3 until all major cities exist in the tree.

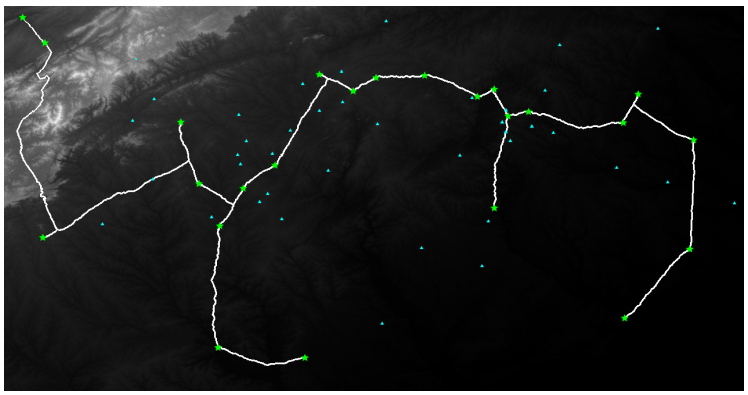
Further detail can be found in Algorithm 1. In addition to the elevation, water height, and foliage cost functions, an optional population cost function was included. Each point on the grid can have a population, typically associated with a nearby city. If there is population at a point, the cost of a road passing through it is reduced by some weighted amount. This has the effect of pulling the major roads towards the minor cities if they pass close enough. If a major road happens to pass through a minor city through this process, it is marked as part of the tree. This effect can be seen in Fig. 3.1. No population weight was used to generate the roads in Fig. 3.1a, so the placement of the minor cities has no effect on the major roads. In Fig. 3.1b, the areas where minor cities pulled the major roads toward them have been highlighted in red. One of these areas has been enlarged in Fig. 3.2.

3.4 Minor Roads

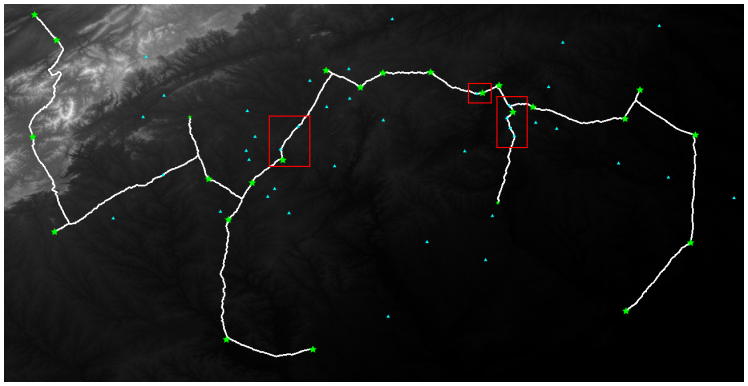
Two separate approaches were tested to generate the minor roads. In the first approach, each minor city is iteratively connected to the existing road network by finding the shortest such connection at each iteration and adding it. This approach is efficient, but does not necessarily generate a convincing network.

noend 1 Minimum Spanning Tree (Major Roads)

```
major ← list of major cities
taken ← ∅
current ← city with minimum distance from any other city
remove current from major, add to taken
while major ≠ ∅ do
    minCost ← ∞
    minRoad ← null
    for all point p ∈ current road network do
        for all city c ∈ major do
            if shortest path between c and p < minRoadCost then
                minCost ← shortest path cost
                minRoad ← shortest path
    remove new city from major, add to taken
    if minRoad connects to any minor cities, mark them as connected
    add minRoad to road network
```

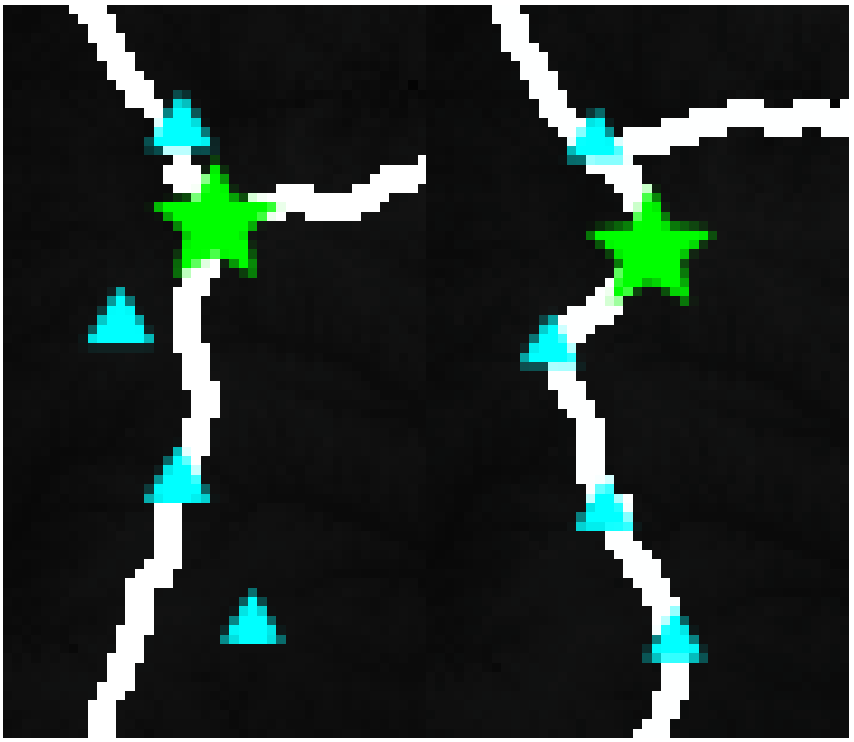


(a) No population effect



(b) Population pulling

Figure 3.1: Effect of population pulling. Affected areas highlighted in red.



