Chapter 2
Vector Data Processing

This chapter will focus on geospatial vector data and presents examples of data processing using the OGR Simple Features Library that is part of the GDAL library. We provide clear examples of how the OGR interface can be used to retrieve metadata and summary information from vector data sets, but more importantly we demonstrate how OGR is used to effectively process geospatial vector data. It is clear that there are multiple formats available for vectorial spatial data, each with their advantages and disadvantages and it is beyond the scope of this book to address each and everyone. For this reason we have decided to focus on some of the more common formats that are in use today. These include the ESRI Shapefile as it is probably the more ubiquitous format around; we also use the OpenStreetMap PBF format, Keyhole Markup Language (KML) files used within Google Earth and the Spatialite database format. Although we only focus on a number of key formats, it is important to bear in mind that you can adapt the examples to use other formats by substituting the format abbreviation within the OGR code (provided your GDAL installation is properly configured).

Initially this chapter will present a brief outline of the vector data model and continue by introducing the OGR library. We then delve into the individual OGR utilities and present examples for each showing its capabilities in terms of format and projection translation to spatial and attribute queries. The examples start off by being quite simple but gradually increase in complexity to achieve more complex processing capabilities. We anticipate that the code snippets can easily be adapted to your spatial data, formats and projection, etc.

2.1 Vector Data Model

The vector data model represents space as discrete entities (objects) that are defined by coordinates on a two dimensional plane and are described by their attributes. The three basic types of vector objects include points, lines and polygons, which are presented in Fig. 2.1.

D. McInerney and P. Kempeneers, *Open Source Geospatial Tools*,
Earth Systems Data and Models 3, DOI 10.1007/978-3-319-01824-9_2

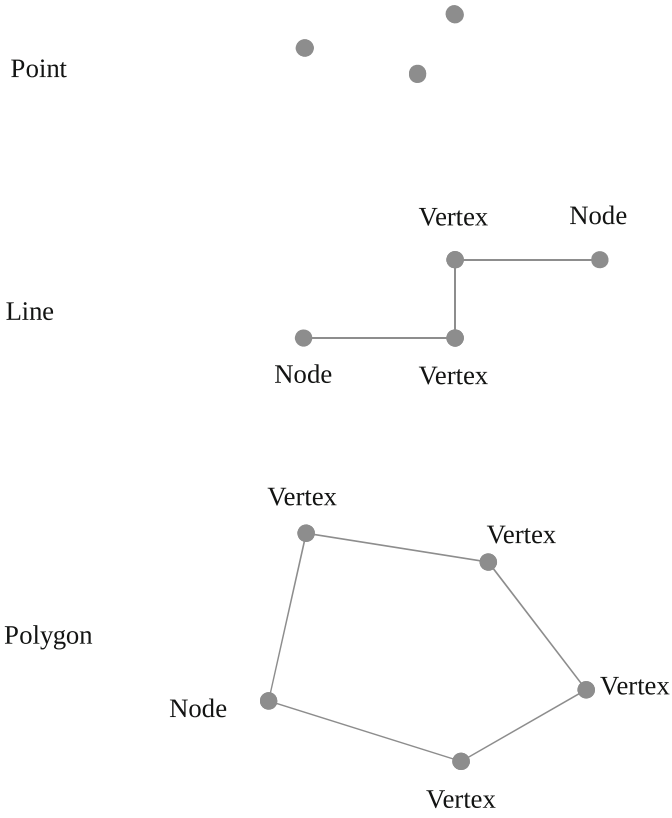

Fig. 2.1 Vector data model: *points*, *lines* and *polygons* (including nodes and vertices)

- A point can be defined either in 2- or 3-dimensions. In 2-dimensions a point consists of a single pair of coordinates that is dimensionless in terms of area and distance and its attributes are stored for each point. In 3-dimensions, a point consists of three coordinates: an x, y and z (height).
- Lines are linear features that are represented by an ordered set of coordinate pairs. Each line consists of line segments that run between adjacent coordinates in the set. In some circumstances, the starting and ending points of a line are referred to as nodes, while the intermediate points are termed vertices.
- Polygons are areas that are formed by a set of bounding line segments. Each polygon has an interior region that is bounded by its perimeter and this region can also enclose other regions. Adjacent polygons can share common borders between polygons. As is the case for the other vector features, attributes can be linked to each polygon.
- Multi-lines and multi-polygons are entities that contain several objects that are grouped together to form one feature. For instance, many islands (multiple polygons) are grouped together to form one island group.
- Multi-linestrings are entities that consist of one to many linestrings.

It is necessary to be aware that many GIS applications that support vector data models also provide topology support for vector data. Topology is the study of the relationship and adjacency of geometric vector objects. In GIS, topology refers to the relative location of spatial objects regardless of their specific location. Topological relationships (e.g. relative position, connectivity) are defined as relationships between nodes, links and polygons. In most cases, topology is stored independently of the coordinate and attribute data. For instance the topology associated with a line includes the to- and from-nodes as well as the polygons located to its right and left. There are many examples of where topology is used, for instance for network spatial analysis or to detect digitizing errors that would cause gaps and slivers (small spaces). Although GDAL/OGR does not inherently provide command line tools to check and build topology, there is the possibility to use the GEOS library[1] in conjunction with the OGR API using the Python bindings. GRASS GIS is an example of a fully topological, open source GIS.

2.2 OGR Simple Features Library

There are many toolkits and software packages that are available to process geospatial vector data. Many of the translation packages, both proprietary and open source, provide much of the same capabilities and functionality although their interfaces, format support and cost all vary. Over the last decade, OGR has emerged as one of the most stable libraries that supports a wide range of formats (both proprietary and non-proprietary) as well as many spatial reference systems.

Although the OGR utilities only consist of a few command line utilities compared to the many GDAL utilities, they provide a very rich set of functionality for managing and processing vector spatial data:

- To manipulate attribute and metadata of vector datasets;
- To translate vector datasets between formats and spatial reference systems (projections);
- To perform vector spatial analysis, such as vector overlays and creating spatial and attribute subsets.

For the purposes of this chapter, we will describe the use of the command line utilities from the OGR Simple Features Library (`ogrinfo`, `ogr2ogr` and `ogrtindex`) using a subset of vector data from OpenStreetMap[2] for the region of the Ile de France (Paris region). It is very easy to download OSM data from the main website under the Export tab or from additional data providers that are listed on the OSM wiki.[3] We will demonstrate how OGR can be used to process the OpenStreetMap data format, referred to as the Protocolbuffer Binary Format (PBF)

[1] http://geos.osgeo.org

[2] http://openstreemap.org

[3] http://wiki.openstreetmap.org/wiki/Downloading_data

data. PBF is an optimized binary format that is smaller in size than other formats used for OSM data. In addition, we will introduce the Spatialite database format as part of the examples.

2.3 ogrinfo

`ogrinfo` is the fundamental GDAL/OGR command line utility that returns information and metadata of OGR supported vector spatial data. `ogrinfo` offers a number of command line options and we will begin by a simple demonstration. To return the summary help it suffices to append `--help-general`. This will print an overview of the command line options and arguments in the shell. Throughout this section and indeed the remainder of the book, we will be taking a close look at the different command line options. When you become more familiar with the tools, you may only want to append `--help` to the command in order to return the command line options and arguments.

As we mentioned previously, GDAL/OGR is licensed under the MIT/X license, but to read the full license information in your terminal, you can type `ogrinfo --license`, which details license those related to files within the GDAL/OGR source tree. To check the version of GDAL/OGR that you are using, simply type `ogrinfo --version` on the command line, which will indicate the GDAL library version number with its release date. The help message of ogrinfo is as follows:

```
ogrinfo --help-general

Generic GDAL/OGR utility command options:
  --version: report version of GDAL/OGR in use.
  --license: report GDAL/OGR license info.
  --formats: report all configured format drivers.
  --optfile filename: expand an option file into the argument
    ↪ list.
  --config key value: set system configuration option.
  --debug [on/off/value]: set debug level.
  --pause: wait for user input, time to attach debugger
  --locale [locale]: install locale for debugging (i.e.
    ↪ en_US.UTF-8)
  --help-general: report detailed help on general options.


ogrinfo [--help-general] [-ro] [-q] [-where restricted_where]
[-spat xmin ymin xmax ymax] [-fid fid]
[-sql statement] [-dialect sql_dialect]
[-al] [-so] [-fields={YES/NO}]
[-geom={YES/NO/SUMMARY}][--formats]
datasource_name [layer [layer ...]]
```

The full list of the supported command line options and their explanations are provided below as per the GDAL/OGR documentation.

