Procedural road network generation
Project DGI14

Martin Engilberge
engi@kth.se
930118-T070

Florian Depraz
depraz@kth.se
931108-T039

June 20, 2014

Background:

With the increase of computing power, it is possible to render large virtual worlds. But creating these worlds from scratch might be time consuming. For example, the generation of mountains, water (lakes, rivers), roads and cities that make the environment realistic and non-repetitive is not an easy task.

Problem:

Description For this project, the focus will be on the road network generation of a city. Using some basic data, like height-map and population's density, it will generate a plausible street network, connecting the neighbourhoods. The size of the road will vary (street, major road, highway, avenue) depending on the traffic density and the places connected by the road.

Implementation Method:

Approach At first, we met in order to specify how to make the project, write the proposal. We discussed about the structures in order to make the project easy to manage. We decided to use C++ in order to use the SDL library.
Then we created a prototype of an L-system in order to create a simple road network without taking in account the population nor the altitude of the map. We used GIT in order to manage the project and to keep track of older versions of the project.

Main data structure An array of "Tile" (called *map*) is used to represent the whole city. *map* is an array of fixed size (MAP_SIZE=700). This is the principal structure of the project. It stores all the information about a zone of the map. Each cell is an instance of the class "Tile" which can store the density, the height, if this zone is accessible and if there is a road.

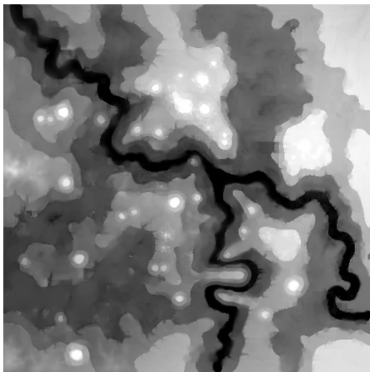
Initialization of the map In the beginning, this array is initialized. Two pictures need to be specified to the program:

- The first one is "dmap.png" representing the density of the population
- The second one is "hmap.png", the height

The function readPng uses a small library (LodePNG [1]) to read each pixel of each image and fill the attributes of *map* (*density* and *height*). When the image is decoded by the library, the algorithm only needs to extract the value of the pixel red. The pictures are in levels of gray therefore the value of red, green, blue are equal. The values are normalized in order to get a value between 0 (low) and 1 (high). To normalize the values the algorithm computes $\frac{currentValue - minValue}{maxValue - minValue}$.

Depending of the picture: A black pixel represents a high density. A white pixel represents low density. A black pixel represents a low altitude (river). A white pixel represents high altitude.

(a) Density map



(b) Altitude map

Figure 1: Example of PNG inputs.

Display The *map* is displayed on the screen using SDL. Reuse of some codes of the labs. The function displayMap loops through all the pixel of the map.

- Black pixel : Road
- Non accessible zone: Gray
- Height: level of red
- Density: Level of green

Finding highest density zones How to deal with the population density? The population density is used to guide the road network. To do so, the map is subdivided in little squares (the size of the squares is parametrizable using the variable sub_square_size) for each square the average density is computed. Then, an ordered list containing the center of each square associated with the average density for these squares is returned. From this list, the algorithm takes the first 40% of them (the most dense since the list is ordered) and the algorithm use these points to create the main road network (Road size: 3), then the next 20% of the list are used to create a secondary road network (Road size: 2) and again the next 20% to do the tertiary network (Road size: 1).

Integration of altitude During the roads generation, the algorithm looks for the closest point to link an existing node to a new one. To find the best point the algorithm finds the minimal value (combining flying distance and height variation) between all the reachable points. The values is computed by summing the flying distance between the two points and the sum of the absolute value of the height variation of all the intermediate points the road is planning to use.

L-system To build the network of roads, the algorithm uses a L-system. A L-system is a collection of propositions used to expand a string. A proposition is composed of a left side symbol and one or more right symbol. To each proposition is associated a probability p to occurs.

