Procedural Content Generation of Villages and Road System on Arbitrary Terrains

Tiago Boelter Mizdal
Programa de Pós-Graduação em Ciência da Computação - PPGCC
Universidade Federal de Santa Maria
Santa Maria, Brazil
tiagomizdal@gmail.com

Cesar Tadeu Pozzer
Programa de Pós-Graduação em Ciência da Computação - PPGCC
Universidade Federal de Santa Maria
Santa Maria, Brazil
pozzer@inf.ufsm.br

Figure 1. A village created with our approach.

***Abstract*— Modeling a terrain with a large amount of details by hand is an arduous task. Creating roads, villages, buildings and other details demands a lot of time from an artist. In this paper we propose a solution that analyzes the terrain, defines proper areas for the villages and generates the roads connecting them. The area delimited to each village is given by the characteristics of the terrain. The road system is generated using the A* algorithm with our own cost functions that consider the slope of the terrain. Our method is also able to create T-junctions and works from a starting point to any other road. Buildings are placed on each village according to a seed and their locations are defined in acceptable areas of the terrain.**

Keywords—Procedural generation; villages; road system; shortest path; weightning function;

I. INTRODUCTION

In many applications, such as video games, simulators and movies, the scenery must contain a vast amount of details to provide good user experience. However, creating all the details manually demands a lot of time and resources. Procedural generation aims at creating data algorithmically, with little to none user input. Modeling landscapes, with rivers, forests, human settlements and roads is a challenging problem. Several procedural modeling techniques are able to generate this kind of content, and they usually require some kind of handmade feature or they have some limitations.

A formula that shapes a village or a road does not exist in the real world. Villages are the result of people settling in convenient locations and roads were created to allow transportation among the villages. Procedural generation of villages and road systems tries to deliver a result that mimics the real life utilizing a pre-defined set of rules and functions.

This paper presents a procedural approach to generate villages and roads connecting them, in a way that permits the result to be easily modified by the user to better suit the application needs. A village generated by our method can be seen in Fig. 1.

The method proposed in this paper has a global to local approach. Given a terrain as input, we create a graph and analyze every node in the grid according to its surroundings, appointing to each node a number indicating how favorable it is to create a city. The nodes act as a seed and the villages are built around them. This way, it is possible to change the location of a village or add a new one by changing or adding a seed node.

The algorithm proposed to create the road system is based on the works of [1] and [4]. It is computed utilizing the A* algorithm, however we created our own cost function that aims at minimizing the standard deviation of the cost of the road. Our approach produces realistic paths that avoid slopes that are too steep and also presenting a way to create roads from a given position to other roads.

The contributions of this paper are as follows:

- We propose a solution to define the center of a village and the area that belongs to it on a given terrain.
- We introduce a method of generating roads connecting a set of cities, utilizing our own cost functions. This method is capable of generating roads from an initial point to any other road in the road system, without a destination point, which creates T-junctions in the process.
- The solutions presented can be easily manipulated to suit different kinds of applications.

This paper is organized as follows: Section II presents related works on the subject, Section III introduces the workflow, Sections IV, V and VI describe the steps used by the method proposed and implementations, Section VII discusses results and Section VIII presents the conclusion and future work.

II. RELATED WORK

Our approach to generate the road network is based on [1], where an algorithm to create complex roads from an initial to a final point that adapts to the characteristics of the terrain is presented. The goal is to build a path that minimizes the line integral of a cost weighting function. It takes in consideration the slope of the terrain and natural obstacles, like rivers, lakes and forests. It is also capable of creating bridges and tunnels. In [4], the paper presents a framework for creating a hierarchical road network connecting a set of cities. However the cities and their sizes must be defined by the user and the roads are connected from a starting point on the edge of a city to a final point on the edge of another, only merging roads that are very similar. Our approach is able to create roads from an initial point to any other road, without the need of a defined final point, which generates T-junctions, this way creating a road system that looks more realistic. Our method also connects cities from their centers.