-ro
Open the data source in read-only mode.

-al
List all features of all layers (used instead of having to give layer names as arguments).

-so
Summary Only: supress listing of features, show only the summary information like projection, schema, feature count and extents.

-q
Quiet verbose reporting of various information, including coordinate system, layer schema, extents, and feature count.

-where
restricted_where: An attribute query in a restricted form of the queries used in the SQL WHERE statement. Only features matching the attribute query will be reported.

-sql statement
Execute the indicated SQL statement and return the result.

-dialect dialect
SQL dialect. In some cases can be used to use (unoptimized) OGR SQL instead of the native SQL of an RDBMS by passing OGRSQL. Starting with GDAL 1.10, the "SQLITE" dialect can also be used with any datasource.

-spat xmin ymin xmax ymax
The area of interest. Only features within the rectangle will be reported.

-geomfield field
(from GDAL 1.11) Name of the geometry field on which the spatial filter operates on.

-fid
fid: If provided, only the feature with this feature id will be reported. Operates exclusive of the spatial or attribute queries. Note: if you want to select several features based on their feature id, you can also use the fact the 'fid' is a special field recognized by OGR SQL. So, '-where "fid in (1,3,5)"' would select features 1, 3 and 5.

-fields=YESNO
(starting with GDAL 1.6.0) If set to NO, the feature dump will not display field values. Default value is YES.

-geom=YESNOSUMMARY
(starting with GDAL 1.6.0) If set to NO, the feature dump will not display the geometry. If set to SUMMARY, only a summary of the geometry will be displayed. If set to YES, the geometry will be reported in full OGC WKT format. Default value is YES.

--formats
List the format drivers that are enabled.

datasource_name
The data source to open. May be a filename, directory or other virtual name. See the OGR Vector Formats list for supported datasources.

layer
One or more layer names may be reported.
If no layer names are passed then ogrinfo will report a list of available layers (and their layerwide geometry type). If layer name(s) are given then their extents, coordinate system, feature count, geometry type, schema and all features matching query parameters will be reported to the terminal. If no query parameters are provided, all features are reported. Geometries are reported in OGC WKT format.

To list the vector formats supported by your GDAL/OGR installation, type `ogrinfo --formats`, which will output a full list of all of the supported formats and related information. The text between the double quotes refers to an abbreviation of the vector data format. The text between parentheses indicates if OGR can read and/or write this format.

```
ogrinfo --formats

Supported Formats:
  -> "ESRI Shapefile" (read/write)
  -> "MapInfo File" (read/write)
  -> "UK .NTF" (readonly)
  -> "SDTS" (readonly)
  -> "TIGER" (read/write)
  -> "S57" (read/write)
  -> "DGN" (read/write)
  -> "VRT" (readonly)
  -> "REC" (readonly)
  -> "Memory" (read/write)
  -> "BNA" (read/write)
  -> "CSV" (read/write)
  -> "GML" (read/write)
  -> "GPX" (read/write)
  -> "KML" (read/write)
  -> "GeoJSON" (read/write)
  -> "GMT" (read/write)
  -> "SQLite" (read/write)
  ...
```

You should check that 'OSM' is listed to ensure that you will be able to follow the subsequent examples and exercise. The GDAL installation on the OSGeo VM provides this support by default, but if you have a customized installation of GDAL, you may need to check your installation.

If you find that your installation does not include a specific format that you require for your analysis, then you will need to customize your GDAL installation by compiling and installing it from source. A brief explanation of this is provided in Appendix B.2.

The most common vector data format is the ESRI Shapefile, which is a binary format that was developed by ESRI and has been available since the early 1990s. It has become one of the most widely used vector formats within the geospatial domain and is consequently read by a plethora of GIS packages. Despite its widespread use, it is now considered to have some limitations with respect to other types of vector spatial data formats. For instance, field name lengths are limited to eight characters and lack of topology is another limitation. Furthermore, the Shapefile consists of several files, a minimum of three (.shp, .shx and .dbf), but commonly five or six; each with the same filename but differing file extensions.

`ogrinfo` can also be used to retrieve basic information about vector spatial data sets. The following examples are described using a subset of the OpenStreetMap data. Unlike the ESRI Shapefile (Shp), the PBF contains many layers that are contained within one file, which is very useful in terms of portability. It is possible to quickly check the geometries of the individual layers stored in the OSM file by passing the `-q` to `ogrinfo`:

```
ogrinfo -q ile-de-france-latest.osm.pbf

    1: points (Point)
    2: lines (Line String)
    3: multilinestrings (Multi Line String)
    4: multipolygons (Multi Polygon)
    5: other_relations (Geometry Collection)
```

Using the OSM PBF data, we can retrieve basic metadata information using `ogrinfo`. To print the basic information about the OSM PBF's dataset coordinate reference system, fields etc., you simply pass the command line option `-so` to ogrinfo followed by the name of the file as follows. The `-al` specifies that information about all layers should be printed, while `-so` prints a summary of the information. (Note only the output for the point here, but the full output will be printed in your shell).

```
ogrinfo -al -so ile-de-france-latest.osm.pbf

Had to open data source read-only.
INFO: Open of 'ile-de-france-latest.osm.pbf'
      using driver 'OSM' successful.

Layer name: points
Geometry: Point
Feature Count: -1
Extent: (1.446244, 48.121800) - (3.558616, 49.240500)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
```

```
        AUTHORITY["EPSG","9108"]],
    AUTHORITY["EPSG","4326"]]
osm_id: String (0.0)
name: String (0.0)
barrier: String (0.0)
highway: String (0.0)
ref: String (0.0)
address: String (0.0)
is_in: String (0.0)
place: String (0.0)
man_made: String (0.0)
other_tags: String (0.0)

Layer name: lines
Geometry: Line String
Feature Count: -1
Extent: (1.446244, 48.121800) - (3.558616, 49.240500)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9108"]],
    AUTHORITY["EPSG","4326"]]
osm_id: String (0.0)
name: String (0.0)
highway: String (0.0)
waterway: String (0.0)
aerialway: String (0.0)
barrier: String (0.0)
man_made: String (0.0)
other_tags: String (0.0)
```

The output shows that the dataset contains five layers (or feature types) (points, lines, multilinestrings, multipolygons and other_relations) that represent different objects. For instance, if you are interested in only printing detailed information about the multipolygons (e.g. geographic extent), it suffices to append the feature type to the above-mentioned command:

```
ogrinfo -al -so ile-de-france-latest.osm.pbf multipolygons
```

And similarly, it is possible to return information for a region of interest by passing the option, `-spat`, followed by the coordinates of the bounding box defined as `xmin ymin xmax ymax`. This can be particularly useful if you have a very large dataset (e.g. global) and you wish to return information for a subset. As the name indicates, `ogrinfo` returns information relating to vector based spatial datasets. Information regarding subsets of vector data can be retrieved using the `-sql` option, which is described subsequently in Sect. 2.4.1. More complex analysis and processing is described in the next section using `ogr2ogr`, where command line arguments such

as `-sql, -where, -dialect` are also available and demonstrated there. We will return to `ogrinfo` during the case studies.