The class L-system is initialized with different propositions (which is represented as a structure in the code). Which are:

S $\rightarrow$ SS	p=0.25
S $\rightarrow$ S[Ps]S	p=0.25
S $\rightarrow$ S[Ms]S	p=0.25
S $\rightarrow$ S[Ps][Ms]S	p=0.25
[$\rightarrow$ [	p=1
] $\rightarrow$]	p=1
P $\rightarrow$ P	p=1
M$\rightarrow$ M	p=1
s $\rightarrow$ sS	p=1

S represents straight: the road will continue one step forward.

[(In the program, O is used to represent a right bracket). It is used to create a sub-roads.

] (In the program, C is used to represent a right bracket). It is used to close the inner sub-road and to restart the parent road.

P represents a deviation of the road of the angle specified. This rotation is counter-clockwise.

M represents a deviation of the road of the angle specified. This rotation is clockwise.

s represents a space without road.

The algorithm reads a string and expends it until the number of iteration is equal to 0.
For example. If the initial string is "S". It will search all proposition with a "S" on the left side. According to the probability, it will choose one of them and replace "S" by the right side of the chosen proposition. In this example, if the proposition chosen was "S→S[Ps]S". The string will become "S[Ps]S". And a new iteration will begin.

The right side of the proposition : S → S[Ps][Ms]S means S[Ps][Ms]S. It creates 2 sub-roads which will expend during the next iteration.

The propositions 4 first propositions all have the same probability. The function get_props_from_rhs find all proposition with the good left side and randomly return a proposition according to the associated probability.

Road_Network class The class Road_Network is used to store all the information about the roads. It contains a set of edges and nodes.

- Edges represents the roads. These objects have a starting point (Node start), an ending point (Node end) and the size of the road.
- Nodes contains the coordinates of the node and the size of the largest road using this node.

The constructor takes in parameter the starting point of the network which can be the access point to the city for example.

To create an edge, the algorithm needs a starting and an ending point. The ending point will be changed in case there is an other node close by. In this case, some parameters (the angle, ...) are corrected to keep the road in the right direction. The maximal distance when two close points are merged is defined by a parameter in this class. If there are multiple points in this zone, the closest one is chosen.

When the edge is created, the map containing all the Tile needs to be actualized. The algorithm interpolate the values between the starting and ending point of the edge. For all points in this line, if the coordinates are valid (not out of bound) then the corresponding Tile in the map is actualized.

Global Algorithm The program loads the data from the files and stores them into *map* Then an instance *network* of Road_Network is created with a defined starting point. The next step is to find all the population areas sorted by their average density. Using this list, the following algorithm is used to create the roads.

```
forall (p : listPop){
        Node start = p;
        Node end = network.closestNode(p);

        While(distance(start, end) > MIN){
                Find parameters For L-system
                Create L-system
                Add to the network
                end = network.closestNode(start);
        }

}
```

Then, the *network* is used to modify *map* and sets all the parameters of the tiles. At the end of the execution, the map is displayed and the result is stored as a PNG picture.

Results:

Here are some results from the program.