Reference [2] proposes a local to global approach, capable of creating villages and road networks. In it, a building is set at random on a valid position of the terrain, and then it creates a cycle that consists in adding a road for that building and adding new buildings. It takes into account the slope of the terrain, water, other building, other villages, road connections, types of villages and types of buildings.

CityEngine [5] is a system that addresses the procedural modeling of complete cities using a set of statistical and geographical data. The base model of the city relies on aerial pictures. The roads are created based on L-systems. It generates the road system and the buildings.

There are several papers [12][13] that focus on modeling the land use. Given a terrain description, they generate different patterns of land use, including layouts for different types of areas, sizes, historic background, and density.

III. WORKFLOW

Given a terrain, created by a procedural method, by hand or extracted from real data sets, our solution works on three steps.

First, we analyze the terrain, giving each node on the terrain a value that indicates how probable it is of acting as seed to generate the city around it, and also define which nodes belongs to which seeds. Secondly, we select the nodes that will generate the villages and create a road system connecting them. The road system begins by connecting a point in one village to a final point in another, and then connecting other villages to an existing road. Lastly, we place the buildings on the villages, selecting the acceptable spots where a building can be positioned. Fig. 2 presents the flowchart of our method.

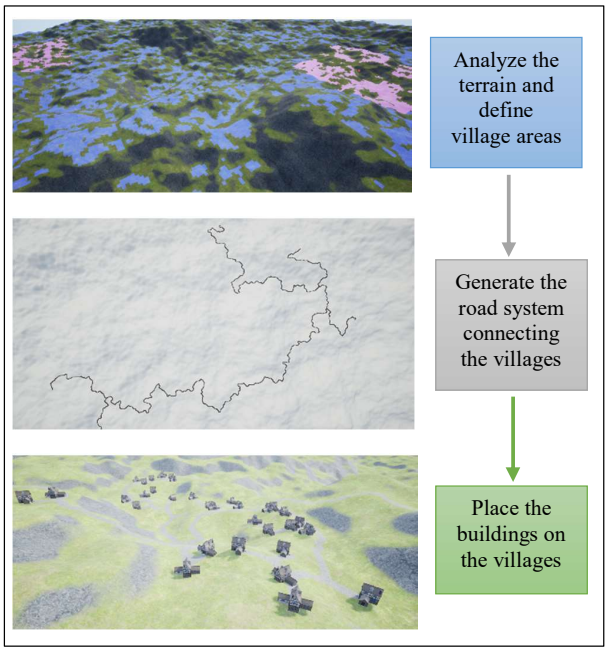


Figure 2. Flowchart presenting the three steps taken by our method to generate the road system and the villages.

IV. FINDING THE LOCATION OF THE VILLAGES

To look realistic, the cities must be placed on locations that are mostly flat, where buildings can be placed near each other and roads can be generated between them. This means that the area in which a village occupy must not have a large subarea where no buildings or roads can be placed. The area that is given to a village must be consistent (Fig. 3). As an example, if a village is near a large gap, buildings belonging to a single village must not be placed on both sides of the gap.

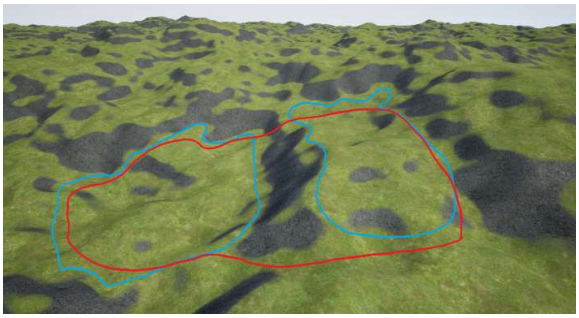


Figure 3. The areas in blue show the proper delimitation, the area in red shows an unfitting approach.

With this in mind, we propose a solution inspired by the algorithms used to calculate the Delaunay Triangulation [6] and the Crust [7].

Delaunay Triangulation is a method used to triangulate a set of vertices. One way to solve it is by creating circles that touch every three vertices of the set. If a circle does not contain another vertex in it, the three vertex touching the circle can be triangulated.

