Description of the Stone Generation Pattern Code

# Description of the folders

* 1. Cropping

This folder one function only crop\_picture.m. As its name suggests, it crops the list of polygons once the stones have been done. This is done in order not to have a layer of mortar on the outer frame of the picture.

* 1. Drawing

This folder contains a variety of functions used to represent graphically the stones. The most used one, draw\_stones, draws all the stones out of a cell of polygons. Some others draw the number of the nodes, the contact points, create the color or draw only one stone at a time.

* 1. Fractalization

This folder realises the fractalization, as described in [5]. The fractalization consists in subdividing the edges of a polygon and putting a random normal coordinate to the edges in order to make them look more realistic/noisy.

* 1. Parameters

This folder contains all the functions used to set the stone generation parameters. The general parameters\_stone\_masonry.m script gives all the general parameters. The other 3 files allow the user to make the parameters space-dependent for the various step they describe.

* 1. Random\_field\_generation

This folder allows the creation of a random field that will be used to generate the random variability of the stones. Correlation\_fun.m and randomfield.m come from [7] and create\_random\_field.m is just a utility that I wrote to make the use of this function easier inside my code.

* 1. Resampling

The resampling folder contains everything that has to do with resampling the polygons once the erosion has been done. The erosion process leaves the polygons with too many vertices, as the polygons are spatially discretized and each element on the outer frame becomes a vertex. The resampling is done with a target edge length as and input. There is also the put\_corners.m function that does a special resampling treatment in the case regular stone masonry is generated.

* 1. Scripts

This folder contains all the scripts that have been used. The main script, main\_script.m does all the treatment from generating the straight pattern to computing the shortest path along the stones. There is also a few tests scripts that I left as they can be useful to test some functions.

* 1. Shaking

The shaking operation takes place right after the straight\_pattern generation. It consists in moving each node by a random distance along x and y axis. The operation is accepted if and only if all the polygons in the picture remain convex.

* 1. Shortest\_path

The shortest path folder contains every file related to the shortest path computation once the visibility graph has been done (so it does not contain the files from the VisiLibity C++ library that are in the visilibity folder). It contains though the function compute\_visibility\_graph.m that uses the functions in the VisiLibity library.

To compute the shortest path, the function dijkstra\_modified.m which is a light modification of dijkstra.m (found in [4]). There is also a function that allows the user to pick the starting and ending points (get\_points.m) and a function to compute the interlocking given an optimal path.

* 1. Sieving

This folder contains only the sieving function that “sieves” the stones, i.e. removes the smallest stones.

* 1. Stone\_erosion

This folder contains everything that has to do with the erosion of the polygons. The erosion process is done according to the method described in [2],[3],[4]. The functions in this folder also allows to put contact points, which will remain uneroded throughout the process.

* 1. Straight\_pattern

The straight pattern generation is the first step of the stone generation algorithm. It consists into filling a rectangular area with rectangles of various (or not) dimensions. This is done row by row until the area is filled.

* 1. Utilities

The utilities folder contains little functions that help in many cases like a function to know if a point is on a stone, a function that gives the distance between two points, a function that saves all the parameters in a .txt file, and two functions that order the polygons nodes (the way the input is given vary).

* 1. Visibility

This folder contains the visilibity C++ library files that are used by my algorithm. I deleted the files not directly useful for the algorithm but they can be downloaded at [6]

# Data Structures

Throughout the process, many different data structures are used. But I tried as much as I could to make them consistent. In the first two steps (Straight pattern and Shaking), the structure is different as each node can belong to many stones. In those two steps the information about the stones is split in two tables. One table, pos\_tab, contains the XY coordinates of the nodes and one other table, stone\_nodes contains the index of the vertices of the stones in the pos\_tab matrix.

The shaking process takes these two tables and return a cell array. After this step, and it will remain like that until the end of the process, the stones are stored in a cell array. Each cell contains a Mx2 matrix with the x-y coordinates of the M vertices of the stone.

# References

[1] Miyata, Kazunori, «A Method of Generating Stone Wall Patterns», in *Computer Graphics,* volume 24, number 4, August 1990, pp. 387-394

[2] Beardall, M. et. Al., “Goblins by Spheroidal Weathering”, in *Proceedings of the Third Eurographics Conference on Natural Phenomena,* 2007, pp. 7-14

[3] Peytavie, A. et. Al., “Arches: a Framework for Modeling Complex Terrains”, in *Eurographics,* volume 28, number 2, 2009, pp. 457-467

[4] Peytavie, A. et. Al., “Procedural Generation of Rock Piles using Aperiodic Tiling”, in *Eurographics,* volume 28, number 7, 2009, pp. 1801-1809

[5] Fournier, A. et. Al., “Computer Rendering of Stochastic Models”, in *Communications of the ACM,* volume 25, number 6, June 1982, pp. 371-384

[6] Kirk J., “*Dijkstra’s Shortest Path Algorithm”* Mathworks File exchange, http://www.mathworks.com/matlabcentral/fileexchange/12850-dijkstra-s-shortest-path-algorithm

[7] Constantine P., “*Random Field Simulation”* Mathworks File Exchange, http://www.mathworks.com/matlabcentral/fileexchange/27613-random-field-simulation

[8] K. J. Obermeyer and Contributors, “*The VisiLibity Library”*, 2008, A C++ Library for floating point visibility computations, http://www.visilibity.org/

**Shortest Path**

**Straight Pattern Generation**

**Shaking Algorithm**

**Erosion Algorithm**

**Resampling, Cropping and Removal of small stones**

**Shortest Path**

Straight pattern

Convex Polygons

Eroded Polygons

Resampled, Cropped and sieved stones

Toolboxes needed :

distrib\_computing\_toolbox

map\_toolbox

matlab

statistics\_toolbox