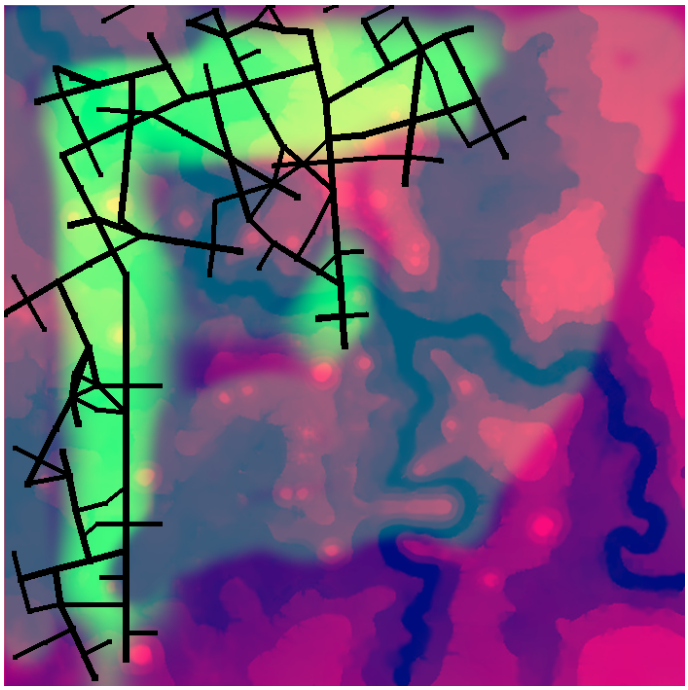


Figure 2: Resulting map- High density zones are covered by roads. No density zone are not covered. But low density does not have roads.

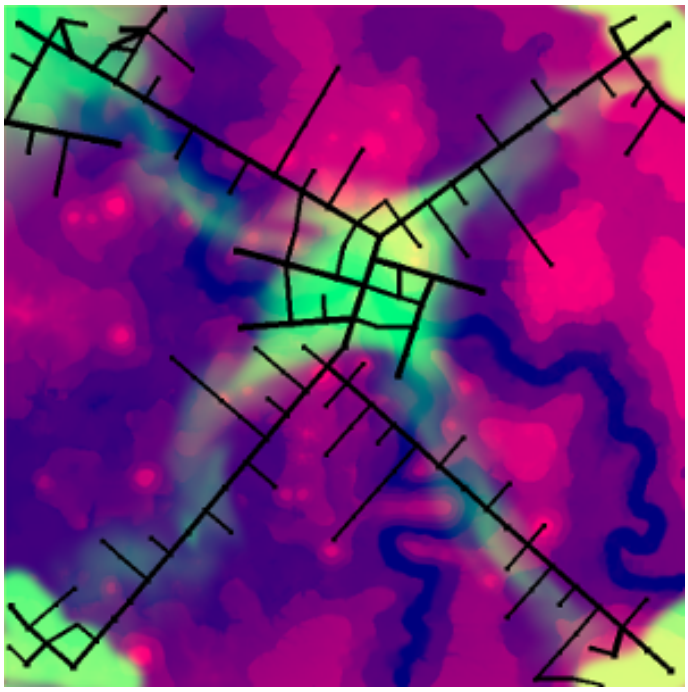


Figure 3: Resulting map- All the different zones are linked by roads.

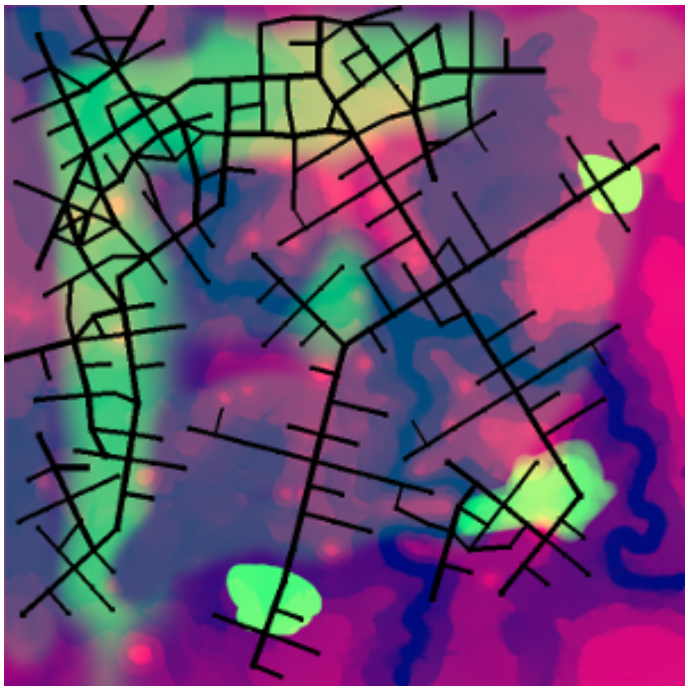


Figure 4: Resulting map- All the zones are linked, but some roads are not usefull. Which is also true in real life: roads going to a forest...