2.4 ogr2ogr

Although the `ogr2ogr` is only one command with a large set of command line options, in its simplest form it can be used to reproject and subset spatial datasets. Its versatility and flexibility extends to allow complex spatial and attribute filters to be passed as command line arguments as well as to perform simple vector based spatial analysis. The full list of command line arguments outlined below can be produced using `ogr2ogr --help`. As a precursor to this section, the general syntax for `ogr2ogr` is to specify the destination datasource name first (`dst_datasource_name`) followed by the input or source datasource name (`src_datasource_name`):

```
Usage: ogr2ogr [--help-general] [-skipfailures] [-append]
    ↪ [-update]
                [-select field_list] [-where restricted_where]
                [-progress] [-sql <sql statement>] [-dialect
    ↪ dialect]
                [-preserve_fid] [-fid FID]
                [-spat xmin ymin xmax ymax]
                [-a_srs srs_def] [-t_srs srs_def] [-s_srs
    ↪ srs_def]
                [-f format_name] [-overwrite] [[-dsco NAME=VALUE]
    ↪  ...]
                dst_datasource_name src_datasource_name
                [-lco NAME=VALUE] [-nln name] [-nlt type] [-dim
    ↪ 2|3] [layer [layer ...]]

Advanced options :
                [-gt n]
                [-clipsrc [xmin ymin xmax
    ↪ ymax]|WKT|datasource|spat_extent]
                [-clipsrcsql sql_statement] [-clipsrclayer layer]
                [-clipsrcwhere expression]
                [-clipdst [xmin ymin xmax ymax]|WKT|datasource]
                [-clipdstsql sql_statement] [-clipdstlayer layer]
                [-clipdstwhere expression]
                [-wrapdateline][-datelineoffset val]
                [[-simplify tolerance] | [-segmentize max_dist]]
                [-fieldTypeToString All|(type1[,type2]*)]
                [-fieldmap identity | index1[,index2]*]
                [-splitlistfields] [-maxsubfields val]
                [-explodecollections] [-zfield field_name]
                [-gcp pixel line easting northing [elevation]]*
    ↪ [-order n | -tps]

Note: ogr2ogr --long-usage for full help.
```

The description of each command line option and argument (from the official GDAL website) is:

-f
format_name output file format name (default is ESRI Shapefile), some possible values are: -f “ESRI Shapefile”; -f “TIGER”; -f “MapInfo File”
-append
Append to existing layer instead of creating new
-overwrite
Delete the output layer and recreate it empty
-update
Open existing output datasource in update mode rather than trying to create a new one
-select field_list
Comma delimited list of fields from input layer to copy to the new layer. A field is skipped if mentioned previously in the list even if the input layer has duplicate field names. (Defaults to all; any field is skipped if a subsequent field with same name is found.) Starting with OGR 2.0, geometry fields can also be specified in the list.
-progress
(starting with GDAL 1.7.0) Display progress on terminal. Only works if input layers have the “fast feature count” capability.
-sql sql_statement
SQL statement to execute. The resulting table/layer will be saved to the output.
-dialect dialect
SQL dialect. In some cases can be used to use (unoptimized) OGR SQL instead of the native SQL of an RDBMS by passing OGRSQL. Starting with GDAL 1.10, the “SQLITE” dialect can also be used with any datasource.
-where restricted_where
Attribute query (like SQL WHERE)
-skipfailures
Continue after a failure, skipping the failed feature.
-spat xmin ymin xmax ymax
spatial query extents. Only features whose geometry intersects the extents will be selected. The geometries will not be clipped unless -clipsrc is specified
-geomfield field
(from OGR 1.11) Name of the geometry field on which the spatial filter operates on.
-dsco NAME=VALUE
Dataset creation option (format specific)
-lco NAME=VALUE
Layer creation option (format specific)
-nln name
Assign an alternate name to the new layer
-nlt type
Define the geometry type for the created layer. One of NONE, GEOMETRY,

POINT, LINESTRING, POLYGON, GEOMETRYCOLLECTION, MULTIPOINT, MULTIPOLYGON or MULTILINESTRING. Add "25D" to the name to get 2.5D versions. Starting with GDAL 1.10, PROMOTE_TO_MULTI can be used to automatically promote layers that mix polygon or multipolygons to multipolygons, and layers that mix linestrings or multilinestrings to multilinestrings. Can be useful when converting shapefiles to PostGIS (and other target drivers) that implements strict checks for geometry type.

-dim val
(starting with GDAL 1.10) Force the coordinate dimension to val (valid values are 2 or 3). This affects both the layer geometry type, and feature geometries. Starting with GDAL 2.0, the value can be set to "layer_dim" to instruct feature geometries to be promoted to the coordinate dimension declared by the layer.

-a_srs srs_def
Assign an output SRS

-t_srs srs_def
Reproject/transform to this SRS on output

-s_srs srs_def
Override source SRS

-preserve_fid
Use the FID of the source features instead of letting the output driver to automatically assign a new one.

-fid fid
If provided, only the feature with this feature id will be reported. Operates exclusive of the spatial or attribute queries. Note: if you want to select several features based on their feature id, you can also use the fact the 'fid' is a special field recognized by OGR SQL. So, '-where "fid in (1,3,5)"' would select features 1, 3 and 5. Srs_def can be a full WKT definition (hard to escape properly), or a well known definition (ie. EPSG:4326) or a file with a WKT definition. Advanced options :

gt n
group n features per transaction (default 200). Increase the value for better performance when writing into DBMS drivers that have transaction support.

-clipsrc
xmin ymin xmax ymax PIPE WKT datasource PIPE spat_extent: (starting with GDAL 1.7.0) clip geometries to the specified bounding box (expressed in source SRS), WKT geometry (POLYGON or MULTIPOLYGON), from a datasource or to the spatial extent of the -spat option if you use the spat_extent keyword. When specifying a datasource, you will generally want to use it in combination of the -clipsrclayer, -clipsrcwhere or -clipsrcsql options

-clipsrcsql sql_statement
Select desired geometries using an SQL query instead.

-clipsrclayer layername
Select the named layer from the source clip datasource.

-clipsrcwhere expression
Restrict desired geometries based on attribute query.
-clipdst xmin ymin xmax ymax
(starting with GDAL 1.7.0) clip geometries after reprojection to the specified bounding box (expressed in dest SRS), WKT geometry (POLYGON or MULTIPOLYGON) or from a datasource. When specifying a datasource, you will generally want to use it in combination of the -clipdstlayer, -clipdstwhere or -clipdstsql options
-clipdstsql sql_statement
Select desired geometries using an SQL query instead.
-clipdstlayer layername
Select the named layer from the destination clip datasource.
-clipdstwhere expression
Restrict desired geometries based on attribute query.
-wrapdateline
(starting with GDAL 1.7.0) split geometries crossing the dateline meridian (long. $= +/- 180$deg)
-datelineoffset
(starting with GDAL 1.10) offset from dateline in degrees (default long. $= +/-$ 10deg, geometries within 170deg to -170deg will be splited)
-simplify tolerance
(starting with GDAL 1.9.0) distance tolerance for simplification. Note: the algorithm used preserves topology per feature, in particular for polygon geometries, but not for a whole layer.
-segmentize max_dist
(starting with GDAL 1.6.0) maximum distance between 2 nodes. Used to create intermediate points
-fieldTypeToString type1, ...
(starting with GDAL 1.7.0) converts any field of the specified type to a field of type string in the destination layer. Valid types are : Integer, Real, String, Date, Time, DateTime, Binary, IntegerList, RealList, StringList. Special value All can be used to convert all fields to strings. This is an alternate way to using the CAST operator of OGR SQL, that may avoid typing a long SQL query.
-unsetFieldWidth
(starting with GDAL 2.0) set field width and precision to 0.
-splitlistfields
(starting with GDAL 1.8.0) split fields of type StringList, RealList or IntegerList into as many fields of type String, Real or Integer as necessary.
-maxsubfields val
To be combined with -splitlistfields to limit the number of subfields created for each split field.
-explodecollections
(starting with GDAL 1.8.0) produce one feature for each geometry in any kind of geometry collection in the source file

-zfield field_name
(starting with GDAL 1.8.0) Uses the specified field to fill the Z coordinate of geometries
-gcp ungeoref_x ungeoref_y georef_x georef_y elevation
(starting with GDAL 1.10.0) Add the indicated ground control point. This option may be provided multiple times to provide a set of GCPs.
-order n
(starting with GDAL 1.10.0) Order of polynomial used for warping (1 to 3). The default is to select a polynomial order based on the number of GCPs.
-tps
(starting with GDAL 1.10.0) Force use of thin plate spline transformer based on available GCPs.
-fieldmap
(starting with GDAL 1.10.0) Specifies the list of field indexes to be copied from the source to the destination. The (n)th value specified in the list is the index of the field in the target layer definition in which the n(th) field of the source layer must be copied. Index count starts at zero. There must be exactly as many values in the list as the count of the fields in the source layer. We can use the 'identity' setting to specify that the fields should be transferred by using the same order. This setting should be used along with the -append setting.
-addfields
(starting with GDAL 1.11) This is a specialized version of -append. Contrary to -append, -addfields has the effect of adding, to existing target layers, the new fields found in source layers. This option is usefull when merging files that have non-strictly identical structures. This might not work for output formats that don't support adding fields to existing non-empty layers.