Crust is a curve reconstruction method that given a set of points creates a graph containing the edges of the curve. It creates a circle touching every two points, and if this circle doesn't contain another point or a point from the medial axis, it is a valid edge.

Given the height map, the vertices become a graph, where every vertex is a node. To define the locations of every village, we must first define what nodes on the terrain can be used to place a building. Every node on the terrain is evaluated, considering its slope. Therefore, given a maximum slope, every node is deemed as acceptable or unacceptable. The slope is given by the cross product between the normal vector of the node and the vector perpendicular to the plane XY. Knowing which nodes are acceptable, we can define which ones are more favorable to act as seeds for the villages and which nodes each seed can use. Even the unacceptable nodes can act as seeds, since the nodes that belong to it can be acceptable.

Given a node N and a maximum node count C. We create a circle of radius R centered at N. Then, we select the acceptable nodes that are the farthest in the X and Y axis from N and create new circles of radius R centered on those nodes. Note that these new circles will contain N. Now we count how many acceptable nodes exists that are contained in three or more circles. If the number of counted nodes is less than C, we start the loop again, creating new circles from our last nodes, and counting the number of acceptable nodes. The complexity of this algorithm is $O(n^3)$, but since it is limited by a maximum node count that is usually small compared to the size of the terrain, it still performs fast. The Algorithm 1 provides a pseudo-code of this method and Fig. 4 presents a resulting area delimited for a seed.

```
Input: N ← node being evaluated
Input: C ← maximum number of nodes
Input: R ← radius of the circles
List<Nodes> circles
List<Nodes> selectedNodes
newCircles ← N
while selectedNodes.Num() < C
    newCircles ← findNewCircles(newCircles, R)
    circles.Add(newCircles)
    if newCircles.Num() ≤ 0
        break while
    end if
    selectedNodes ← findNodesIn3orMoreCircles(circles)
end while
return selectedNodes
```

Algorithm 1. Pseudo-code to find the area belonging to a village that can be created by a seed node N.

This way we guarantee that the maximum distance between a node to three other nodes is no larger than R, thus avoiding large gaps and areas without any building. When we have found which nodes belongs to which seed, we give the seed a number P corresponding to probability of success. P is given by the sum of the distances between each node and N.

By changing the values of R and C we can create villages with different densities and sizes, allowing the user to manipulate the way this algorithm behaves. The value of C is proportional to the size of the village. By setting R to a high value, the villages will have large subareas where no building will be placed, creating villages that are sparser.

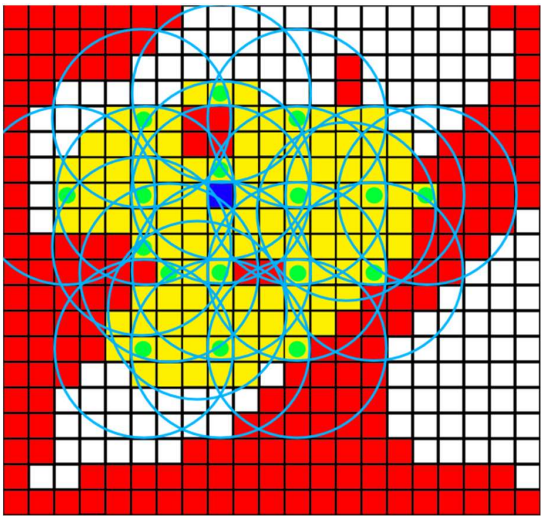
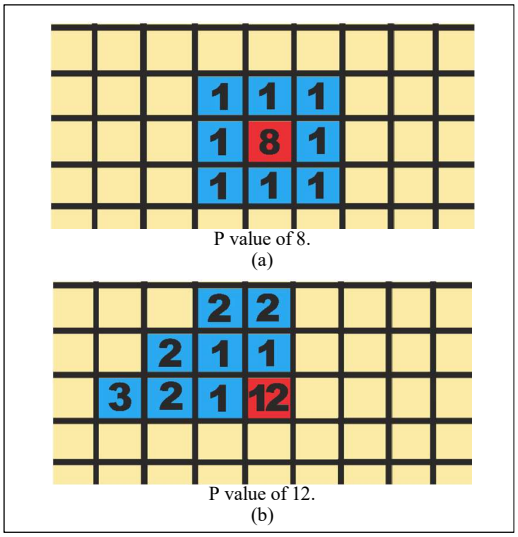