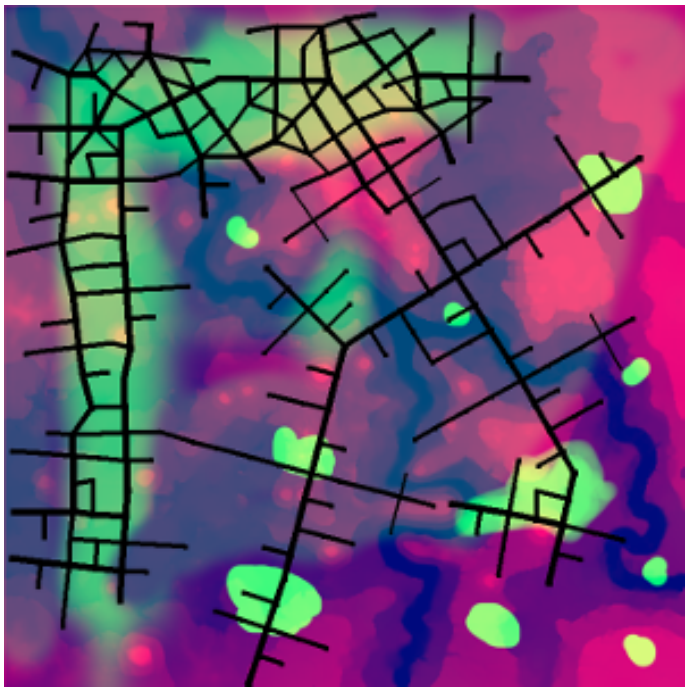


Figure 5: Resulting map - Some zones are not linked. This can be fixed by changing some parameters like diminushing sub_square_size which will find smaller zone of population in the map. The algorithm will then take in count these small zone to build the network