(a) Zoomed in w/o population (b) Zoomed in with population

Figure 3.2: Zoomed in area affected by population pulling. Major roads were pulled towards the minor (blue) cities.

The second approach uses a rapidly exploring random tree to build the minor network of roads. Rapidly exploring random trees were introduced in [20] as a basic path planning mechanism. The tree is constructed by selecting points in the search space at random and connecting it to the closest point on the existing tree. Once a maximum cost, c, and number of branches per iteration, n, are specified, the rapidly-exploring random tree algorithm for minor city connection works as follows (detailed more in Algorithm 2):

1. Connect any minor city where the cost of the shortest path to a point on the network is less than c.

2. Repeat step 1 until no more cities are added. If there are no more minor cities to add, the algorithm is done.

3. Randomly select n points on the grid where the shortest path to the network is less than c, and add those roads to the network.

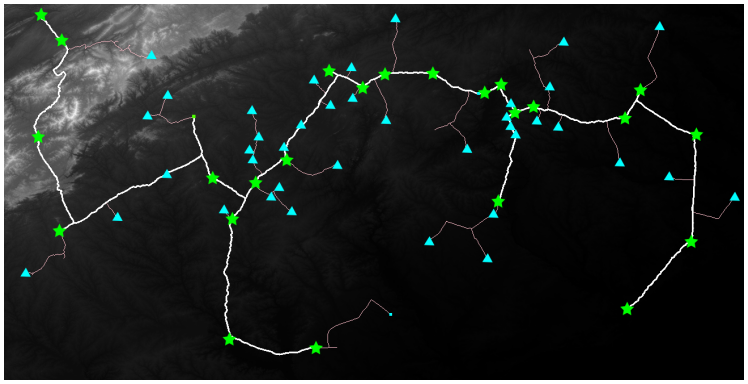
4. Repeat from step 1.

noend 2 Rapidly Exploring Random Tree (Minor Roads)

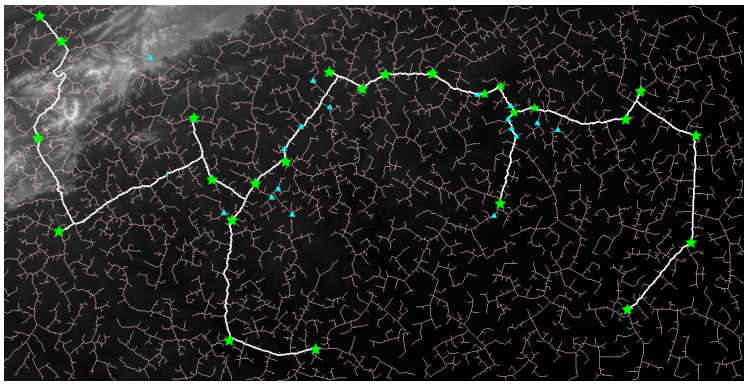
Require: roads, a list of all roads in the major network; maxCost, the maximum cost of any minor road; numRoads, the number of roads to generate in each iteration

```
minor ← list of minor cities not already connected
candidate ← ∅
while minor ≠ ∅ do
  for all city c ∈ minor do
    minRoad ← shortest path between c and any point in candidate or actual road network
    if minRoad < maxCost then
      add minRoad to roads, remove from candidate
      search the endpoints until a major road is found
      add each of the roads in this path to roads, remove from candidate
    if no cities were successfully connected to the network then
      for i = 1 → numRoads do
        p ← random point within maxCost from the network (including candidate roads)
        add shortest path to candidate
```

This tree can then be pruned to remove segments that are not part of a connection to any of the minor cities. Doing so creates a more straightforward network, while leaving the nonessential roads makes the network appear fuller. Fig. 3.3 demonstrates this difference using a simple pruning approach of removing road segments that are not part of a path from a major road segment to a minor city. This could be made more robust by using more com-



(a) Pruned



(b) Unpruned

Figure 3.3: Pruned vs. unpruned tree

plex pruning methods. For example, with a more detailed population map, pruning could be done by only removing roads that do not approach larger concentraions of population.

In both the naive and RERT method, each minor road is built in a manner similar to the major road construction. A parallel search is done over the set of points existing on the road for the lowest cost path, using the same type of short-circuiting if a road with a lower cost has already been found.

Chapter 4

Experiments and Discussion

4.1 Experiments

All experiments were run on a 2.67 GHz Core i7 (quad-core) machine with 12 GB of RAM. One set of tests was run on real world elevation data, using information from the USGS elevation web service [21] and population data from the 2010 US Census. Other tests were run using new elevation data generated with the L3DT software [22]. C# and the parallel language-integrated query extension (PLINQ) were used for efficient and simple parallelization (see code in Appendix A). In these experiments, eight threads were available for finding the shortest-path, but it would be trivial to expand given the resources. Maps were output as text files indicating locations of the roads as well as a graphical map for easy viewing.

The two real world areas used for testing were an approximately 300x200 mile region consisting of most of New York and northern Pennsylvania and a 400x200 mile region containing most of North Carolina as well as parts of Tennessee and South Carolina. A mask value of 2 was used in the shortest

path computations so as to avoid large jumps between points, since the actual distance between two adjacent points on the grid could be hundreds of meters. The mask essentially determines how many pixels an individual road segment can span. Major roads were generated with and without cost reduction from population, and the naive approach was compared with various parameter sets for the rapidly exploring random tree approach.

