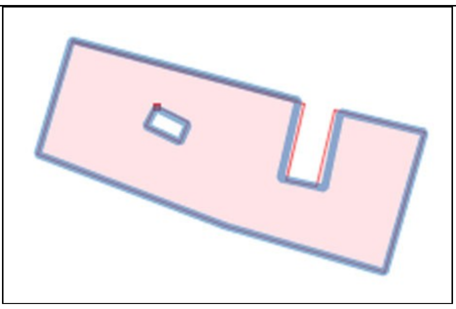
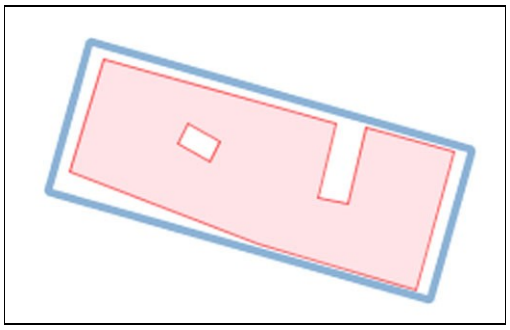
OpenJump Plugin Description Document			
Plugin Name	**Map Generalisation Toolbox**		
Version	1.0	**Release Date**	March 2006
Authors	Stefan Steiniger (mentaer@jpp)	**Author Contact**	perriger@gmx.de
File Name	mapgentools.jar	**License**	MapGen Open Source License
Short Plugin Description What does it?	The toolbox provides several map generalisation algorithms: 1) Building Spread Narrow Parts 2) Enlarge Building to Rectangle 3) Square Building Walls 4) Eliminate Small Buildings 5) Eliminate “Points in Line” of Building 6) Simplify Building 7) Simplify Building to Rectangle 8) Change Elongation of Building 9) Displace Lines 10) Line Smoothing Simple Version 11) Line Simplify JTS 1.5 Algorithm 12) Merge Polygons		

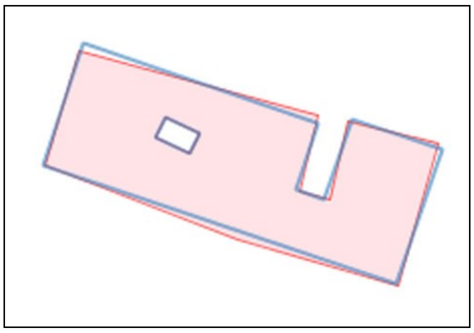
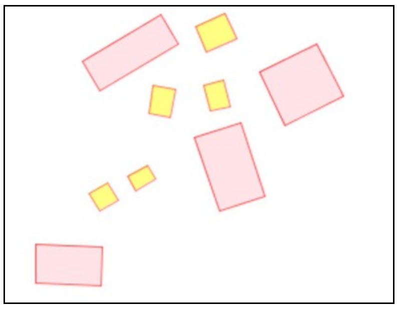
Plugin Dependencies (list the libraries, specify also library version)	Library	Version	Info on library
	. needs JUMP core and - JTS . JMat . snakesalgos.jar . bdgoutlinesimplify.jar	 R 1.6 5.0	 Java GIS Geometry library Provides math function for matrix operations Line generalisation algorithms A building generalisation algorithm

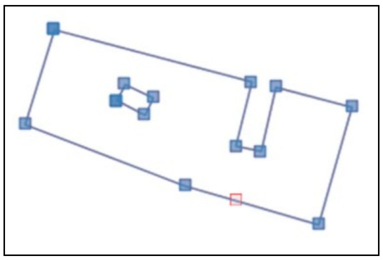
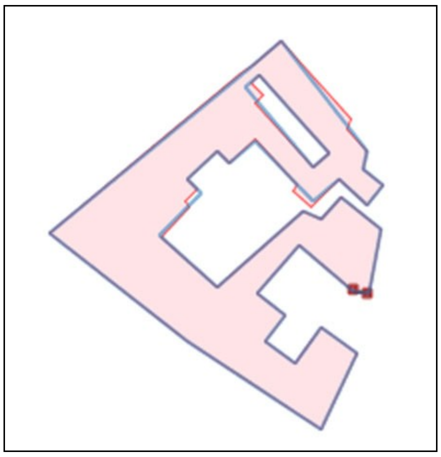
Tested with JUMP versions	JUMP version	Test result (ok or failed)
	. Jump R1.1.2 . Jump i18n R1.2 . OpenJump R1.0	ok ok ok

