UCL

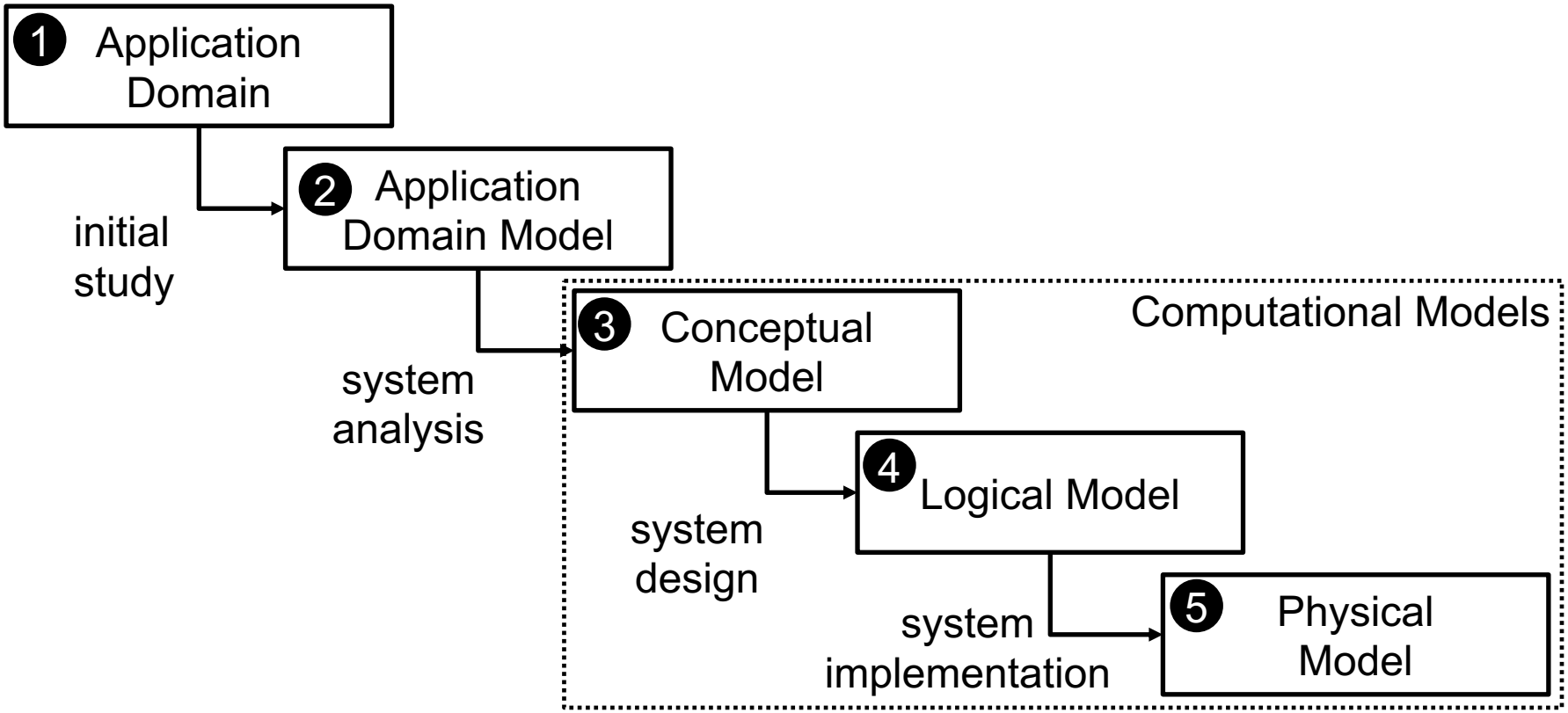
CEGE0096: Geospatial Programming

Lecture 7: Vector and Raster Representations

November 18, 2019

Aldo Lipani

The Modelling Process – Waterfall Model

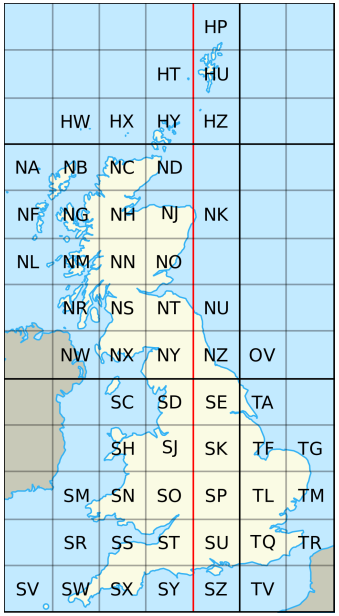


COORDINATE REFERENCE SYSTEMS

Coordinate Reference Systems

On this module we will predominantly use these two Coordinate Reference Systems (CRS):

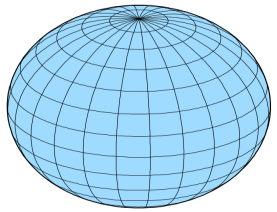
- World Geodesic System (WSG)
- British National Grid (BGN)



World Geodesic System

This refers to the last one made in 1984 (EPSG:4326).