Both the slope and water height functions were used in the shortest path cost function. In addition, a population function was used in some cases. Slope was defined simply as the absolute height difference between the two points being compared, while the water height function took the average of the water heights at both points. If population was considered, the population of a city was first expanded as a distribution stretching some specified distance away from the city (in pixels). For some distance d, each pixel within d pixels of a city c received the population:

$$pop(d) = pop(c) * \left(\frac{1}{2}\right)^{distance(d,c)} \tag{4.1}$$

Each of these functions were modified with a linear transfer function defined by a maximum threshold. Any cost over that threshold then became infinite, and no path would be considered over that point. This can be represented as, for some cost c:

$$c_{new} = \begin{cases} \frac{c_{old}}{c_{max}} & \text{if } c_{old} < c_{max} \\ \infty & \text{otherwise} \end{cases} \tag{4.2}$$

Each cost was then weighted and summed along with a simple distance metric, as in Equation 2.2, to obtain the overall cost of that particular road segment. The distance metric was necessary, as each set of points in the mask

used in the shortest path algorithm will not necessarily be the same distance from each other. Each of these weights and maximum values was varied in order to determine which values produced the best results. Precision was calculated as the percentage of generated road segments that fell within one kilometer of a real world counterpart; recall was calculated as the percentage of real world road segments that were within one kilometer of a generated segment. For each region, a control network of major roads was built with a simple minimum spanning tree across the major cities for comparison.

Specifically, every combination of the following parameters was generated:

1. Maximum slope change: 10m-90m in 20m steps (NY), 20m-70m in 10m steps (NC)
2. Population spread (pixels): 2-5
3. Maximum population per pixel: 2000-5000
4. Slope weight: 1-3
5. Population weight: 0-3

For each data set, the most precise major road network was selected for analysis of minor road generation. Minor roads were generated with varying parameters for number of roads per iteration and maximum cost per segment. Twenty-five maps for each combination were generated due to the non-deterministic nature of the rapidly exploring random tree. A control map was also generated, in which each minor city was connected to the road network by adding a single road with the smallest cost to an exisiting node. These maps were compared against more maps obtained from OpenStreetMap with

	New York		North Carolina	
	Precision (1 km)	Recall (1 km)	Precision (1 km)	Recall (1 km)
Control	25.5%	3.75%	19.4%	4.77%
Average	33.0%	5.28%	17.9%	4.73%
Standard deviation	5.60%	1.05%	2.48%	0.714%

Table 4.1: Control vs. average precision and recall, major roads

minor roads included. A new map was generated for each combination of the following parameters:

1. Number of roads per iteration: 5, 10, 15, 20
2. Maximum road cost: 25, 50, 75, 100

4.2 Results

4.2.1 Major roads

The general results for precision and recall of major road generation can be seen in Table 4.1, in which the control maps that were generated with a simple shortest path algorithm are compared with the ones generated from this system. The major roads generated for the New York terrain map showed significant improvement compared to their control counterpart, while the generated roads for the North Carolina data set were slightly less precise than the control map. Recall for both sets stayed at approximately the same level across all runs.

Max. change in slope (m)	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
10	46.6%	1.44%	4.59%	0.0834%
30	42.5%	0.570%	7.04%	0.0950%
50	30.9%	1.77%	4.94%	0.390%
70	29.1%	0.399%	4.50%	0.0502%
90	29.6%	0.497%	4.65%	0.107%

Table 4.2: Maximum slope experiment, NY data

Max. change in slope (m)	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
30	19.7%	1.56%	5.23%	0.487%
40	19.4%	0.786%	5.18%	0.243%
50	19.2%	2.22%	5.10%	0.639%
60	14.4%	1.47%	3.75%	0.426%
70	16.7%	0.299%	4.39%	0.0876%

Table 4.3: Maximum slope experiment, NC data

Tables 4.2 and 4.3 show the results of changing just the maximum slope parameter in the shortest path algorithm. Decreasing the maximum slope noticably improved precision in the New York maps and also appeared to boost the recall to a lesser extent. The North Carolina data set showed little variation in both precision and recall with the exception of a dip in both when the maximum slope change was set to 60 meters per pixel.

Tables 4.4 and 4.5 show the results of varying the population parameters in the cost calculation method. Little variation was observed when changing

	New York				North Carolina			
Population spread	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
0	31.5%	6.05%	5.05%	1.14%	17.5%	1.43%	4.59%	0.467%
2	32.9%	5.72%	5.23%	1.07%	17.7%	2.46%	4.68%	0.713%
3	33.0%	5.68%	5.29%	1.07%	18.1%	2.46%	4.78%	0.702%
4	33.2%	5.61%	5.31%	1.05%	17.8%	2.52%	4.70%	0.719%
5	33.2%	5.57%	5.31%	1.05%	18.0%	2.61%	4.77%	0.753%

Table 4.4: Population spread experiment

either of these parameters in both the New York and North Carolina data sets.

4.2.2 Minor roads

The overall precision and recall results for minor roads is shown in Table 4.6. On average, the RERT method of generation had worse precision than the naive method, and approximately equal recall.

The effects of varying the number of roads per iteration can be seen in Tables 4.7 and 4.8. In both data sets, changing the number of roads generated in each iteration did not have an effect on the precision or recall.

The effects of varying the maximum cost of the roads is shown in Tables 4.9 and 4.10. In the New York data set, increasing the maximum cost of the roads produced significantly better precision, although the recall stayed approximately the same. Increasing the maximum cost also improved the precision in the North Carolina data set, although to a much smaller degree.