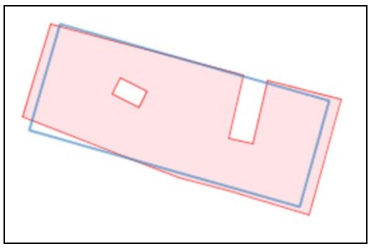
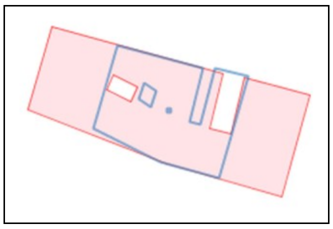
Detailed description of functions
(list every function and their Use, include examples and screen shots if appropriate)

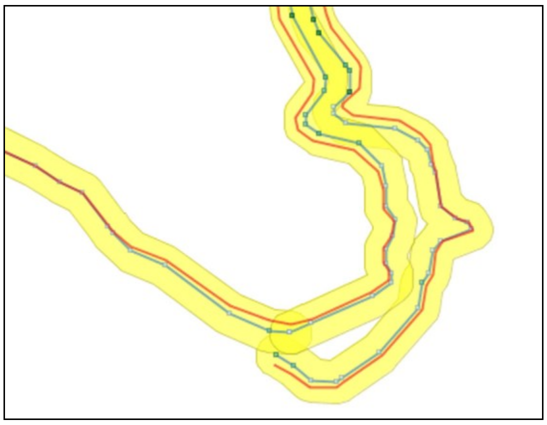
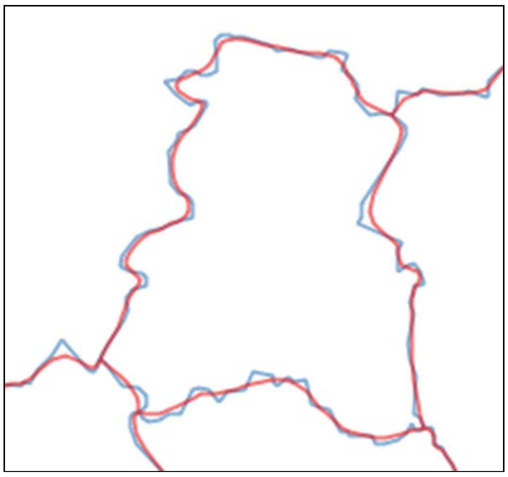
Function: menu item	Detailed description: What does it? and How to use?
\PlugIns\Map Generalisation\ Scale Dependent Algorithms\ Buildings\ Building Spread Narrow Parts	. Enhancing legibility of buildings by spreading parts of a building that are to narrow . The algorithm is described by N. Regnauld, A. Edwardes and M. Barrault (ACI Workshop, 1999) and in Agent Work Package D1. **. input:** 1) selection of buildings 2) target map scale 3) decision if points and edges (walls) or only edges are moved 4) number of iterations **. output:** Modified buildings in new layer. The layer contains a new attribute BuildingSpread reporting the status. A second layer contains edges which still have conflicts. **. image:**