Figure 4. The yellow nodes represent the selected area. The blue node is the seed. The white and red nodes are the acceptable and unnaceptable areas, respectively. The green circles are the center of the expanded circles.

The format of the village is indicated by P. A low valued P means that all nodes found are closer to N, and thus able to create a denser village in an area that resembles a circle. However, a high value for P means that the nodes are sparser, and the village would be placed on a larger area, that adapts to the shape of the terrain. Fig. 5 shows examples of different values of P.



P value of 8.
(a)

P value of 12.
(b)

Figure 5. Both images have the same node count of 8. The seed is in red and the center number is the value of P. The nodes belonging to the seed are in blue, and the number in the center of each node corresponds to the distance between the node and the seed.

V. GENERATING ROADS

The A* algorithm [8][9][10][11] is a best-first graph search algorithm that finds the path from an initial to a final point. We use our own cost function to determine the order the A* searches the nodes. Since the terrain is arbitrary, our cost function must allow the graph to be connected. Only considering steep paths when necessary.

The A* algorithm works as follows. For every point evaluated, we calculate its cost value to the end and its cost value to the parent. Given a starting point p_1 and a goal point p_2, we set all costs of p_1 to zero, and add it to a list of points L. Then we start a loop that does the following: While L is not empty; we select a point p_c in L that has the lowest cost. If p_c is the goal point, we stop the loop. Then we evaluate the costs of all points connecting p_c, set their parent as p_c and add them to L; if the point already existed in L, we only add it if it has a lower cost. Once p_c is equal to p_2, we find the resulting path by following its parents until we reach p_1.

Given the terrain, we create a graph with connectivity of eight. Each vertex is a node and it is connected to its eight closest neighbors, see Fig. 6 below.

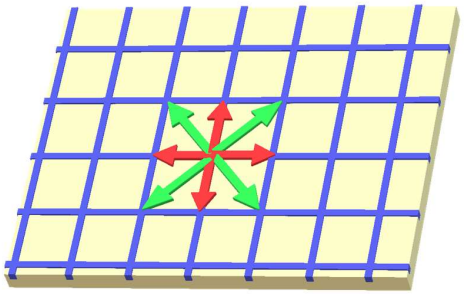


Figure 6. The eight possible connections between all nodes on the grid. The red arrows represent the sideways distance in XY, and the green arrows represent the diagonal distance in XY.

Our cost function between two points p_1 and p_2 takes in consideration the distance in the XY plane between the two points, and the slope α of the first point. The cost function is defined by the distance between p_1 and p_2 plus this same distance multiplied by one minus the slope α multiplied by a constant β, as shown in the following equation.

$$C = DistXY(p_1,p_2) + DistXY(p_1,p_2) \times (1-\alpha) \times \beta \quad (1)$$

This cost function is applied in the cost of a node to its goal and to its parent. The slope α can be pre-calculated and is defined by the dot product between the normal vector of the vertex and the vector perpendicular to the plane XY. We use the distance only in the XY plane because this distance can be pre-calculated, allowing the algorithm to run in less time. The sideways distance to every other vertex is constant and the same occurs on the diagonal distance (see Fig. 6). The value of β affects the impact that the slope has on the path. A pseudo-code example of this method can be seen in Algorithm 2.

