

Automated Vectorisation of Raster Geometry

Project Specification

Andrew Davey

Problem Outline

Large amounts of geographical data exist purely in printed form. Those that are electronic are often only stored in raster form. However, people want to query this data in ways far richer than simple pixels can provide. Digitizing the raster data into vectors is a critical step.

Current GIS (Geographical Information Systems) provide only basic, manual, tools to capture vector information. The process relies solely on the user to select points, making it tedious and error prone.

Three forms of vector data are of interest to this project, namely points, poly-lines and polygons. Points tend to be small objects on a map, such as houses. Examples of poly-lines are roads and rivers. Polygons represent areas, like woodland.

This project will develop algorithms capable of transforming aspects of raster imagery into representational vector objects. However, general vectorisation of an entire raster is unrealistic and for most circumstances inappropriate. Raster maps have intersecting, noisy and layered data. Mostly a user requires vectorisation of individual features, for example, a road. The algorithms developed should take a small amount of input, providing information about what features to vectorise.

Technical Objectives

The primary technical objective is creating a library of functions that receive rasters as input and return vector data. Broadly, the algorithms fall to three categories, but all share a common base of pixel selection operations.

Pixel Selection

Raster input can contain many potential vectorisation objects. A selection algorithm determines if a pixel belongs to the target object. The distinguishing attribute of pixels is principally their colour. An assumption is that a single feature uses a small fixed set of colours. Therefore, a naïve algorithm can simply check a pixel's colour is contained in that set. Rasters scanned from printed sources tend to have a range of colours that look similar, but are not. The HSL colour space enables comparison of colour similarity. It is then possible perform fuzzy membership of a feature's colour set.

Vectorisation algorithms may require a Boolean membership value, in which case a similarity threshold determines actual membership. However, more advanced algorithms can possibly use the extra information to better vectorise a raster.

Considering similar colours, only if they are near an exact match pixel is a potential extension to the algorithm. This would lessen the amount of erroneous pixels one often finds scatter on scanned rasters.

Points

Point objects usually consist of a cluster of pixels on a raster. The point vectorising algorithm will use a pixel selection algorithm to get the set of pixels it must vectorise. Clusters of pixels representing a single point feature must be grouped according to a defined radius.

Optional processing may be applied to remove clusters that are too small. This could be due to a noisy input raster.

The final output is a list of Cartesian coordinates representing the centre of each cluster.

Poly-lines

Line objects on maps typically have a width, however this algorithm will return a purely one dimensional centre line. The vector object is a poly-line consisting of straight-line segments. The assumption is that, locally, map lines are straight.

Other map features often obscure line objects. This leaves gaps in the data used to vectorise the lines. The algorithm should try to continue past gaps and rejoin at the next segment. The method chosen to join line segments could be an input from the user. For example, either extending each line and calculating an intersection point or attempting to generate a smooth curve between the end-points.

There will often be noise around the line in the form of same coloured pixels present in separate map features. The algorithm must avoid being misled by such noise.

A number of techniques and implementations for this algorithm are to be considered. Linear regression is a likely candidate for fitting a line to a set of points.

At each segment of the poly-line, a window of pixels must be selected from the source raster. The size of this window should be relative to the width of the line. Therefore, a method to estimate the line width is required.

Polygons

Two dimensional map features should be transformed into their vector polygon approximations. A single pixel on a map has an area. A polygon of a single pixel is therefore its bounding rectangle. A map feature polygon is the union of all the pixel polygons. The polygon data structures must support holes.

The Weiler-Atherton¹ algorithm will be used to perform polygon union.

Polygon area thresholds allow for the removal of small polygons and holes due to noisy data.

Demonstration

Maps are an inherently visual form of data. Therefore, a graphical application must exist to demonstrate the algorithms at work.

Methods

The approach taken to this project will be very much an iterative process. Creating successful algorithms relies very much on empirical analysis of results from many test map images. Naïve algorithms developed for simple map features must be refined to cope with data that are more complex. Documentation of the theories developed will show this evolution.

Individual algorithms that address separate concerns will be designed with composition in mind. The aim being to build a “toolkit” of functions that can be assembled to best vectorise some raster geometry.

The author’s Warwick blog will keep a record of the project’s life cycle. Regular updates will track progress and discussions of current work.

Implementation of the algorithms will take place in the Nemerle² programming language. This is a hybrid functional/object-orientated language that compiles into

¹ <http://www.cs.drexel.edu/~david/Classes/CS430/HWs/p214-weiler.pdf>

² <http://www.nemerle.org/>

MSIL³ byte-code capable of executing on any .NET CLI implementation. This allows access to the .NET framework's rich library, making file I/O, image processing, user interface, etc, trivial.

Timetable

Project work will be undertaken in two-week "phases". Each phase consists of a planning stage, where the goals are set. Performance during previous phases dictates the formation of realistic goals. Implementation is the main stage of a phase. During the implementation stage, research, design, programming, testing, etc, takes place. The goals set during planning determine the precise nature of implementation. Finally, a review stage compares the achieved work to the initial goals. Where possible, recording documentation and example implementations demonstrates a phase's progress.

The overall goal for the first four phases is to design, develop and test the algorithms discussed above. After which, further phases will enhance the algorithms and develop a full graphical demonstration application.

The outline below details the first few phases. Later phases are dependant on performance in previous phases. They are therefore open to change as the project proceeds.

Phase 1

Week 4 – 5

Goals

- Setup and configure development environment and tools such as IDEs, compilers and build systems.
- Collect example raster maps.
- Design and implement a basic algorithm to select pixels from a raster.
- Design and implement a basic algorithm to vectorise a simple line. The input raster should not have large amount of noise or errors.

Phase 2

Week 6 – 7

Goals

- Design and implement an algorithm to vectorise point objects.
- Research and implement the Weiler-Atherton algorithm.
- Design and implement a basic algorithm to vectorise polygons.

Phase 3

Week 8 – 9

Goals

- Research examples of noisy and erroneous raster data
- Research methods to cope with noise within the context of algorithms developed so far.
- Implement modified algorithm to improve handling of noisy input.

³ <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-335.pdf>

Resources

The project requires the following:

- .NET CLI implementation, such as Microsoft .NET or Mono
- Nemerle compiler
- Example raster map data

Optional tools that facilitate development are:

- Microsoft Visual Studio 2005
- Microsoft Virtual PC
- SubVersion (source control system)
- NUnit (unit testing framework)

The Department of Computer Science's network does not have the above tools available. However, developing the project outside of the department is the author's preferred approach. Regular back-ups are critical. These must be stored externally to the main development environment. Therefore, back-up data will be copied to the department's network.