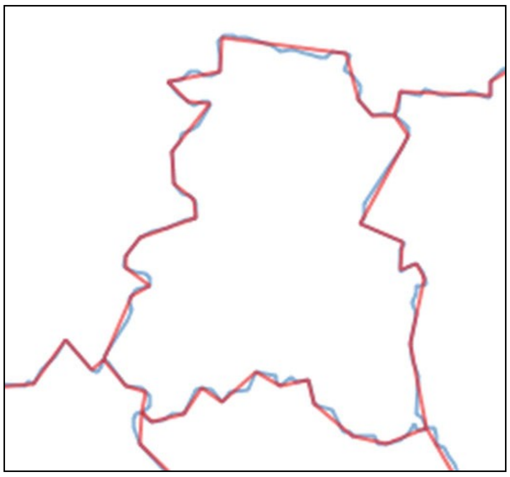
	 Figure 1. Red: original building, brown dot: not solved conflict, blue: spreaded building. **. note:** threshold value for map: 0.25mm taken from Swiss Society of Cartography (issue no. 17, 2005)
\PLUGINS\MAP GENERALISATION\ SCALE DEPENDENT ALGORITHMS\ BUILDINGS\ ENLARGE BUILDING TO RECTANGLE	. Tests if the building area is smaller than a defined minimum area (dependend on map scale) or a minimum width conflict appears. If the size is too small the algorithm calculates the minimum bounding rectangle to replace the original shape and afterwards enlarges it to the minimum size. If the size is ok, the original building will be retained. Holes (courtyards) will be deleted. . The algorithm is described in Agent Work Package D1 by M. Bader. **. input:** 1) selection of buildings 2) target map scale 3) number of iterations (because further enlargement could be necessary) **. output:** Modified buildings (not in a new layer – *therefore do a copy first*). **. image:**  Figure 2. Red: original building, blue: simplified and enlarged building. **. note:** threshold value for map: 0.25mm for width and for area 0.35mm*0.35mm taken from Swiss Society of Cartography (issue no. 17, 2005)
\PLUGINS\MAP GENERALISATION\ SCALE DEPENDENT ALGORITHMS\ BUILDINGS\ SQUARE BUILDING	. Rectifies (squares) the walls of a building. Therefore the building main directions are obtained from the longest building walls. The algorithm works with respect to two thresholds: First it allows a maximum change in the wall angle given by the user; second it respects a maximum point displacement of the corner points calculated from the user given target map scale value. . The algorithm is described by N. Regnauld, A. Edwardes and M. Barrault (ACI Workshop, 1999) and in Agent Work Package D1. **. input:** 1) selection of buildings 2) target map scale to calculate a maximal acceptable point displacement 3) maximum allowed angle of rotation of building wall **. output:** Modified buildings (not in a new layer – *therefore do a copy first*).

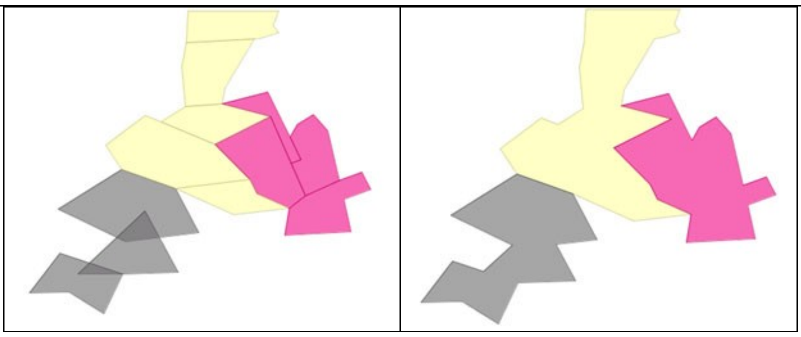
	. image:  Figure 3. Red: original building, blue: squared building. **. note:** threshold value for map: 3*0.2mm for maximum point displacement. The value of 0.2mm is taken from Swiss Society of Cartography (issue no. 17, 2005)
\PLUGINS\MAP GENERALISATION\ SCALE DEPENDENT ALGORITHMS\ BUILDINGS\ ELIMINATE SMALL BUILDINGS	. Eliminates Buildings which are to small, falling below an area threshold calculated from the given target map scale. **. input:** 1) selection of buildings (must be of same layer) 2) target map scale to calculate minimum size threshold **. output:** 1) Building layer without the deleted buildings. 2) Building layer containing only the eliminated buildings. **. image:**  Figure 4. Red outline: original buildings, yellow fill: deleted buildings. **. note:** threshold value for map: 0.35*0.35mm for maximum point displacement. The value of 0.2mm is taken from Swiss Society of Cartography (issue no. 17, 2005)
\PLUGINS\MAP GENERALISATION\ SCALE DEPENDENT ALGORITHMS\ BUILDINGS\ ELIMINATE POINTS IN LINE OF BUILDING	. Eliminates intermediate wall points of a building if the adjacent walls have nearly the same direction. The case of points in a line is usually a result if two adjacent buildings are merged. . The criteria if a point can be delete are: 1. the point must be within a line (no large change of wall direction in the point) and 2. the distance between point and wall without point must be below a threshold, which is an indicator for the wall displacement. The maximum displacement is given by the target map scale. **. input:** 1) selection of buildings 2) target map scale to calculate displacement threshold **. output:** Building layer with the simplified buildings. **. image:**