All roads created are added to the road system R. In order to create T-junctions, we utilize a small variation of the Algorithm 2. We no longer define a goal point, instead we give the function the start point and all the existing roads. When calculating the cost of a node to the goal point, we calculate the cost of the point to the closest node that belongs to a road. Then, we stop the loop when one of the nodes that were evaluated is contained in R. An example can be seen in Algorithm 3.

```
Input: p1 ← starting point
Input: p2 ← goal point
List<Point> L
List<Point> possibilities
L.Add(p1)
while L.Num() > 0
    select the point P in L that has the lowest cost
    possibilities ← eight closest neighbors of P
    for each Point S in possibilities
        S.parent ← P
        S.costToParent ← DistXY(S, P) + DistXY(S, P)*(1-slope)*β
        S.costToGoal ← DistXY(S, p2) + DistXY(S, p2)*(1-slope)*β
    end for
    if possibilities contains p2
        break while
    end if
    Add possibilities to L
end while
```

Algorithm 2. Procedure to find the best path between two points utilizing our cost function.

```
Input: p1 ← starting point
Input: List<Roads> R ← all existing roads
List<Point> L
List<Point> possibilities
L.Add(p1)
while L.Num() > 0
    select the point P in L that has the lowest cost
    possibilities ← eight closest neighbors of P
    for each Point S in possibilities
        S.parent ← P
        T ← closest node that belongs to R
        S.costToParent ← DistXY(S, P) + DistXY(S, P)*(1-slope)*β
        S.costToGoal ← DistXY(S, T) + DistXY(S, T)*(1-slope)*β
    end for
    if possibilities contains a node in R
        break while
    end if
    Add possibilities to L
end while
```

Algorithm 3. Procedure to create a road giving a start point and a road system.

Given a number of possible village locations, we choose two of these locations to start our road system and create a path between their seed nodes, utilizing algorithm 2. The path to the other villages is then created with algorithm 3, giving their seed nodes and the road system. Fig. 7 shows an example of this.

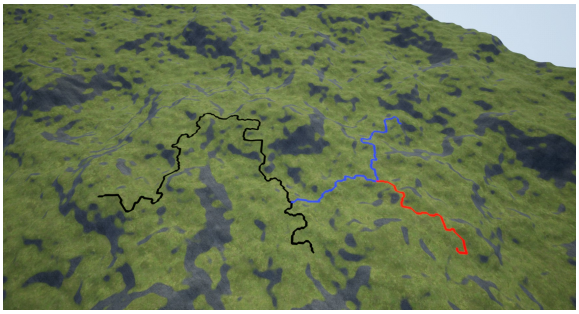


Figure 7. Road system created utilizing our method. The road in black was generated with Algorithm 2. The roads in blue and red were created using Algorithm 3.

A road with a lower standard deviation σ is a path that has less steep parts that deviate from the average slope of the road. By changing β the user can control how the

algorithm behave, creating a path that better suit the user needs. The value of β also affects the standard deviation. This method was extensively tested in order to give an insight about the impact that β has on the road. Fig. 8 shows different roads created with different values of β.

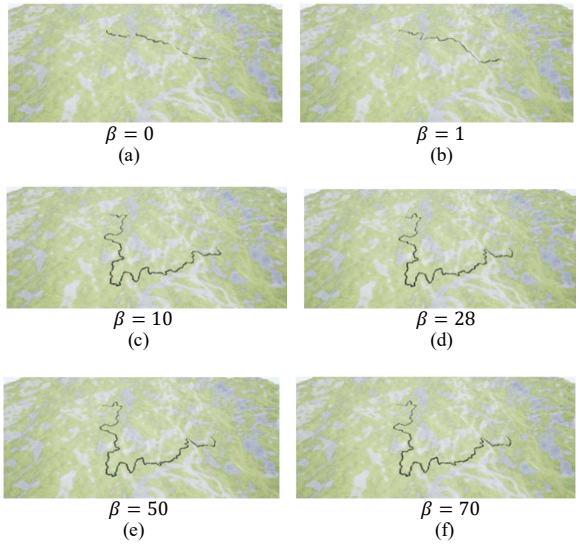


Figure 8. Roads created with differents value of β. All roads have the same starting and ending points.