	New York				North Carolina			
Maximum population	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
0	31.5%	6.05%	5.05%	1.14%	17.5%	1.43%	4.59%	0.467%
2000	32.9%	5.72%	5.26%	1.04%	17.4%	2.61%	4.59%	0.764%
3000	33.0%	5.68%	5.28%	1.07%	17.7%	2.56%	4.68%	0.734%
4000	33.1%	5.26%	5.29%	1.06%	18.2%	2.41%	4.83%	0.686
5000	33.3%	5.69%	5.31%	1.07%	18.2%	2.37%	4.83%	0.675%

Table 4.5: Maximum population experiment

	New York		North Carolina	
	Precision (1 km)	Recall (1 km)	Precision (1 km)	Recall (1 km)
Control	52.6%	6.45%	35.4%	5.88%
Average	45.3%	6.49%	30.5%	5.87%
Standard deviation	3.71%	0.254%	1.37%	0.264%

Table 4.6: Control vs. average precision and recall, minor roads

Roads per iteration	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
5	45.4%	3.77%	6.50%	0.238%
10	45.1%	3.69%	6.45%	0.263%
15	45.4%	3.79%	6.52%	0.262%
20	45.4%	3.65%	6.48%	0.252%

Table 4.7: Roads per iteration experiment, NY data

Roads per iteration	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
5	30.6%	1.31%	5.86%	0.259%
10	30.3%	1.41%	5.85%	0.246%
15	30.5%	1.44%	5.88%	0.278%
20	30.4%	1.33%	5.89%	0.276%

Table 4.8: Roads per iteration experiment, NC data

Maximum cost	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
25	41.3%	2.32%	6.46%	0.247%
50	44.6%	2.50%	6.41%	0.231%
75	46.8%	2.74%	6.52%	0.247%
100	48.6%	2.52%	6.57%	0.265%

Table 4.9: Maximum road cost experiment, NY data

Maximum cost	Precision (1 km)	Standard deviation	Recall (1 km)	Standard deviation
25	29.7%	1.31%	6.02%	0.271%
50	30.2%	1.23%	5.91%	0.255%
75	30.8%	1.24%	5.83%	0.234%
100	31.2%	1.23%	5.71%	0.195%

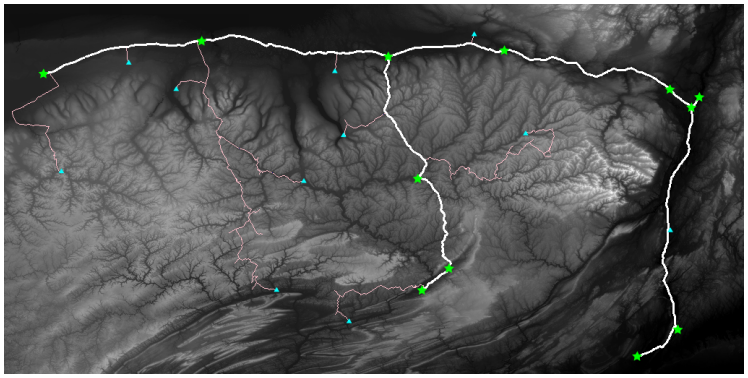
Table 4.10: Maximum road cost experiment, NC data

4.3 Discussion

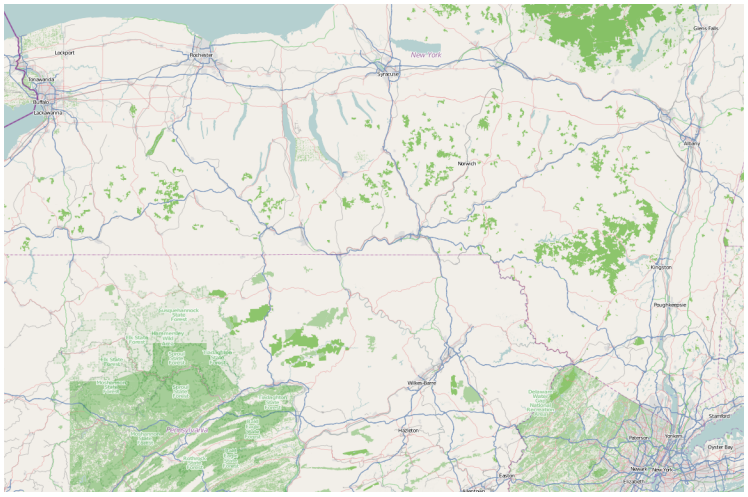
The minimum spanning tree approach for generation of major roads produced some reasonable results. In most of the New York tests, the system replicated a significant portion of Interstate 90, the largest road in the region. A number of other roads also saw similar replication in both the New York and North Carolina testing regions. Fig. 4.1 demonstrates the similarities between the major roads generated for the New York/Pennsylvania region and the actual road network of the area. Fig. 4.2 shows a generated network of roads overlaid on an actual map of the region obtained from OpenStreetMap [23]. Note the similarities in the major roads, particularly Interstates 90 and 81. Some of the minor roads even loosely mimic the real world example despite being randomly generated.

A similar outcome can be seen in the overlay in Fig. 4.3, a sample network generated over most of North Carolina and some of South Carolina and Tennessee. Some major roads, such as I-26 and I-77, are closely replicated in some areas. In other areas, the major roads in the generated map are quite different than those in the real map. This is likely caused by the simple population model employed; the split between major and minor cities is determined solely by a population threshold.

The RERT approach for generating minor roads created minor road networks that were almost universally more interesting than those generated with the naive approach. An example of this can be seen in Fig. 4.4. Both of these networks were built on top of the same artificially generated map and set of cities. The RERT approach can be seen to break up roads into segments that more closely resemble real life roads.