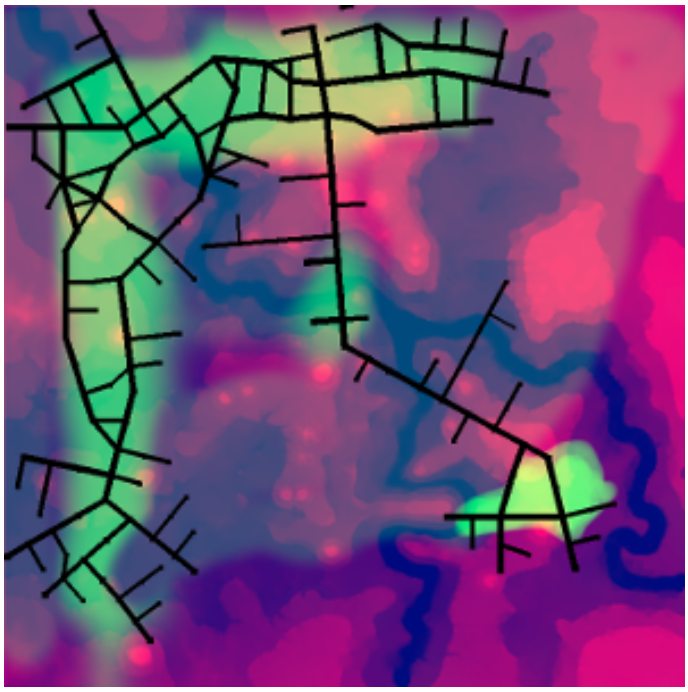


Figure 6: Resulting map

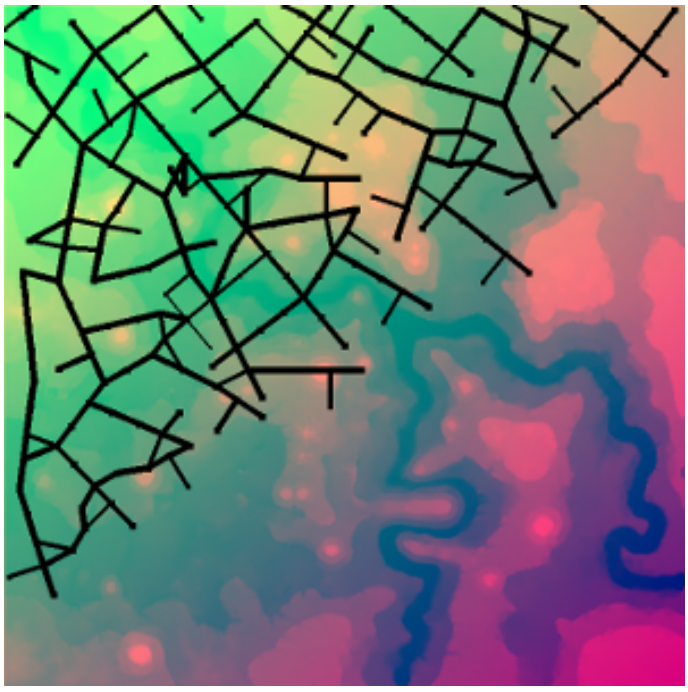
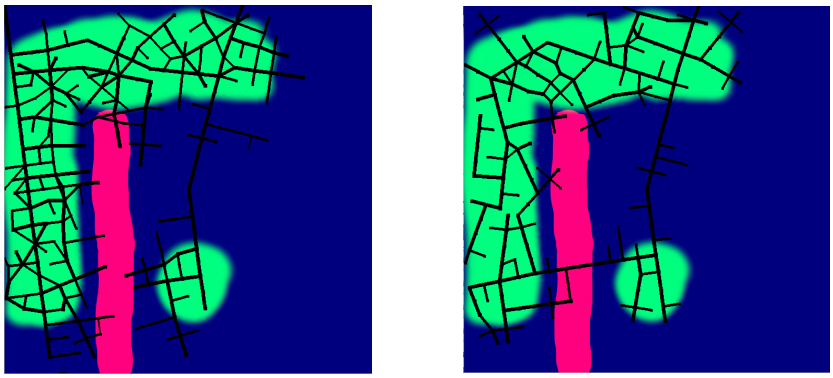


Figure 7: Resulting map showing the whole covering of the zone where the population is situated.

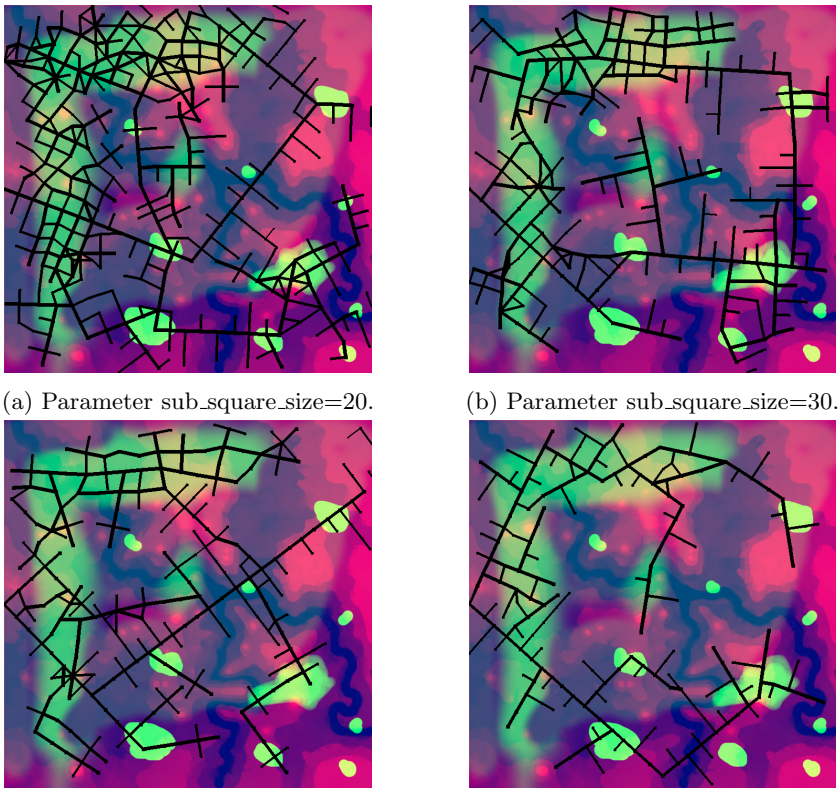


(a) Density map (b) Altitude map

Figure 8: Illustration of the network depending or not or the height. In the second picture, roads between the hill (represented in pink) are not linked because the altitude is too high.