A frequent operation that geospatial analysts are faced with is to convert spatial data from one format into another. This can be done very easily using `ogr2ogr`. The following example is the most basic application of `ogr2ogr` and demonstrates how an OpenStreetMap PBF dataset can be converted into an ESRI Shapefile. As specified above, the general syntax is to first specify the destination datasource name followed by the source destination.

```
ogr2ogr  ile-de-france.shp ile-de-france-latest.osm.pbf
    ↪ multipolygons
```

It is important to note that by converting the PBF dataset to an ESRI Shapefile, it creates a directory named `ile-de-france.shp`, which contains individual ESRI Shapefiles for each of the vector feature types listed below. This is done since ESRI Shapefiles can only hold one feature type per file, whereas other formats like Spatialite can hold many feature types within the one file:

- landuse
- natural

- places
- points
- railways
- roads
- waterways.

It is quite common to either download vector spatial data from the Internet or to receive a dataset that is not in the Spatial Reference System (SRS) that you require for your particular purposes. As a result, it is necessary to reproject the dataset into another SRS. For instance, you may want to reproject an OSM dataset that is projected to the WGS84 (EPSG:4326) projection to the Spherical Mercator projection (EPSG:3857), which is used by Web mapping services like Google Maps. The codes[4] are read by GDAL/OGR and provide a succinct way of specifying the Spatial Reference System. The following example demonstrates how to convert the multipolygons within the OSM dataset using `ogr2ogr`.

```
ogr2ogr  -s_srs "EPSG:4326"-t_srs"EPSG:3857"
    ↪ ile-de-france-latest_3857.osm.pbf
    ↪ ile-de-france-latest.osm.pbf multipolygons
```

2.4.1 Manipulating Data

Both `ogrinfo` and `ogr2ogr` provide a very useful SQL (Structured Query Language) interface, which permits the use of SQL statements on the command line to create subsets of datasets defined by SQL queries. GDAL 1.8.0 saw the biggest improvement in terms of the SQL support, with older versions not supporting arithmetic and field expressions. The following examples will demonstrate how both `ogrinfo` and `ogr2ogr` can be used with the SQL filter. For instance, to query a specific field in a vector dataset we can first use `ogrinfo` to list the fields and then run an SQL expression on one or more of the fields. In the following example, we use `ogrinfo` to identify the unique types of natural classes. We start by retrieving the attributes of the 'natural' features from the ESRI Shapefile version of the OSM dataset.

```
ogrinfo -so -al natural.shp

    osm_id: Real (11.0)
    name: String (48.0)
    type: String (16.0)
```

We then use `ogrinfo` to return the list of unique types of features labelled as "natural" using the following SQL statement.

[4] http://epsg.io/

```
ogrinfo  natural.shp -sql 'select DISTINCT type from natural'

    INFO: Open of 'natural.shp'
        using driver 'ESRI Shapefile' successful.

    Layer name: natural
    Geometry: Polygon
    Feature Count: 4
    Layer SRS WKT:
    (unknown)
    type: String (16.0)
    OGRFeature(natural):0
      type (String) = riverbank

    OGRFeature(natural):1
      type (String) = park

    OGRFeature(natural):2
      type (String) = water

    OGRFeature(natural):3
      type (String) = forest
```

The results indicate that there are four classifications of natural areas within the dataset that include: riverbanks, parks, water and forests. In the next example, the SQL statement is adapted to count the number of forests polygons within the `natural.shp` dataset using the COUNT SQL operator (7566).

```
#return a list of unique natural classes from the 'type' field
ogrinfo  natural.shp -sql 'select COUNT(type) from natural WHERE
    ↪  type = "forest"'

    INFO: Open of 'natural.shp'
        using driver 'ESRI Shapefile' successful.

    Layer name: natural
    Geometry: Polygon
    Feature Count: 1
    Layer SRS WKT:
    (unknown)
    COUNT_type: Integer (0.0)
    OGRFeature(natural):0
     COUNT_type (Integer) = 7566
```

In some instances, you may want to extract all features from a dataset that match a particular query. For instance, all polygons above a certain area threshold or all features of a particular classification. At this stage, it is important to hightlight two different filters that can be used with `ogr2ogr`, which is `-select` and `-where`. The main distinction between the two is that `-where` selects a subset of the features including all attributes and outputs them to the destination data source, whereas `-select` outputs all features, but with a subset of the attributes. In other words, unless the wildcard (`*`) is used, each field name that should be written to the desti-

nation file, needs to be explicitly specified. For instance, to select all forest polygons from the natural.shp dataset we can do:

```
ogr2ogr natural_forest.shp natural.shp -where 'type="forest"'
ogrinfo -so -al natural_forest.shp

    Layer name: natural_forest
    Geometry: Polygon
    Feature Count: 7566
    Extent: (1.393187, 48.113148) - (3.576257, 49.237916)
    Layer SRS WKT:
    GEOGCS["GCS_WGS_1984",
        DATUM["WGS_1984",
        SPHEROID["WGS_84",6378137,298.257223563]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]]
    osm_id: Real (11.0)
    name: String (48.0)
    type: String (16.0)
```

Whereas using `-select`, it is possible to create an output dataset that contains a subset of attributes that have been specified:

```
ogr2ogr natural_select.shp natural.shp -select 'osm_id, name'
oginfo -so -al natural_select.shp

INFO: Open of 'natural_select.shp'
using driver 'ESRI Shapefile' successful.

Layer name: natural_select
Geometry: Polygon
Feature Count: 17259
Extent: (1.393187, 48.113148) - (3.576257, 49.237916)
Layer SRS WKT:
GEOGCS["GCS_WGS_1984",
    DATUM["WGS_1984",
        SPHEROID["WGS_84",6378137,298.257223563]],
    PRIMEM["Greenwich",0],
    UNIT["Degree",0.017453292519943295]]
osm_id: Real (11.0)
name: String (48.0)
```

It is useful to note that both `-select` and `-where` can be combined within the one command. For instance, we could merge the above examples to select a subset of all of the forest polygons from the `natural.shp` and output a subset of the attributes.

```
ogr2ogr natural_select_subset.shp natural.shp -select 'osm_id,
    ↪ name' -where 'type="forest"'
oginfo -so -al natural_select_subset.shp

    INFO: Open of 'natural_select_subset.shp'
      using driver 'ESRI Shapefile' successful.

Layer name: natural_select_subset
```

```
Geometry: Polygon
Feature Count: 7566
Extent: (1.393187, 48.113148) - (3.576257, 49.237916)
Layer SRS WKT:
GEOGCS["GCS_WGS_1984",
    DATUM["WGS_1984",
        SPHEROID["WGS_84",6378137,298.257223563]],
    PRIMEM["Greenwich",0],
    UNIT["Degree",0.017453292519943295]]
osm_id: Real (11.0)
name: String (48.0)
```

The option `-sql` provides `ogr2ogr` with a lot of flexibilty in terms of the queries that can be run on the input dateset. The following examples will demonstrate how to carry out a range of data processing steps with SQL. Supposing you want to rename the field `name` from the above example to something that is more explantorary, such as `osm_name`, it suffices to append the following SQL statement to `ogr2ogr`.

```
ogr2ogr natural_select_rename.shp natural_select.shp -sql
    ↪ "SELECT name AS osm_name FROM natural_select"
```

The words in capitals in the SQL statement are core words from SQL and the words: `name`, `osm_name` and `natural_select` represent variables names, the latter indicating the name of the dataset. The use of capitals for SQL terms is not mandatory, but is more of a convention. Similarly, the `SQL` statement can be adapted to calculate the area of the polygon using the `OGR_GEOM_AREA` special field.