(a) Generated



(b) OpenStreetMap

Figure 4.1: Comparison between generated roads and actual

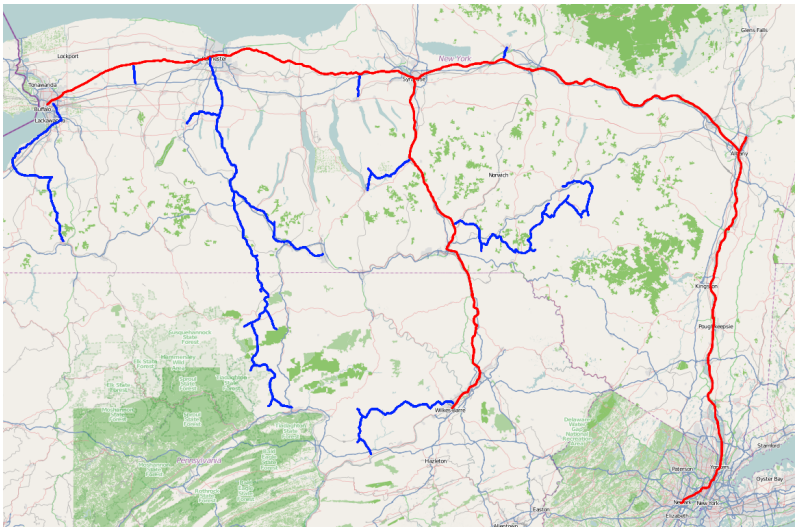
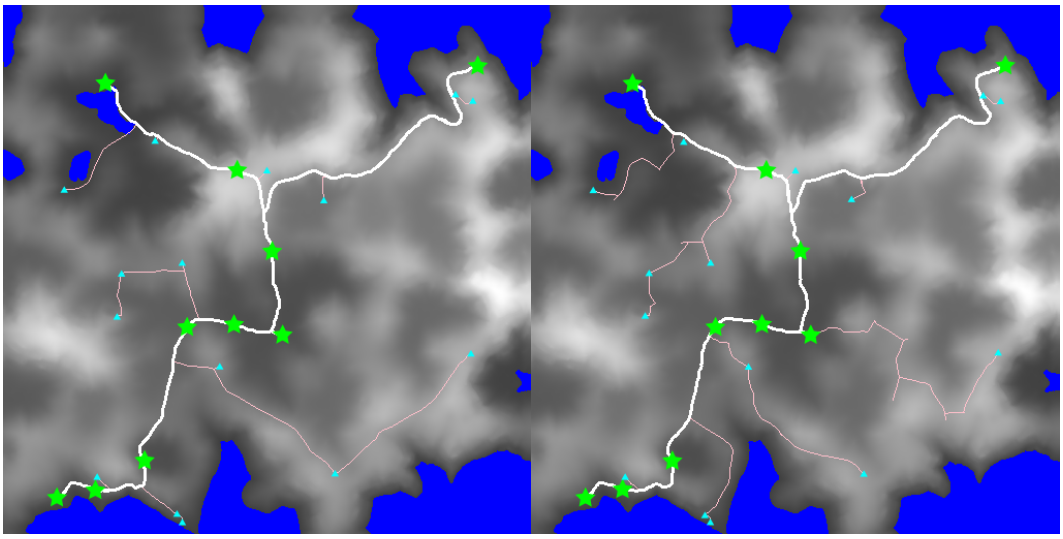


Figure 4.2: Overlay of generated roads over OpenStreetMap, NY data set

Figure 4.3: Overlay of generated roads over OpenStreetMap, NC data set

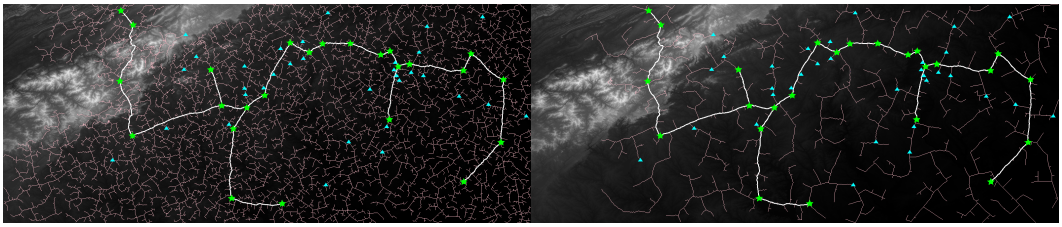
This approach uses only two parameters to build the rapidly exploring random tree, but changing these wildly varied the output of the generated minor roads. Trees with a longer minimum road length tended to generate a smaller number of long, spaced out roads. Conversely, those with a shorter minimum length resulted in denser regions of shorter roads. The other parameter controlled the number of roads to generate in each iteration. A larger number here results in a denser network, while smaller numbers lead to sparse networks. These parameters combined give the user significant control over the output without requiring too much tweaking. Examples of this can be seen in sample networks generated on the North Carolina data set. In Fig. 4.5, the effects of varying the minimum length are clear. A relatively small minimum length was used in Fig. 4.5a, making the network much denser than the larger value used in Fig. 4.5b. In Fig. 4.6, only the number of roads to generate each iteration was varied. A smaller number created the dense network seen in Fig. 4.6a, while a larger number created the sparser network in Fig. 4.6b.

The precision and recall results from the major road generation suggested that the performance of the algorithm depends on the general layout of the terrain. In a mountainous area like New York, roads are generally built to conform to valleys and whatever other flat segments may exist in the area. This characteristic made the slope parameter have more effect on the precision of the generated roads. Of note, there was also a spike in recall when the maximum change in slope value was set at 30 meters, as this parameter produced a road that followed I-90 very closely. In North Carolina, where the terrain is much flatter, slope changes have less impact on where the roads were built. The population parameter seemed to have a negligible effect on the precision of the roads in both data sets, but it seems likely that this is due to the large number



