Procedural road network generation
Project DGI14

Florian Depraz
931108-T039
depraz@kth.se

June 20, 2014

Work Flow:

When we chose this project, we did not know how a L-system could be used to generate roads or paths. At first we did some researches in order to have a precise idea on how to use it. We saw different implementations of L-system on Wikipedia [1] and we decided to implement a prototype (See code at the end of the document). We also used the paper "Procedural modeling of cities" [5] to have a guideline. We used those propositions with the probability associated to it (inspired from [5] and slightly modified):

$$
\begin{array}{ll}
S \to SR & p=0.20 \\
S \to SL & p=0.20 \\
S \to SS & p=0.50 \\
S \to Ss & p=0.10 \\
\\
R \to RS & p=0.70 \\
R \to Rs & p=0.30 \\
\\
L \to LS & p=0.70 \\
L \to Ls & p=0.30 \\
\\
s \to sS & p=0.40 \\
s \to sR & p=0.10 \\
s \to sL & p=0.10 \\
s \to ss & p=0.30
\end{array}
$$

It is a basic prototype without the notion of sub-roads. The path could create roads that are not linked to another. The basic idea is to create the string without any constraint of population or altitude and to follow this string to generate the network. To read this string we implemented a "turtle" which reads character by character and advances according to the character read: S the turtle goes staight in its direction. R it goes right. L it goes left. s it goes straight but without a road. The turle and has a position (x, y) and the program actualizes it.

This implementation gave this kind of results (where 1 is a road and 0 is not):

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 1 1 1 1 0 0 0 1 0 0 0
0 1 0 0 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
1 1 1 1 0 0 0 1 0 1 1 1 1 1 0
0 0 0 1 0 1 0 1 1 1 0 1 0 1 1
0 0 0 1 0 1 0 0 1 1 0 1 0 0 0
0 0 0 1 0 1 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

We saw that the propositions were not enough complex and that we would need a better L-system with more constraints to get a good network. We used some part of the previous labs (like the installation of GML) for this project to save time.

Group work:

As specified in the previous report we used a GIT repository with a "wiki" where we wrote the different steps needed to be implemented. We discussed on how to implements some functions to find the best way. Martin worked on the main algorithm and the L-system. I made some functions in the rng_utility.cpp. I also implemented the functions to read from and write to a PNG image and to use the data read. I tried to find the best combinaison of parameters but since it is subjective, it could have been a good idea to ask to a banch of people which configuration is the best one and to make some statistics.

Experiences:

Finding a good library. In the other report we explained why we chose the PNG format for the inputs and output of the program. One difficulty was to find a good library that fitted our needs. There exist lots of different ones: LodePNG [2], FreeImage[3], PNGwriter[4]... I had to look at all of these to find the best one well documented, easy to instal and to integrate to the project.

Writing a program on a new language, making some search on how the algorithm should work and also working with a partner was also a good experience.

Problems encountered It is always hard to instal and configure the project in the beginning. We decided to use GML and GIT. It required some times to get it functional.

Finding a good function to pick a proposition according to the probability. I took time to figure out how to write the function chosing the next proposition.

Some problems led to infinite loops. Those problems were hard to find since they did not occured at each execution of the program.

Improvements:

All the information about the map are stored in an array (streets, altitude, population density). It could be a good idea to represent the map in 3D using these data. I tried to implement this feature but the time was missing

The L system could be also improved by specifying what kind of network the user wants: we can have different patterns for it: Basic (no specification), New York(Rectangular pattern), Paris (Centralised pattern) [5] Design a more complex L-system (parametric and constraint).

The object *Tile* has a variable "is_accessible" which could be used if there is some zones where no road is allowed. For example if there is a building, or a forest... We could add a new variable to specify if the road is a

The roads are represented using edges: one road is a line. It makes it difficult to have curved roads. The representation units per units (here pixel per pixel) might not be the best choose for this kind of problem. A higher level algorithm may give smoother results. Here, the result might differ depending of the screen size.

Lots of variables (probability of the propositions in the L system, subdivision size to find the population's blocks, road's rotation angle (90 degrees in the program), maximal distance to find a close node, ...) could be adjusted during the execution of the program in order to get a more realistic network. Here those variables are only constants.

Conclusion:

This project was a good and interesting introduction to procedural implementation. Even if the result could be improved with more time the program gives a good network. I learnt how to generate roads by manipulating a L-system and adding constraints to it. Also, I saw how to use the PNG format, like how it is encoded and how to decode it in order to store information. I was not at ease with C++/GIT, it gave me an opportunity to code with this language and to see how to manage a project using the GIT commands.