As of GDAL 1.10, both `ogrinfo` and `ogr2ogr` support the SQLite dialect to query Shapefiles and other OGR supported vector formats. The advantage over SQL is that it provides more functionality as well as the spatial functions. In the following example we use the SQLite dialect to return the coordinates of the centroid and area of the polygon `natural_select_rename.shp` using the functions `ST_Centroid` and `ST_Area`. It is important to note that as input each of the functions take the *geometry* as input.

```
 ogrinfo natural_select_rename.shp -dialect SQLite -sql "select
    ↪ ST_Centroid(geometry), SUM(ST_Area(geometry)) from
    ↪ natural_select_rename"
Layer name: SELECT
Geometry: Unknown (any)
Feature Count: 1
Extent: (3.116159, 48.947486) - (3.116159, 48.947486)
Layer SRS WKT:
GEOGCS["WGS 84",
   DATUM["WGS_1984",
       SPHEROID["WGS 84",6378137,298.257223563,
           AUTHORITY["EPSG","7030"]],
       AUTHORITY["EPSG","6326"]],
   PRIMEM["Greenwich",0,
       AUTHORITY["EPSG","8901"]],
   UNIT["degree",0.0174532925199433,
       AUTHORITY["EPSG","9122"]],
   AUTHORITY["EPSG","4326"]]
```

```
Geometry Column = ST_Centroid (geometry)
SUM(ST_Area (geometry)): Real (0.0)
OGRFeature (SELECT): 0
 SUM(ST_Area (geometry)) (Real) = 0.252206490693434
 POINT (3.116159337389504 48.94748608019286)
```

Vector spatial analysis typically includes merging, clipping and intersecting vectors to generate new sets of data. These geometric intersections are illustrated in Fig. 2.2. Some of these methods can be done using `ogr2ogr`, while others can be implemented using OGR and/or the GEOS library. We demonstrate some examples using `ogr2ogr`.

Clipping can be used to create a vector subset of the features using either the spatial extent or another vector dataset. **Merging** multiple shapefiles into one shapefile is often required during geospatial analysis. For instance, multiple administrative boundaries need to be combined into one file or multiple sets of points need to be

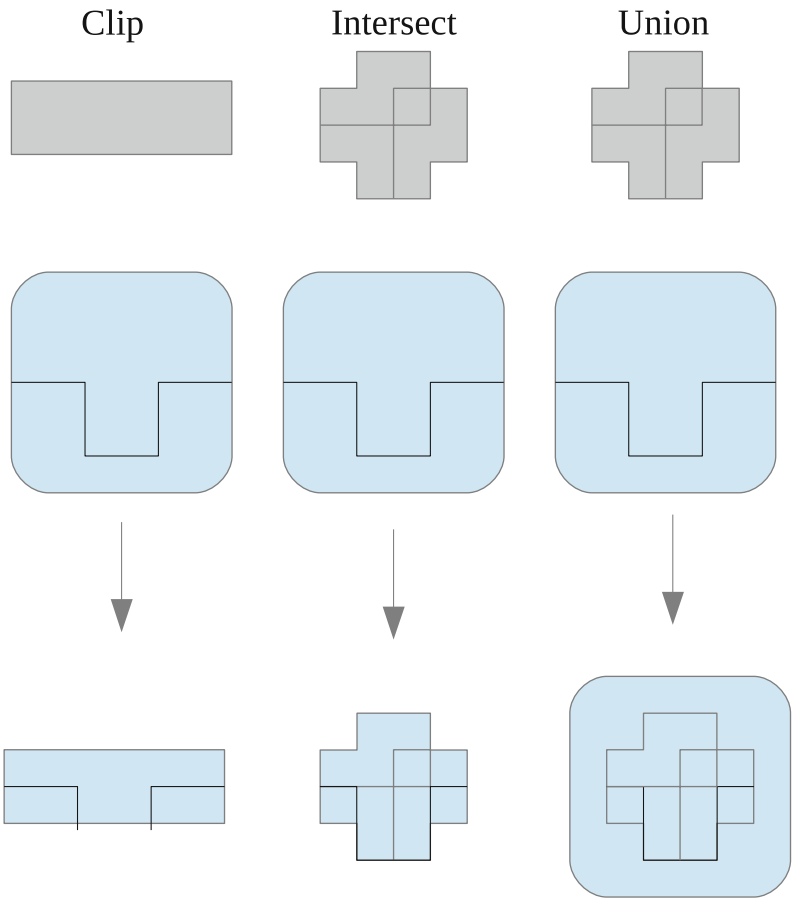

Fig. 2.2 Vector overlay operations

merged. The key arguments to pass to `ogr2ogr` are `-update` and `-append`. In this way, a Shapefile is appended to the destination merged Shapefile, which is updated. The following bash script demonstrates how this can be done effectively by searching for all of the Shapefiles in the working directory and appending them to the output, merged shapefile located in the directory `merged`.

```
#!/bin/bash
    mkdir merged_shps
    ogr2ogr -f 'ESRI Shapefile' merged_shps/merged.shp first.shp
    for file in $(ls subsequent*.shp);
    do
       ogr2ogr -f  ESRI Shapefile   -update -append
       ↪ merged/merged.shp $file -nln Merged
    done
```

In many instances, a spatial subset of a dataset is required to extract a section of a dataset for a region of interest, which contains only the features located within a specific area and thereby reduces the file size. This can be achieved using `ogr2ogr` by passing a list of coordinates that specify the bounding box of interest using the `-clipsrc` argument followed by the list of coordinates specified in the following order `xmin ymin xmax ymax` which respectively represents the minimal x, y and maximum x, y of the bounding box. In the subsequent sections we will demonstrate how other forms of clip operations can be performed using vector datasets.

```
ogr2ogr subset_ile-de-france.shp ile-de-france-latest.osm.pbf
       ↪ multipolygons -clipsrc 1.7 48.63 2.13 48
```

2.5 ogrtindex

The `ogrtindex` program can be used to create a tileindex—a file containing a list of the identities of several other files along with their spatial extents. A tile index is generally used by a Web Mapping System as means of optimizing the speed of data transfer based on location rather than loading the full vector dataset and impacting on performance.

```
Usage: ogrtindex [-lnum n] [-lname name] [-f output_format]
                 [-write_absolute_path]
       ↪ [-skip_different_projection]
                 [-accept_different_schemas]
                 output_dataset src_dataset...
```

-lnum n
Add layer number 'n' from each source file in the tile index.
-lname name
Add the layer named 'name' from each source file in the tile index.
-f output_format
Select an output format name. The default is to create a shapefile.
-tileindex field_name
The name to use for the dataset name. Defaults to LOCATION.
-write_absolute_path
Filenames are written with absolute paths
-skip_different_projection
Only layers with same projection ref as layers already inserted in the tileindex will be inserted. If no `-lnum` or `-lname` arguments are given it is assumed that all layers in source datasets should be added to the tile index as independent records.
If the tile index already exists it will be appended to, otherwise it will be created. It is a flaw of the current ogrtindex program that no attempt is made to copy the coordinate system definition from the source datasets to the tile index (as is expected by MapServer when PROJECTION AUTO is in use).

Assuming we wanted to create a tile index for the following Shapefiles exported from the OpenStreetMap dataset for the Ile de France region, we would need to either navigate to directory or use the relative path.

```
ls *.shp
landuse.shp
natural.shp
places.shp
points.shp
railways.shp
roads.shp
tileindex.shp
waterways.shp
```

We would then run the following command where we specify the output format as an ESRI Shapefile. We also define the new attribute that holds the file path as "Location". The output file name is set to `ile_de_france_tileindex.shp` and the wildcard `*.shp` indicates that all ESRI Shapefiles located within the directory should be added to the tile index. The option `accept_different_schemas` is important if the attributes in the individual Shapefiles differs, but it is necessary to note that this may cause an incompatability with UMN MapServer.

```
ogrtindex -f 'ESRI Shapefile' -tileindex 'Location'
       ↪ -accept_different_schemas ile_de_france_tileindex.shp
       ↪ *.shp
```

The result is a polygon Shapefile with each polygon feature defining the extent of the input Shapefile. The `ogrinfo` command summarises the dataset containing the 10 features.

```
 ogrinfo -so -al tileindex.shp

INFO: Open of 'tileindex.shp'
      using driver 'ESRI Shapefile' successful.

Layer name: tileindex
Geometry: Polygon
Feature Count: 10
Extent: (0.000000, 0.000000) - (3.942579, 49.286794)
Layer SRS WKT:
GEOGCS["GCS_WGS_1984",
    DATUM["WGS_1984",
        SPHEROID["WGS_84",6378137,298.257223563]],
    PRIMEM["Greenwich",0],
    UNIT["Degree",0.017453292519943295]]
LOCATION: String (200.0)
```

2.6 OGR Virtual Format

The virtual format driver available in OGR is similar to the GDAL VRT described in Chap. 11, but applies specifically to vector datasets. You can check that your GDAL installation supports this driver by checking that `-> "VRT" (readonly)` is listed under `ogrinfo --formats`.