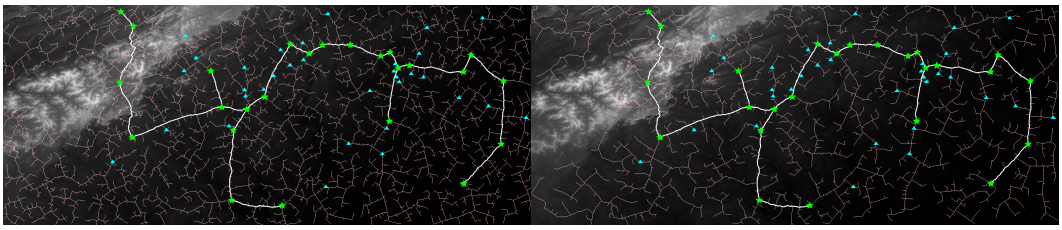
(a) Naive approach (b) RERT approach

Figure 4.4: Comparison between naive approach and rapidly exploring random trees



(a) Smaller minimum length (b) Larger minimum length

Figure 4.5: Comparison between small minimum length and large minimum length



(a) Smaller number of roads per iteration (b) Larger number of roads per iteration

Figure 4.6: Comparison between small and large number of roads per iteration

of roads being generated away from population centers in these simulations. If a more sophisticated population model were in place, this parameter would likely have a stronger positive impact on results.

In minor road generation, using the RERT method with a simple pruning algorithm performed worse than a naive approach. This is likely due to the unnatural angles created by the random trees, as they become particularly pronounced after pruning. However, using no pruning generates far too many roads and produces a very cluttered map. Using a more intelligent pruning method would almost certainly improve the precision and recall of the generated networks.

Varying the number of roads built in a single iteration had no effect on either data set. This suggests that it is not important to have deep trees of minor roads as opposed to shallow ones. Visually, the two extremes of this parameter produce quite similar results. Changing the maximum cost of each minor road did have an observable impact though, especially in the New York data set. In both cases, increasing the maximum cost improved precision while having little effect on recall. Thus it appears that a smaller number of long roads is more accurate when compared to real world examples than a larger number of small roads.

Recall was persistently low across all simulations, as the minor road pruning method generates significantly fewer roads than are seen in the real world examples. However, not pruning the minor roads at all would result in a far too dense map that would look much less convincing. A smarter pruning algorithm could certainly improve both the precision and recall of these experiments.

4.4 Future Work

In its current form, this method only generates roads for a single large scale. It could potentially be adapted to provide a full hierarchical road network model, by starting at the largest scale and progressively zooming in to refine the previous model. With this method, an entire world of cities could be efficiently connected by roads with minimal input from a world designer. It would also be interesting to note if the described population parameters would have more of a noticeable effect on smaller-scale terrain maps. If the model took into account existing roads, this could be particularly helpful for plotting out new roads in a developing area.

The system would also benefit from use of a general population model instead of being based on connecting cities. In this way, the approach could first locate major centers of population that should be connected, and do so in much the same way that the major road approach takes here. The minor roads could then be constructed based on proximity to the smaller clusters of population. This could be combined with a more intelligent way of determining which cities are major and which are minor, as well as some sort of traffic modeling as an additional cost function to create a more realistic network.

Real world road cities tend to be more connected by the road network than

those in the networks generated by this system. This could be overcome by using a GRASP system like that of [15] to determine which major cities to connect instead of using a minimum spanning tree approach. In the minor road generation, more connectivity could be created by allowing the system to select points already on the road instead of just those outside of it. As discussed in Chapter 1, flow control could also be introduced to help evaluate and improve the effectiveness of a generated network.

Once the road network is generated, creating a three dimensional model could work exactly like the clothoid spline method used in [19].

Appendices

A Code

A.1 Shortest Path

```
/**
 * Generates the shortest path between two points.
 * Modified form of the A* algorithm found on Wikipedia.
 *
 * isMajor - true if a major road should be generated, false if minor
 * useMinRoad - true if using the minimum road variable to prune expensive ↩
     roads
 *
 * Requires minRoadCost, a static thread-safe variable
 */
private Road ShortestPath(Point source, Point goal, bool isMajor, bool ↩
    useMinRoad) {
  // If the straight line distance is greater than the existing minimum,
  // there's no reason to go through all this setup
  if (useMinRoad && Distance(source, goal)*DistanceWeight > Thread.↩
      VolatileRead(ref minRoadCost)) {
    return new Road();
  }
  Dictionary<Point, Point> predecessors = new Dictionary<Point, Point>();
  Dictionary<Point, double> g_score = new Dictionary<Point, double>();
  Dictionary<Point, double> h_score = new Dictionary<Point, double>();
  PriorityQueue<Point> pq = new PriorityQueue<Point>();
  g_score[source] = 0;
  h_score[source] = Distance(source, goal);
  pq.Enqueue(h_score[source], source);
  // A*
  while (pq.Count > 0) {
    KeyValuePair<double, Point> top = pq.Dequeue();
    // shortcut out if this isn't going anywhere
    if (top.Key == Double.PositiveInfinity
      || top.Key > Thread.VolatileRead(ref minRoadCost)
      || top.Value.Equals(goal)) break;
```

```
    foreach (Point p in MaskPoints(top.Value)) {
      GridInfo gi = grid[p.X, p.Y];
      if (!pq.Contains(p)) pq.Enqueue(Double.PositiveInfinity, p);
      if (!g_score.ContainsKey(p) || g_score[top.Value] +
          Cost(grid[top.Value.X, top.Value.Y], gi) < g_score[p]) {

        predecessors[gi.Location] = top.Value;
        g_score[p] = g_score[top.Value] + Cost(grid[top.Value.X, top.Value.Y↩
            ], gi);
        h_score[p] = Distance(gi.Location, goal);
        pq.ChangePriority(p, g_score[p] + h_score[p]);
      }
    }
  }
  // return infinite road cost
  if (!predecessors.ContainsKey(goal)) return new Road();
  // else reconstruct path
  List<Point> path = new List<Point>();
  path.Add(goal);
  Point current = goal;
  while (!current.Equals(source)) {
    current = predecessors[current];
    path.Insert(0, current);
  }
  double cost = 0;
  for (int i = 0; i < path.Count - 1; i++) {
    cost += Cost(grid[path[i].X, path[i].Y], grid[path[i + 1].X, path[i + ↩
        1].Y]);
  }
  if(useMinRoad) {
    if (cost >= Thread.VolatileRead(ref minRoadCost)) return new Road();
    Thread.VolatileWrite(ref minRoadCost, cost);
  }
  return new Road(path, isMajor, cost);
}
```