VI. CREATING THE VILLAGES

In order to create a suitable village, the buildings must be place in areas where most of the building's base is in contact with the terrain. Every 3D model has a mask of its base, which is defined by the user, hence, we must find a place that can fit the model's mask. An example of a mask can be seen in Fig. 9.

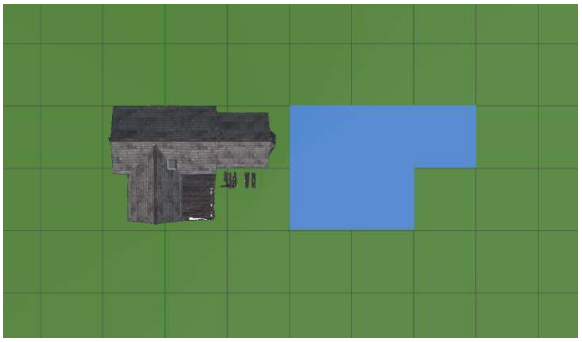


Figure 9. A 3D model of a house and its mask on the grid.

In section II we defined a seed node and the nodes that belong to that seed. Given a village, defined by a seed S and a set of 3D models which are the buildings. Each village has a determined number of building. Each building also has a determined radius, where no other building must be placed.

To place each building, we randomize a node N contained by S. Then we verify if the area around N can fit the 3D model's mask, also checking if it doesn't overlap a road or if it isn't too close to other buildings. If every requirement is met, we place the building and rotate it in a way that it will face the nearest road. Then we create a road from the front of building using the algorithm 3. We can adjust the density of the village by changing the permitted distance between each building.

VII. RESULTS

We implemented this method in C++, using the Unreal Engine 4.19. The tests were executed on a Ryzen 5 1600 at 3,2GHz. We created a terrain of 700x700 vertices, utilizing Perlin Noise [3] and fractional Brownian motion (fBm) with eight octaves. We randomized the Perlin Noise vectors utilizing an integer seed. Thus, giving a different seed we can generate a different terrain.

Our solution to define the area for the villages and the placement of the buildings works on flat terrains (Fig. 10), and on uneven terrains (Fig. 11).

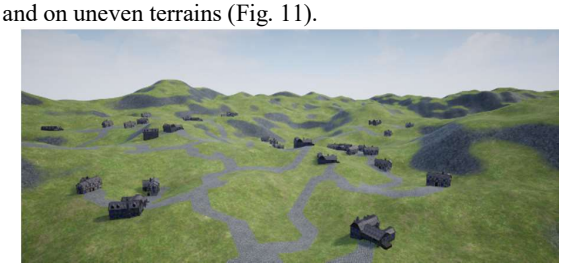


Figure 10. A small village created on smooth terrain.

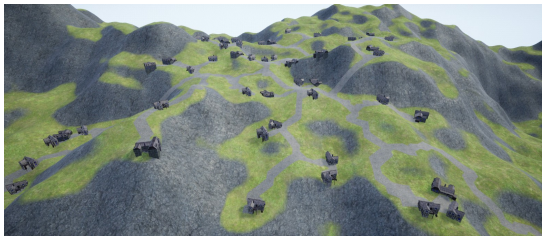
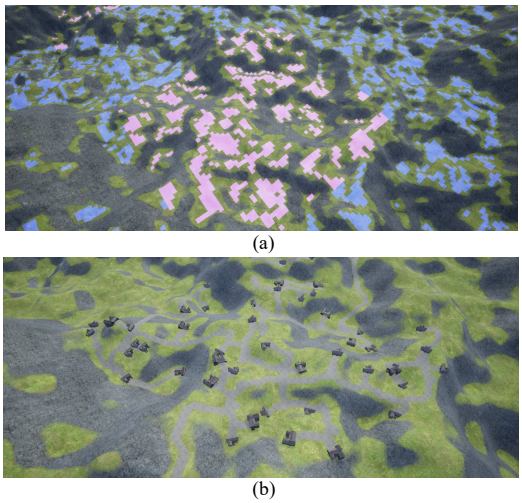


Figure 11. A village generated on an uneven terrain.