This driver is mainly used to transform features from other formats using a specification in an XML file. In many cases, it is used to create spatial layers from geocoded data stored in ASCII files. For instance, you may have a series of geocordinates representing field sampling plots that have associated measurements or a set of coordinates representing a GPS path. The virtual format provides the ability for different datasets to be merged and to associate a CRS with each dataset.

In the following example we will demonstrate how the OGR VRT format can be used to generate a virtual dataset from an ASCII or CSV file. Supposing we want to create a virtual dataset using the following text file (`gps_paths.csv`) that contains a set of lines using the Well Known Text (WKT) encoding.[5]

[5] WKT is text mark-up language for vector geometries that was originally defined by the OGC in their specifications for Simple Feature Access and Coordinate Transformation Services. Both WKT and the associated Well Known Binary (WKB) can be used to store the same objects in a relational database management system, with the latter version being optimized for storage.

```
line_id;Line_xy
1;LINESTRING(189957 234427, 128433 190130, 132124 174133, 166578
      ↪  234427)
2;LINESTRING(89957 34427, 90123 36232, 95000 37344, 96132 39800)
```

We then need to create the XML file that will act as the OGR VRT driver. The following example includes the minimum tags that are required to create the driver, which are `OGRVRTDataSource`, `SrcDataSource`, `OGRVRTLayer`. We also specify a layername, which is "gps_paths".

```
<OGRVRTDataSource>
    <OGRVRTLayer name="gps_paths">
        <SrcDataSource Separator="SEMICOLON">gps_paths.csv</
      ↪ SrcDataSource>
        <GeometryType>wkbMultiLineString</GeometryType>
        <LayerSRS>EPSG:29900</LayerSRS>
        <GeometryField encoding="WKT"field="Lines"/>
    </OGRVRTLayer>
<!-- FINAL XML TAG -->
</OGRVRTDataSource>
```

It is important to note that the name specified in the `OGRVRTLayer` tag needs to be the same as the input file name (without the file extension). Although the `GeometryType` and `LayerSRS` are not mandatory, it is good practice to include. We can quickly check that the VRT has been created correctly using `ogrinfo virtual.vrt`.

```
INFO: Open of 'virtual.vrt'
      using driver 'VRT' successful.

Layer name: gps_paths
Geometry: Multi Line String
Feature Count: 2
Extent: (89957.000000, 34427.000000) - (189957.000000,
      ↪ 234427.000000)
Layer SRS WKT:
PROJCS["TM65 / Irish National Grid (deprecated)",
    GEOGCS["TM65",
        DATUM["TM65",
            SPHEROID["Airy Modified
      ↪ 1849",6377340.189,299.3249646,
                AUTHORITY["EPSG","7002"]],
            TOWGS
      ↪ 84[482.5,-130.6,564.6,-1.042,-0.214,-0.631,8.15],
            AUTHORITY["EPSG","6299"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4299"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",53.5],
```

```
    PARAMETER["central_meridian",-8],
    PARAMETER["scale_factor",1.000035],
    PARAMETER["false_easting",200000],
    PARAMETER["false_northing",250000],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","29900"]]
line_id: String (0.0)
Lines: String (0.0)
```

It is often necessary to generate a spatial layer from a set of coordinates that may be stored as a comma separated value (CSV). For instance you may have a list of field plots defined by their geo-coordinates with associated attributes. We can extend the VRT created previously to also hold a set of plot data (plots.csv) listed as follows:

```
X;Y;ID;LUT
199223;224322;1;Forest
56102;102324;2;Water
250253;254203;3;Agriculture
199212;134242;4;Forest
```

We can then append a new layer to the VRT.

```
<OGRVRTDataSource>
    ...
        <LayerSRS>EPSG:29900</LayerSRS>
        <GeometryField encoding="WKT"field="Lines"/>
    </OGRVRTLayer>
    <OGRVRTLayer name="plots">
        <SrcDataSource
       ↪ Separator="SEMICOLON">plots.csv</SrcDataSource>
        <GeometryType>wkbPoint</GeometryType>
        <LayerSRS>EPSG:29900</LayerSRS>
        <GeometryField encoding="PointFromColumns"x="X"y="Y"/>
    </OGRVRTLayer>
<!-- FINAL XML TAG -->
</OGRVRTDataSource>
```

To check its contents, we use `ogrinfo`:

```
ogrinfo -so virtual.vrt

INFO: Open of 'virtual.vrt'
      using driver 'VRT' successful.
1: gps_paths (Multi Line String)
2: plots (Point)
```

One of the particular advantages of the VRT driver is that it is possible to register a variety of different data types (points, lines and polygons) and formats within a single VRT. We will continue our example by adding the county boundaries for Ireland from the GADM website.[6] These are available for download as ESRI Shapefiles projected in WGS84 (EPSG:4326), but we use the VRT to reproject them to the Irish National Grid (EPSG:29900) on the fly.

```
 ...
<OGRVRTLayer name="Ireland_Admin">
        <SrcDataSource>./IRL_adm0.shp</SrcDataSource>
        <SrcLayer>IRL_adm0</SrcLayer>
        <TargetSRS>EPSG:29900</TargetSRS>
    </OGRVRTLayer>
<!-- FINAL XML TAG -->
</OGRVRTDataSource>
```

We can go a step further with the VRT and generate another virtual layer that represents the convex hull of county boundary of Clare (located on the west coast of Ireland). This can be achieved using the SQLite dialect within the `SrcSQL` tag. The output of these administrative datasets are presented in Fig. 2.3 and the convex hull generated for county Clare (Fig. 2.4).

```
 ...
 <OGRVRTLayer name="Convex_Ireland_Admin">
        <SrcDataSource>./IRL_adm1.shp</SrcDataSource>
        <SrcLayer>IRL_adm1</SrcLayer>
        <SrcSQL dialect="sqlite">SELECT ConvexHull(geometry)
       ↪ from IRL_adm1 WHERE NAME_1 = 'Clare'</SrcSQL>
        <TargetSRS>EPSG:29900</TargetSRS>
    </OGRVRTLayer>

<!-- FINAL XML TAG -->
</OGRVRTDataSource>
```

If we wanted to generate the country boundary from the set of county boundaries, we could add a new VRT layer and use the `ST_Union` function from SQLite. This would essential dissolve all boundaries to generate a vector layer consisting of only one feature. The definition of this layer is outlined below.

```
 ...
<OGRVRTLayer name="IRL_Counties_diss">
        <SrcDataSource>./IRL_adm1.shp</SrcDataSource>
        <SrcLayer>IRL_adm1</SrcLayer>
        <SrcSQL dialect="sqlite">SELECT ST_Union(geometry) from
       ↪ IRL_adm1</SrcSQL>
        <TargetSRS>EPSG:29900</TargetSRS>
</OGRVRTLayer>

<!-- FINAL XML TAG -->
</OGRVRTDataSource>
```