A.2 Minimum Spanning Tree

```
public void MinimumSpanningTree() {
  List<City> major = cities.Where(n => n.IsMajor).ToList<City>();
  List<City> taken = new List<City>();
  List<KeyValuePair<Point, City>> mst = new List<KeyValuePair<Point, City↩
      >>();
  if (major.Count == 0) return;
  // select the city closest to any other as our start
  City current = major.SelectMany(x => major
              .Select(y => new KeyValuePair<City, City>(x, y)))
            .Where(x => !x.Key.Equals(x.Value))
            .OrderBy(x => Distance(x.Key.Location, x.Value.Location))
            .First().Key;

  Console.WriteLine("Start city is {0}", current.Name);
  major.Remove(current);
  taken.Add(current);
  while (major.Count > 0) {
    Console.WriteLine("{0} major cities left...", major.Count);
    minRoadCost = Double.PositiveInfinity;
    Road minRoad = taken.Select(x => x.Location)
              .Concat(roads.SelectMany(x => x.Path))
              .SelectMany(x => major
                .Select(y => new KeyValuePair<Point, City>(x, y)))
              .AsParallel().OrderBy(x => Distance(x.Key, x.Value.Location))
              .AsParallel().Select(x => ShortestPath(x.Key, x.Value.Location↩
                  ))
              .AsParallel().OrderBy(r => r.Cost).First();
    minRoad.EndPoint = major
        .Where(x => x.Location.Equals(minRoad.Path[minRoad.Path.Count - 1]))
        .First();
    if (taken.Any(x => x.Location.Equals(minRoad.Path[0]))) {
      minRoad.StartPoint = taken.Where(x => x.Location.Equals(minRoad.Path↩
          [0]))
                .First();
    } else {
      minRoad.StartPoint = roads.Where(x => x.Path.Contains(minRoad.Path[0])↩
          )
                .First();
    }
```

```
    major.Remove((City)minRoad.EndPoint);
    taken.Add((City)minRoad.EndPoint);
    minRoad.Finalized = true;
    roads.Add(minRoad);
    cities.Where(x => !x.IsMajor &&
              minRoad.FilledInPath.Any(y => x.Location.Equals(y)))
        .ToList().ForEach(x => x.IsConnected = true);
  }
}
```

A.3 Rapidly Exploring Random Tree

```
public void RERT(double maxCost, int numRoads) {
  Console.WriteLine("{0} cities already connected",
          cities.Where(x => !x.IsMajor && x.IsConnected).Count());
  List<City> minor = cities.Where(x => !x.IsMajor && !x.IsConnected).ToList↩
      ();
  List<Road> candidateRoads = new List<Road>();
  while (minor.Count > 0) {
    Console.WriteLine("{0} minor cities left", minor.Count);
    bool modified = false;
    List<City> toRemove = new List<City>();
    foreach (City c in minor) {
      minRoadCost = maxCost;
      // attempt to connect directly to current roads
      Road minRoad = roads.SelectMany(x => x.Path)
                .Concat(candidateRoads.SelectMany(x => x.Path))
                .OrderBy(x => Distance(c.Location, x))
                .AsParallel().Select(x => ShortestPath(x, c.Location, false)↩
                    )
                .AsParallel().OrderBy(r => r.Cost).First();
      if(minRoad != null && minRoad.Cost < maxCost) {
        minRoad.EndPoint = c;
        if (cities.Any(x => x.Location.Equals(minRoad.Path[0]))) {
          minRoad.StartPoint = cities
                .Where(x => x.Location.Equals(minRoad.Path[0])).First();
```

```
      } else {
        // find the roads that connect this one to the main network
        Road connect = roads.Concat(candidateRoads)
                  .Where(x => x.Path.Contains(minRoad.Path[0]))
                  .First();
        if (!connect.IsMajor && !connect.Finalized) {
          connect.Finalized = true;
          if (!connect.StartPoint.IsCity()) {
            Road current = (Road)connect.StartPoint;
            while (current != null && !current.IsMajor && !current.↩
                Finalized) {
              current.Finalized = true;
              candidateRoads.Remove(current);
              roads.Add(current);
              if (current.StartPoint.IsCity()) break;
              current = (Road)current.StartPoint;
            }
          }
        }
        candidateRoads.Remove(connect);
        roads.Add(connect);
        minRoad.EndPoint = connect;
      }
      minRoad.Finalized = true;
      roads.Add(minRoad);
      toRemove.Add(c);
      modified = true;
    }
  }
  toRemove.ForEach(x => minor.Remove(x));
  if(!modified) {
    // add some random roads
    for(int i = 0; i < numRoads; i++) {
      Point randP = new Point(rand.Next(width), rand.Next(length));
      // ensure this endpoint isn't on an existing part
      // of the road or another city, or in water
      if(cities.Any(x => x.Location.Equals(randP)) ||
          roads.SelectMany(x => x.FilledInPath).Any(x => x.Equals(randP)) ↩
              ||
```

```
            grid[randP.X, randP.Y].WaterHeight > 0) {

          i--;
          continue;
        }
        minRoadCost = maxCost;
        Road minRoad = roads.SelectMany(x => x.Path)
                  .Concat(candidateRoads.SelectMany(x => x.Path))
                  .OrderBy(x => Distance(randP, x))
                  .AsParallel().Select(x => ShortestPath(x, randP, false))
                  .AsParallel().OrderBy(r => r.Cost).First();

        // make sure they fall within our bounds
        if (minRoad != null && minRoad.Cost < maxCost) {
          if (cities.Any(x => x.Location.Equals(minRoad.Path[0]))) {
            minRoad.StartPoint = cities
                .Where(x => x.Location.Equals(minRoad.Path[0])).First();
          } else {
            minRoad.StartPoint = roads.Concat(candidateRoads)
                          .Where(x => x.Path.Contains(minRoad.Path[0]))
                          .First();
          }
          candidateRoads.Add(minRoad);
        }
      }
    }
  }
  // add any remaining roads to the network,
  // these will appear in the unpruned results
  candidateRoads.Where(x => !roads.Contains(x))
          .ToList().ForEach(x => roads.Add(x));
}
```