For irregular terrains, where we cannot place more than a couple building per subarea, our method still performs well. The villages generated have all the building connected and they are as close as possible to each other, Fig. 12 shows an area delimited and the resulting village on a rough terrain.



(a)

(b)

Figure 12. In (a) the blue areas are the acceptable nodes and the pink areas represent the nodes that belong to the village seed. (b) shows the resulting village.

The generation of roads also works on uneven terrains. When the only possible path is by going on a very steep terrain, the path generated takes the flatter possible track. An example of this can be seen in Fig. 13.

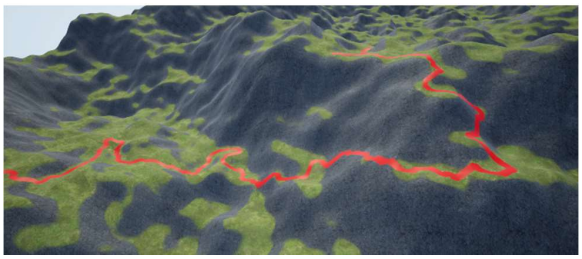


Figure 13. A path climbing a steep terrain.

One of the most impactful variables on our road algorithm is the constant β, both on the execution time and on the path created. The road generation was tested on several different terrains, from the point (100,100) to the point (600,600), with β ranging from 0 to 70. Values larger than 70 showed no difference on the path created, only increasing the execution time. Table I shows the average time in seconds of this test with different values of β. The average time to create eight villages and the road system on the terrain is about 30 seconds.

TABLE I. EXECUTION TIMES IN SECONDS FOR DIFFERENT β VALUES

β	Time
0	0.011326
5	6.355239
10	15.17072
15	21.73047
20	24.77973
29	34.52233
30	37.55081
31	37.49178
40	47.00348
50	56.50435
70	59.15096

The data resulted from the tests can be seen in Fig. 14. The data of all the tests were averaged and normalized so they are shown between 0 and 1. It shows that values of β closer to 30 produces paths with the lowest standard deviation with an acceptable execution time. The difference in cost and the number of nodes tends to remain constant with β above 25.

VIII. CONCLUSION AND FUTURE WORKS

This paper presented a method to generate villages and roads that works on arbitrary terrains. The road generation produces realistic paths utilizing a simple cost function, and is able to create T-junctions when connecting an initial point to any other point belonging to an existing road. The solution proposed to delimit the village area works well and can be easily modified to suit different kinds of application.

The road generation only takes into account the slope of the terrain and only creates paths that follow the terrain geometry. A new research can be made to add more cost functions that would approach other obstacles, like water, vegetation or different types of terrain and also capable of deciding for the generation of bridges and tunnels. Modifying the terrain to adapt to the road is another research problem.

When defining the area for the village, algorithm 1 does not take into consideration the height difference between the nodes. This can be a problem near large cliffs with a small area on the XY plane, since the algorithm may still recognize that area as acceptable, despite the nodes being far away from each other on the Z axis.

Another way to improve the villages is by utilizing a more complex approach to place the buildings on the terrain. This can be done by making use of a method for generating land usage [12][13], which separates the village land into small portions that belongs to a certain building.

ACKNOWLEDGMENT

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. We also thank the Brazilian army for the support through the SIS-ASTROS project.