[6] http://www.gadm.org/country

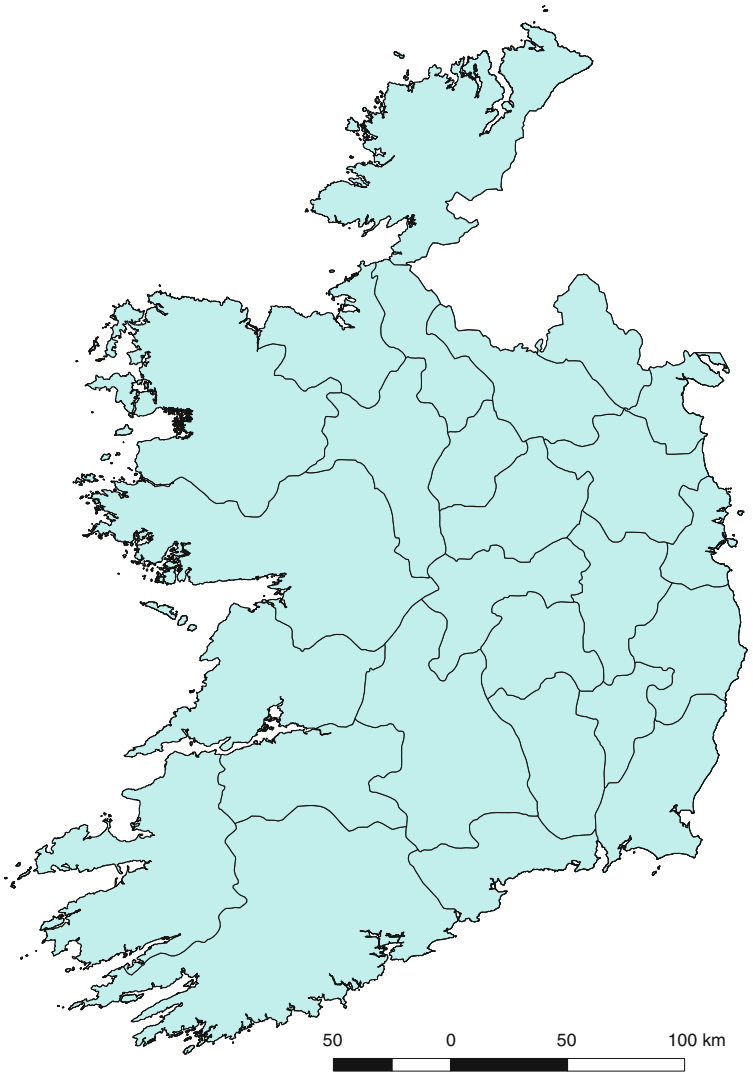

Fig. 2.3 Irish county boundaries from VRT layer

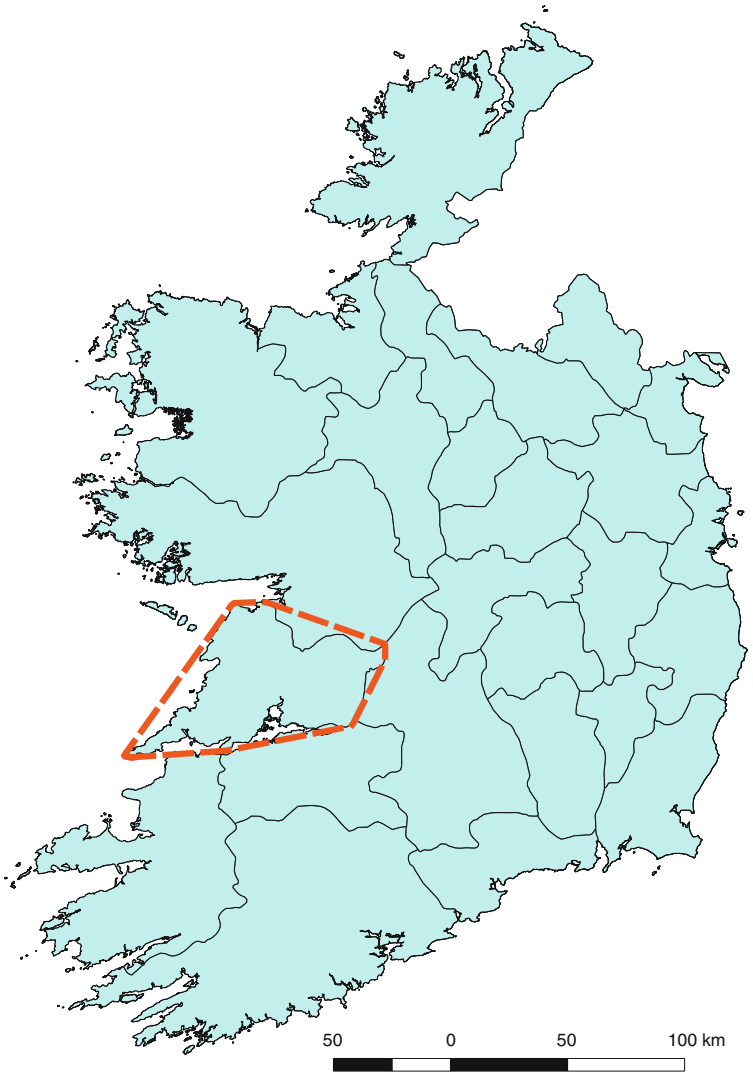

Fig. 2.4 Convex hull of the spatial extent of county Clare (West of Ireland) from VRT layer

It is important to note that the output from this example (Fig. 2.5), we see that there are numerous gaps generated in the dissolved dataset due to the lack of topology.

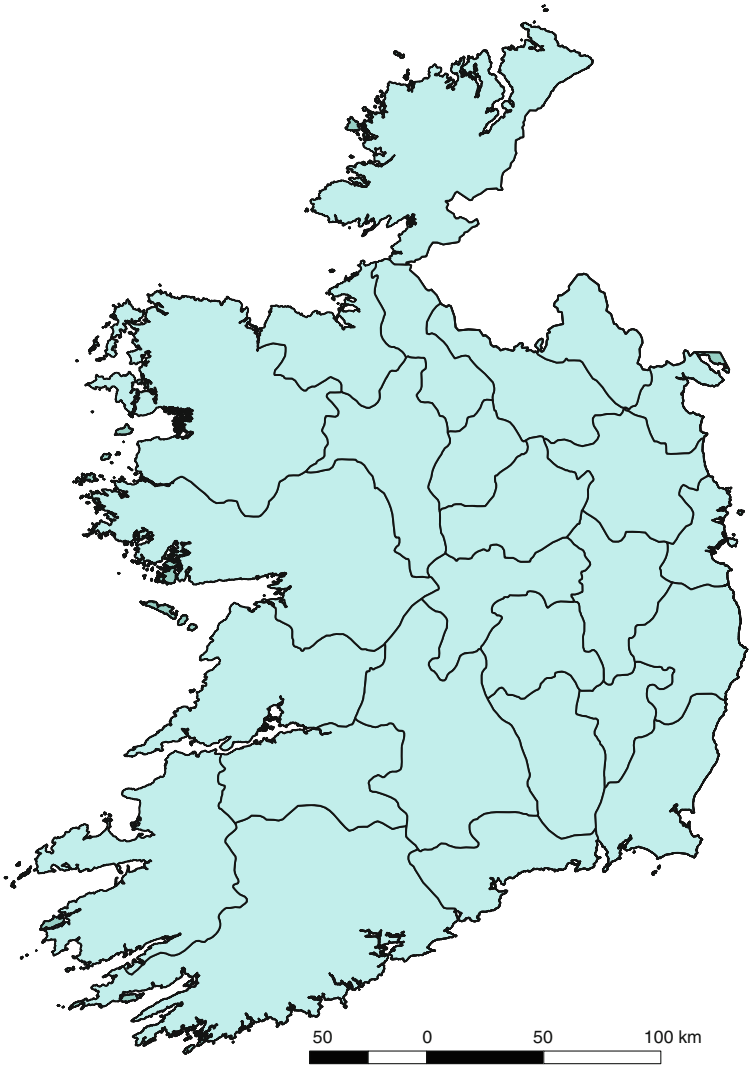

Fig. 2.5 Country boundary generated by dissolving county boundaries as a VRT layer

2.7 Spatial Databases

So far we have discussed binary vector formats that are widely used by geospatial analysts, but we have only briefly mentioned some of the more advanced formats that are used to store geospatial data. Relational database management systems (RDBMS) have traditionally been used for large-scale GIS projects, but with the wide availability of RDBMSs their use is becoming more common. On the one hand they provide

support to existing GIS formats to store the attributes in lieu of the legacy dBase file format (DBF files), but they also provide the facility to simultaneously store both the geometries and attributes. There are a wide number of systems, such as PostGIS, Spatialite, MSSQL and ORACLE Spatial that are considered to be spatially enabled extensions of their underlying databases. It is beyond the scope of this book to treat all of the different types of spatial databases. For the purposes of the subsequent examples and case studies, we focus primarily on Spatialite, but we provide some brief explanations of how to use OGR to access PostGIS tables. Nonetheless, it is important to note that many of the queries that are presented for Spatialite can also be adapted to other spatial databases.