Bibliography

[1] Reddit user courageousrobot, "Large video game worlds," http://www.reddit.com/r/gaming/comments/c0p1f/just_cause_2_got_me_thinking_large_video_game/, 2010.

[2] R. M. Smelik, K. J. de Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen, "A survey of procedural methods for terrain modelling," in *Proceedings of the CASA Workshop on 3D Advanced Media in Gaming and Simulation*, Amsterdam, The Netherlands, June 2009.

[3] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *Institute of Technology Blanchardstown Journal*, vol. 14, pp. 87–130, 2006.

[4] A. Peytavie, E. Galin, S. Merillou, and J. Grosjean, "Arches: a Framework for Modeling Complex Terrains," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 28, no. 2, pp. 457–467, 2009.

[5] B. Weber, P. Mller, P. Wonka, and M. Gross, "Interactive geometric simulation of 4D cities," in *Proceedings of Eurographics 2009*, vol. 28, 2009, pp. 481–492.

[6] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang, "Interactive procedural street modeling," in *ACM SIGGRAPH 2008 papers*, ser. SIGGRAPH '08. New York, NY, USA: ACM, 2008, pp. 103:1–103:10. [Online]. Available: http://doi.acm.org/10.1145/1399504.1360702

[7] Y. I. H. Parish and P. Mller, "Procedural modeling of cities," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001, pp. 301–308.

[8] G. Kelly and H. McCabe, "Citygen: An interactive system for procedural city generation," in *Proceedings of GDTW 2007: The Fifth Annual Conference in Computer Game Design and Technology*, 2007, pp. 8–16.

[9] K. R. Glass, C. Morkel, and S. D. Bangay, "Duplicating road patterns in south african informal settlements using procedural techniques," in *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, ser. AFRIGRAPH '06. New York, NY, USA: ACM, 2006, pp. 161–169. [Online]. Available: http://doi.acm.org/10.1145/1108590.1108616

[10] R. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating procedural generation and manual editing of virtual worlds," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10. New York, NY, USA: ACM, 2010, pp. 2:1–2:8. [Online]. Available: http://doi.acm.org/10.1145/1814256.1814258

[11] E. Bruneton and F. Neyret, "Real-time rendering and editing of vector-based terrains," *Computer Graphics Forum*, vol. 27, pp. 311–320, 2008. [Online]. Available: http://hal.inria.fr/inria-00207679/PDF/article.pdf

[12] J. Sun, X. Yu, G. Baciu, and M. Green, "Template-based generation of road networks for virtual city modeling," in *Proceedings of the ACM symposium on Virtual reality software and technology*, ser. VRST '02. New York, NY, USA: ACM, 2002, pp. 33–40. [Online]. Available: http://doi.acm.org/10.1145/585740.585747

[13] Z. Jia and P. Varaiya, "Grid discretization based method for anisotropic shortest path problem over continuous regions," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 4, December 2004, pp. 3830–3837.

[14] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, "Determining approximate shortest paths on weighted polyhedral surfaces," *J. ACM*, vol. 52, pp. 25–53, January 2005. [Online]. Available: http://doi.acm.org/10.1145/1044731.1044733

[15] M. Scaparra and R. Church, "A GRASP and path relinking heuristic for rural road network development," *Journal of Heuristics*, vol. 11, no. 1, pp. 89–108, 2005.

[16] C. Chen, Z. Jia, and P. Varaiya, "Causes and cures of highway congestion," *Control Systems, IEEE*, vol. 21, no. 6, pp. 26 –32, dec 2001.

[17] R. C. Carlson, I. Papamichail, M. Papageorgiou, and A. Messmer, "Optimal mainstream traffic flow control of large-scale motorway networks," *Transportation Research Part C: Emerging Technologies*, vol. 18, no. 2, pp. 193 – 212, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0968090X09000771

[18] A. Chen, H. Yang, H. K. Lo, and W. H. Tang, "Capacity reliability of a road network: an assessment methodology and numerical results," *Transportation Research Part B: Methodological*, vol. 36, no. 3, pp. 225 – 252, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0191261500000485

[19] E. Galin, A. Peytavie, N. Maréchal, and E. Guérin, "Procedural generation of roads," *Computer Graphics Forum*, vol. 7, no. 29, p. 429438, 2010.

[20] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *In*, vol. TR 98-11, no. 98-11, pp. 98–11, 1998. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Rapidly-exploring+random+trees:+A+new+tool+for+path+planning#0

[21] USGS, "Elevation web service," http://gisdata.usgs.net/XMLWebServices/TNM_Elevation_Service.php, 2005, data available from the U.S. Geological Survey.

[22] A. Torpy, "Large 3D terrain generator (L3DT)," http://www.bundysoft.com/L3DT, 2011.

[23] "OpenStreetMap Contributors, OpenStreetMap," http://openstreetmap.org, 2011, cC-BY-SA.