Discussion of those results:

The program is quite flexible. Lots of parameters can be changed in order to find the best network. Figure 5 showed that some zones where not covered. The parameter sub_square_size has been change in the program in order to find smaller population zones. Figure 9 shows the results.



(a) Parameter sub_square_size=20.

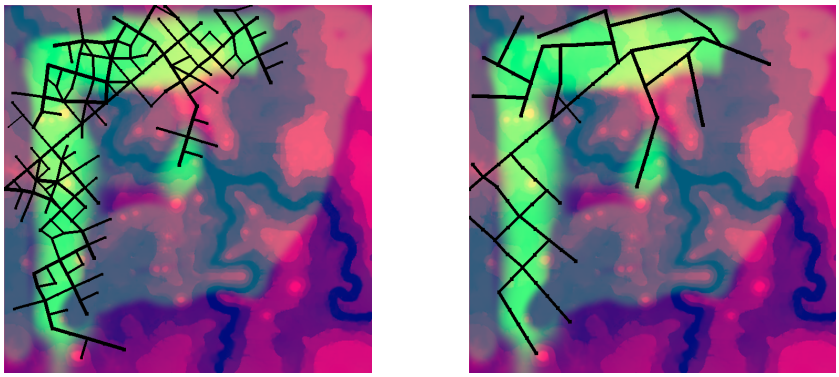
(b) Parameter sub_square_size=30.

(c) Parameter sub_square_size=40.

(d) Parameter sub_square_size=45.

Figure 9: Evolution of the road network depending of the parameter sub_square_size.

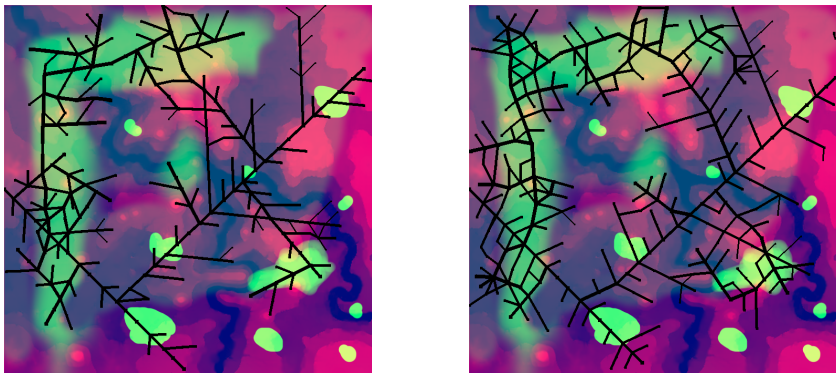
The other parameter that can be modified is the probability of the propositions. Figure 10 illustrates the results. If the first probability is high, the network has more roads straight and less sub-roads.



(a) p1 = 40 - p2 = 20 - p3 = 20 - p4 = 20 (b) p1 = 70 - p2 = 10 - p3 = 10 - p4 = 10

Figure 10: Evolution of the road network depending of the probability of each proposition

The angles between roads can also be adjusted as shown in Figure 11:



(a) Angle: 50 degrees (b) Angle: 70 degrees

Figure 11: Evolution of the road network depending of the angle between the roads

The parameter find_close_node can be changed. If this number is increased, roads can be linked to a node far away. If this parameter is set to zero, roads will not be linked even if they are close. It will create more dead-ends.

References:

[1] LodePNG. Lightweight library for manipulation of PNG. `http://lodev.org/lodepng/`