	 Figure 5. Blue: original building points, red: eliminated point in line. **. note:** threshold value for map: 0.2mm for maximum wall / point displacement taken from Swiss Society of Cartography (issue no. 17, 2005)
\PLUGINS\MAP GENERALISATION\ SCALE DEPENDENT ALGORITHMS\ BUILDINGS\ SIMPLIFY BUILDING	. Simplifies the outline of a building. Small edges/walls, invisible with respect to the map scale, are deleted. . Such an algorithm is proposed and described by N. Regnauld, A. Edwardes and M. Barrault (ACI Workshop, 1999) and in Agent Work Package D1. Other descriptions are given by Sester (IJGIS, 2005). The implementation here is an own implementation slightly different from Regnauld et al. (1999). **. input:** 1) selection of buildings 2) target map scale to calculate displacement threshold 3) option to solve iterative including the number of iterations **. output:** 1) Building layer with the simplified buildings. The output layer contains a new attribute describing the solution status. 2) Layer with problematic edges describing conflicts which could not be solved within the number of iterations or are not solvable for the algorithm **. image:**  Figure 6. Red: original building, blue: simplified outline, brown points and line: not solvable problem. **. note:** threshold value for map: 0.25mm for minimum wall length taken from Swiss Society of Cartography (issue no. 17, 2005)
\PLUGINS\MAP GENERALISATION\ NOT SCALE DEPENDENT ALGORITHMS\ BUILDINGS\ SIMPLIFY BUILDING TO RECTANGLE	. Simplifies the outline of a building to a rectangle. Therefore the Minimum Bounding rectangle is calculated. Holes, e.g courtyards, are deleted but the size of the area should be preserved. The algorithm is similar to ENLARGE BUILDING TO RECTANGLE. . The algorithm is described in Agent Work Package D1 by M. Bader. **. input:** selection of buildings

	. output: Modified buildings (not in a new layer – *therefore do a copy first*). **. image:**  Figure 7. Red: original building, blue: simplified building.
\PLUGINS\MAP GENERALISATION\ NOT SCALE DEPENDENT ALGORITHMS\ BUILDINGS\ CHANGE ELONGATION OF BUILDING	. Changes the elongation of a building / polygon by a given scale factor. The centre point is the centroid of polygon. The edge defining the length of the building is the longest edge of the minimum bounding rectangle. . The algorithm is described in Agent Work Package D1 by M. Bader. **. input:** 1) Selection of buildings (must be from same layer) 2) Scale Value: value to stretch/compress between [0.0 < 1.0 < X]. A value of 1.0 retains the original building shape. **. output:** New layer with modified buildings. **. image:**  Figure 8. Red: original building, blue: stretched building for value 0.5. To see is as well the origin of stretching.
\PLUGINS\MAP GENERALISATION\ SCALE DEPENDENT ALGORITHMS\ LINES\ DISPLACE LINES	. The algorithm displaces lines from lines to ensure cartographic legibility, that means to preserve visual separability of lines. It might be useful for generalization of contour lines or roads in combination with simplification and smoothing algorithms. . The algorithm: ▪ splits long lines with a huge number of vertices to avoid big matrices. ▪ allows only a maximum number of lines to proceed to avoid a to big network matrices ▪ partly also displaces network nodes . The displacement algorithm is based on the "Snakes technique" described in Steiniger and Meier (2004). Other references are Burghardt and Meier (1997), Burghardt (2001) or Bader (2001). **. input:** 1) selection of lines 2) target map scale to calculate displacement values from the minimum distance for separability. 3) The width (diameter) of the future line symbol 4) Decision if the algorithm is processed iterative and how many iterations. (stronger displacements need more iterations) **. output:** 1) Layer with the displaced lines. 2) Layer with the buffers, whereby the buffer radius is calculated from the sum of future symbol width and the minimum separation distance