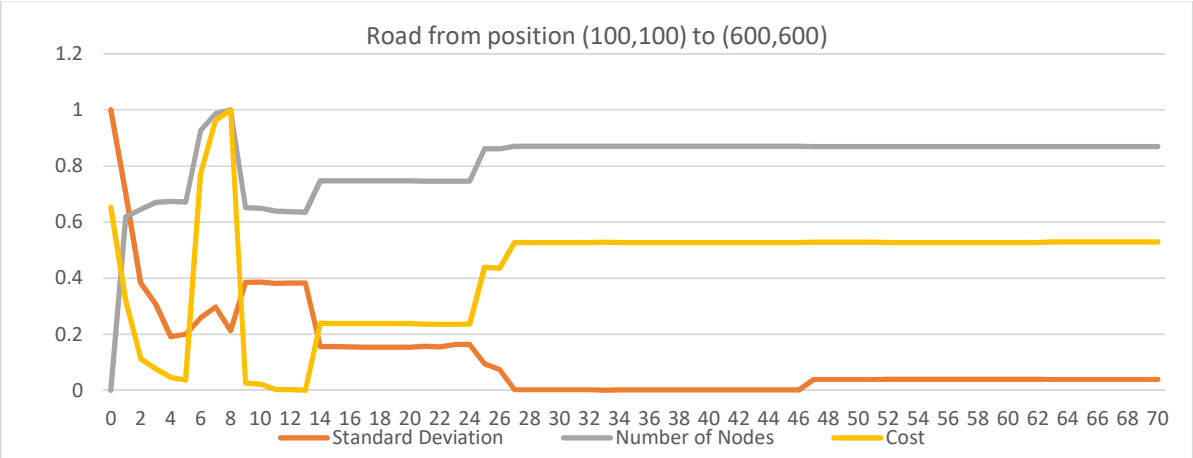


Figure 14. Graph of the average value of standard deviation after several costs. All values were normalizes between 0 and 1.

References

[1] E. Galin, A. Peytavie, N. Maréchal, and E. Guérin, "Procedural generation of roads." Computer Graphics Forum, vol. 29, no. 2, Oxford, UK: Blackwell Publishing Ltd, 2010. pp. 429-438.

[2] A. Emilien, A. Bernhardt, A. Peytavie, M. Cani, and E. Galin. "Procedural generation of villages on arbitrary terrains." The Visual Computer 28, no. 6-8. 2012. pp. 809-818.

[3] K. Perlin. An image synthesizer. ACM SIGGRAPH Computer Graphics, 19(3):287–296, 1985.

[4] E. Galin, A. Peytavie, E. Guérin, and Bedřich Beneš. "Authoring hierarchical road networks." In Computer Graphics Forum, vol. 30, no. 7, pp. 2021-2030. Oxford, UK: Blackwell Publishing Ltd, 2011.

[5] Y. Parish, and P. Müller. "Procedural modeling of cities." In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 301-308. ACM, 2001.

[6] M. De Berg, M. Kreveld, M. Overmars and O. Schwarzkopf. "Computational geometry." In Computational geometry, pp. 1-17. Springer, Berlin, Heidelberg, 1997.

[7] N. Amenta, M. Bern and D. Eppstein. "The crust and the β-skeleton: Combinatorial curve reconstruction." Graphical models and image processing 60, no. 2 (1998): 125-135.

[8] A. Botea, M. Müller and J. Schaeffer. "Near optimal hierarchical path-finding." Journal of game development 1, no. 1 (2004): 7-28.

[9] N. Sturtevant and M. Buro. "Partial pathfinding using map abstraction and refinement." In AAAI, vol. 5, pp. 1392-1397. 2005.

[10] K. Khantanapoka and K. Chinnasarn. "Pathfinding of 2D & 3D game real-time strategy with depth direction A∗ algorithm for multi-layer." In Natural Language Processing, 2009. SNLP'09. Eighth International Symposium on, pp. 184-188. IEEE, 2009.

[11] S. D. Goodwin, S. Menon and R. G. Price. "Pathfinding in open terrain." In Proceedings of International Academic Conference on the Future of Game Design and Technology, p. 8. 2006.

[12] S. Groenewegen, R. M. Smelik, K. Jan de Kraker, and R. Bidarra. "Procedural City Layout Generation Based on Urban Land Use Models." In Eurographics (Short Papers), pp. 45-48. 2009.

[13] T. Lechner, P. Ren, B. Watson, C. Brozefski, and U. Wilenski. "Procedural modeling of urban land use." In ACM SIGGRAPH 2006 Research posters, p. 135. A