References:

[1] L-system http://en.wikipedia.org/wiki/L-system
[2] LodePNG. Lightweight library for manipulation of PNG. http://lodev.org/lodepng/
[3] FreeImage http://freeimage.sourceforge.net/
[4] PNGWriter http://pngwriter.sourceforge.net/
[5] Yoav I. H. Parish and Pascal Müller. 2001. Procedural modeling of cities. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01) http://graphics.ethz.ch/Downloads/Publications/Papers/2001/p_Par01.pdf

Prototype

```
#include <iostream>
#include <vector>
#include <string>
#include <cstdlib>


#define MAP_X 30
#define MAP_Y 30

using namespace std;
typedef struct Proposition{

    string leftSide;
    string rightSide;
    double probability;
    double random;

} Proposition;

enum direction {U=0, L=1, D=2, R=3};

typedef struct Turtle{

    int x;
    int y;
    int dir;

} Turtle;


string getPattern(string init);
Proposition getNextProposition(string leftSide);
void moveTurtle(string moves);
void moveStraight(bool draw);
void drawMap();
int mod(int x, int m);

int map[MAP_Y][MAP_X];
Turtle t = {MAP_Y/2,MAP_X/2, U};

const int iterations = 10;
string init = "S";
vector<Proposition> propositions;

int main(int argc, char **argv){

    srand(time(NULL));

    Proposition S1 = {"S", "SR", 0.20, 0.0};
    Proposition S2 = {"S", "SL", 0.20, 0.0};
    Proposition S3 = {"S", "SS", 0.50, 0.0};
```

```
    Proposition S4 = {"S", "Ss", 0.10, 0.0};

    Proposition R1 = {"R", "RS", 0.70, 0.0};
    Proposition R2 = {"R", "Rs", 0.30, 0.0};

    Proposition L1 = {"L", "LS", 0.70, 0.0};
    Proposition L2 = {"L", "Ls", 0.30, 0.0};

    Proposition s1 = {"s", "sR", 0.40, 0.0};
    Proposition s2 = {"s", "sL", 0.10, 0.0};
    Proposition s3 = {"s", "sS", 0.10, 0.0};
    Proposition s4 = {"s", "ss", 0.30, 0.0};

    propositions.push_back(S1);
    propositions.push_back(S2);
    propositions.push_back(S3);
    propositions.push_back(S4);
    propositions.push_back(R1);
    propositions.push_back(R2);
    propositions.push_back(L1);
    propositions.push_back(L2);
    propositions.push_back(s1);
    propositions.push_back(s2);
    propositions.push_back(s3);
    propositions.push_back(s4);

    string result = init;
    int i;
    for (i = 0; i < iterations; ++i) {
        result = result + getPattern(result);
    }

    moveTurtle(result);
    drawMap();
    return 0;
}

int mod(int x, int m) {
    return (x%m + m)%m;
}


void moveTurtle(string moves){


    for (int i = 0; i < moves.size(); ++i) {

        switch (moves.at(i)) {
        case 'S':
            moveStraight(true);
            break;

        case 'L':
            t.dir = mod((t.dir+1), 4);
```

```
            break;

        case 'R':
            t.dir = mod((t.dir-1), 4);
            break;

        default:
            moveStraight(false);
            break;
        }


    }

}


void drawMap(){
      for (int i = 0; i < MAP_Y; ++i) {
         for (int j = 0; j < MAP_X; ++j) {
          cout << map[j][i] << " ";
        }
        cout << endl;
    }
}

void moveStraight(bool draw){

    switch (t.dir) {
    case U:
        t.y = t.y -1;
        break;
    case D:
        t.y = t.y + 1;
        break;
    case L:
        t.x = t.x - 1;
        break;
    case R:
        t.x = t.x + 1;
        break;
    default:

        cout << "Problem with direction " << t.dir << endl;
        exit(0);
        break;
    }


    if(t.x == MAP_Y || t.x < 0){
        return;
        t.x = MAP_Y/2;
    }
```

```
    if(t.y == MAP_X || t.y < 0){
        return;
        t.y = MAP_X/2;
    }

    if(draw){
        map[t.x][t.y] = 1;
    }

}

string getPattern(string init){

    string result = "";

    for (int j = 0; j < init.size(); ++j) {
        result = result + getNextProposition(init.substr(j,1)).rightSide;
    }

    return result;
}


Proposition getNextProposition(string leftSide){

    vector<Proposition> p;
    for (int var = 0; var < propositions.size(); ++var) {
        if(propositions[var].leftSide == leftSide){

            propositions[var].random = (rand() % 1000) / 1000.0 -
                (1-propositions[var].probability);
            p.push_back(propositions[var]);

        }
    }

    double max_p = -10;
    int index = 0;
    for (int var = 0; var < p.size(); ++var) {

        if(p[var].random > max_p){
            max_p = p[var].random;
            index = var;
        }

    }

    return p[index];

}
```