	. image:  Figure 9. Blue: original lines, red: displaced lines, yellow area: minimum distance buffer, points show the line vertices. **. note:** . Threshold value for map: 0.2mm for maximum point displacement taken from Swiss Society of Cartography (issue no. 17, 2005) . *The algorithm needs appropriate computational power*.
\PlugIns\Map Generalisation\ Not Scale dependent Algorithms\ Lines\ Line Smoothing Simple Version	. The algorithm iteratively smoothes lines with a Spline like approach until a maximum displacement, defined by the user, is reached. Thereby the algorithm is stopped if the first point of all points in a line reaches the displacement limit. The implementation should preserve network connections. . The smoothing is based on the "Snakes technique" described in Steiniger and Meier (2004). Another reference is Burghardt (2005). **. input:** 1) selection of lines 2) The maximum displacement [in m] of a point (accuracy parameter). 3) Decision if the line should be split on point with strong changes of line direction to preserve such salient points. **. output:** Modified lines (not in a new layer – *therefore do a copy first*). **. image:**  Figure 10. Blue: original lines, red: smoothed lines without split on salient points to preserve them. **. note:** *The algorithm needs appropriate computational power*.

\PlugIns\Map Generalisation\ Not Scale dependent Algorithms\ Lines\ Line Simplify JTS 1.5 Algorithm	. The algorithm simplifies lines by reducing the number of points. Thereby a maximum distance of accuracy must be specified by the user. Points falling under the accuracy threshold, accuracy is here the distance of point to a imaginary line defined by line start and end point, are eliminated. . The point reduction is based on an approach by Douglas and Peucker (1976) and implemented by VividSolutions in the JTS geometry library. **. input:** 1) selection of lines 2) The maximum displacement [in m] of a point (accuracy parameter). **. output:** Modified lines (not in a new layer – *therefore do a copy first*). **. image:**  Figure 11. Blue: original lines, red: simplified lines (with same accuracy value like in Figure 10).
\PlugIns\Map Generalisation\ Not Scale dependent Algorithms\ PolyGons\ Merge Polygons	. The algorithm unions polygons if the polygon borders touch or overlap each other. Three different ways of merging are possible: ▪ Merge 2 polygons with attributes: The algorithms merges the geometry of two previously selected polygons. The new layer contains all attributes of both polygons ▪ Merge all touching polygons: Union operation based solely on touching or overlapping geometry. Attributes are not kept. ▪ Merge all polygons if attribute value is similar. Merge touching or overlapping polygons if additionally the attribute value of one selectable attribute is similar (see Figure 12 below). The attribute and the layer will be selected in a second dialog. . The algorithm uses JTS built in union method. **. input:** 1) Selection of Polygons or Selection of Layer with Polygons. Depending on the chosen merge option. 2) If the third option is chosen, in a second dialog the attribute has to be selected, where values must be similar to merge the polygons. **. output:** The merged polygons in a new layer. **. image:**

	 Figure 12. Example for merge of touching polygons with similar attribute values (option 3). Left: original polygons, right: merged polygons.
NOTES	The cited algorithm references might be found using http://scholar.google.com.

document written on: [22/09/2012]
by: [Stefan Steiniger]