2.7.1 PostGIS

PostGIS is an extension to the PostgreSQL database that can store vector data and also provides a series of spatial functions that can be used on the database. In most cases, the GDAL binaries are pre-compiled with PostGIS support, but if not it is possible to compile GDAL by passing the correct paths to PostgreSQL (`pg_config`). Accessing PostGIS tables is done by specifying a database connection string, e.g. `PG:"dbname='databasename' host='addr' port='5432' user='x' password='y'"`. This connection string is used with OGR commands. For instance, if we would like to return summary information regarding a PostGIS database, including tables and schema it can be done as follows:

```
ogrinfo PG:"dbname='databasename' host='addr' port='5432'
       ↪ user='x' password='y'"
```

2.7.2 Spatialite

Spatialite is a library that provides spatial SQL functionality to the SQLite database.[7] It can store vector geometries and allows for vector operations, including queries using spatial functions to be performed within the files in the database. The spatial functions follow the specifications defined by the Open Geospatial Consortium, specifically the OGC Simple Feature Specification (OGC-SFS).[8] The underlying SQLite database supports the SQL92 standard. For these reasons, it shares many similarities with PostGIS and Oracle Spatial, but differs in that the database, SQL

[7] http://www.sqlite.org

[8] http://www.opengeospatial.org/standards/sfs

engine and data objects are all embedded within one file and are not based on a client-server architecture. This makes its use more straightforward as it does not require the installation and adminstration overhead that is commonly required for some PostGIS and other server based spatial databases. However, it also means that Spatialite tends to be more suited to environments where it will be used for single user access rather than in distributed, multi-user environments. It is also noted[9] that Spatialite is optimized for use on physical drives of local filesystems rather than in networked environments that rely on NFS and Samba as it does not support file locking very well. Nonetheless, for quick tasks and everyday use, the format offers significant advantages over legacy, multi-file formats such as ESRI Shapefiles with the associated DBF files. Spatialite is also interoperable and can be used on Linux, Unix, MS Windows and Mac operating systems. It implements the OGC Simple Features Access Standard and provides SQL functions that can be used with ST_Geometry types supported by the GEOS library[10]; R-Tree Spatial Indexing is available in versions of SQLite 3.6 and above. Many of the functions in Spatialite are based on the Minimum Bounding Rectangle (MBR). It is worth noting that an equivalent "Spatialite" for raster data is librasterlite,[11] which is also built on the SQLite database.

The installation of Spatialite is straightforward and is described in the Appendix B.5. In addition to the Spatialite application, binaries exist for a `spatialitegui`, which is the Spatialite graphical user interface that can be used to view data and access much of the Spatialite functionality through menus and buttons.

Spatialite also provides its own command line interface. The following would be automatically printed within your terminal when you issue the command:

```
spatialite myfile.sqlite

SpatiaLite version ..: 2.4.0     Supported Extensions:
        - 'VirtualShape'         [direct Shapefile access]
        - 'VirtualDbf'           [direct DBF access]
        - 'VirtualText'          [direct CSV/TXT access]
        - 'VirtualNetwork'       [Dijkstra shortest path]
        - 'RTree'                [Spatial Index - R*Tree]
        - 'MbrCache'             [Spatial Index - MBR cache]
        - 'VirtualFDO'           [FDO-OGR interoperability]
        - 'SpatiaLite'           [Spatial SQL - OGC]
PROJ.4 version ......: Rel. 4.7.1, 23 September 2009
GEOS version ........: 3.2.0-CAPI-1.6.0
SQLite version ......: 3.7.13

================ FDO-OGR Spatial Metadata detected
       ↪ ===============
        created VirtualFDO table 'fdo_subset_ile_de_france'
        created VirtualFDO table 'fdo_forest_select'
Accessing these fdo_XX tables you can take full advantage of
FDO-OGR auto-wrapping facility
```

[9] http://www.gdal.org/ogr/drv_sqlite.html

[10] http://geos.osgeo.org

[11] http://www.gaia-gis.it/fossil/librasterlite/index

```
This allows you to access any specific FDO-OGR Geometry as if it
where native SpatiaLite ones in a completely transparent way
============================================================

Enter ".help" for instructions
spatialite>
```

To list the tables (including geometric tables), it suffices to type `.tables` (n.b. you must start with the period):

```
spatialite> .tables

fdo_forest_select          forest_select
       ↪ spatial_ref_sys
fdo_subset_ile_de_france   geometry_columns
       ↪ subset_ile_de_france
```

To get a full list of columns in a table and their respective data types, you can use the following command:

```
PRAGMA table_info(subset_ile_de_france)

0|OGC_FID|INTEGER|0||1
1|GEOMETRY|BLOB|0||0
2|osm_id|VARCHAR(80)|0||0
3|osm_way_id|VARCHAR(80)|0||0
4|name|VARCHAR(80)|0||0
5|type|VARCHAR(80)|0||0
6|aeroway|VARCHAR(80)|0||0
7|amenity|VARCHAR(80)|0||0
8|admin_leve|VARCHAR(80)|0||0
```

It is beyond the scope of this book to provide a detailed explanation of SQLite and Spatialite, as a thorough discussion of the topic would merit a book in itself. However, we feel that it is worthwhile to briefly outline some of the main functionalities that are provided by Spatialite, which can be carried out directly using the Spatialite command line. In addition to the standard SQL queries that can be performed on the database based on the SQL92 Standard, from a geospatial perspective the real advantage of Spatialite is the ability to use spatial functions. Some brief examples are presented next, with a more in-depth look at these functions in the case study (Chap. 16). For example, if we want to initially return some basic information regarding the layers, we can issue the following command:

```
spatialite> SELECT * FROM geometry_columns;
f_tale_name | f_geometry_column | geometry_type | coor_dimension
       ↪  | srid | geometry_format
subset_ile_de_france|GEOMETRY|3|2|4326|WKB
forest_select|GEOMETRY|3|2|4326|WKB
```

Indexing database tables is often recommended as it speeds up SQL queries and other database operations. We can use Spatialite to create an index on the table `subset_ile_de_france` by issuing the next command followed by the vacuum on the table:

```
spatialite> create unique index spatidx on
       ↪ subset_ile_de_france(osm_id);
spatialite> vacuum subset_ile_de_france;
```

2.7.3 ogr2ogr with Spatialite

This next section will demonstrate how you can create a Spatialite database using `ogr2ogr` and to add multiple spatial vector datasets to it. We begin by using `ogr2ogr` to create the Spatialite database from the OSM ESRI Shapefile:

```
ogr2ogr -f "SQLite" myfile.sqlite subset_ile-de-france.shp
```

A quick consistency check on the database using `ogrinfo` returns basic information about the newly created Spatialite database, including the geometry type, the number of features within the layer and its spatial extent.

```
ogrinfo -so -al myfile.sqlite

INFO: Open of `myfile.sqlite'
      using driver `SQLite' successful.

Layer name: subset_ile_de_france
Geometry: Polygon
Feature Count: 27012
Extent: (1.700000, 48.272681) - (2.130000, 48.630000)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
FID Column = OGC_FID
Geometry Column = GEOMETRY
osm_id: String (0.0)
osm_way_id: String (0.0)
name: String (0.0)
type: String (0.0)
aeroway: String (0.0)
amenity: String (0.0)
admin_leve: String (0.0)
barrier: String (0.0)
```

```
boundary: String (0.0)
building: String (0.0)
craft: String (0.0)
geological: String (0.0)
historic: String (0.0)
land_area: String (0.0)
landuse: String (0.0)
leisure: String (0.0)
man_made: String (0.0)
military: String (0.0)
natural: String (0.0)
office: String (0.0)
place: String (0.0)
shop: String (0.0)
sport: String (0.0)
tourism: String (0.0)
other_tags: String (0.0)
```

For instance, we can easily add a second dataset to Spatialite database by issuing the following command, which updates our SQLite database with the features from a Shapefile called `France_admin.shp`. Note that the option `-nln` defines the new layer name in the database:

```
ogr2ogr -f  QLite  -update  -append myfile.sqlite
      ↪ France_admin.shp -update -append -nln france_admin
```

One of the great advantages of Spatialite is that multiple layers and geometry types can be stored in the one relational database. Furthermore, the fact that an array of spatial functions can be used is also very efficient. For example, it is possible to calculate the convex hull of the newly added layer, `france_admin`:

```
ogrinfo -dialect SQLite -sql "select ConvexHull(Geometry) from
      ↪ france_admin" myfile.sqlite
```

```
ogrinfo -ro myfile.sqlite

INFO: Open of 'myfile.sqlite'
      using driver 'SQLite' successful.
1: subset_ile_de_france (Polygon)
2: forest_select (Polygon)
3: france_admin (Polygon)
```

http://www.springer.com/978-3-319-01823-2

Open Source Geospatial Tools
Applications in Earth Observation
McInerney, D.; Kempeneers, P.
2015, XXVII, 358 p. 96 illus., 50 illus. in color., Hardcover
ISBN: 978-3-319-01823-2