This coordinate system is located at the earth's center of mass;
it models earth as an ellipsoid.



This is a geodetic coordinate system based upon the **WGS 84 datum**:

- **Latitude**: The angular distance of a place north or south of the equator;
- **Longitude**: The angular distance of a place east or west of the Greenwich meridian.

These are often provided in sexagesimal notation (in degrees, minutes, seconds):

```
051° 31' 33.846 N
000°  7' 58.537 W
```

British National Grid

This is a local coordinate reference system used in Great Britain, known as OSGB36 (EPSG:27700).

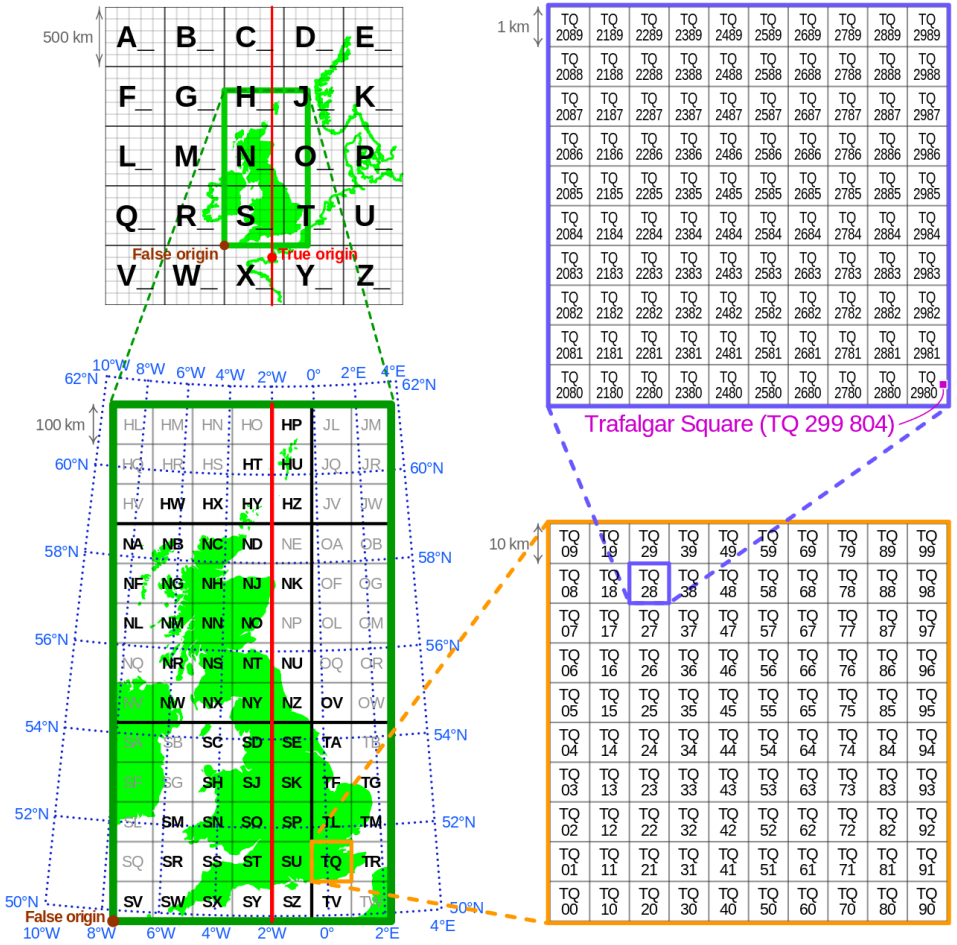
It uses the transverse Mercator Projection.

It is based upon the OSGB 1936 datum:

- Easting (Metres East of False Origin)
- Northing (Metres North of False Origin)

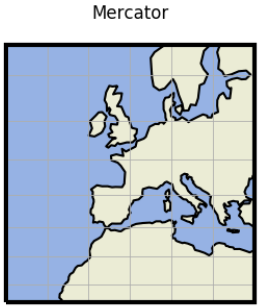
Coordinates are represented as integers:

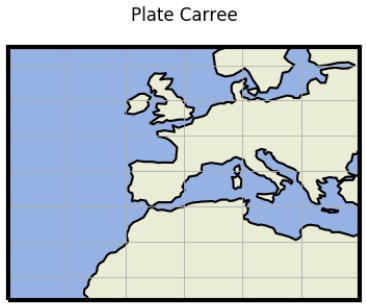
```
Easting: 529620
Northing: 182447
```

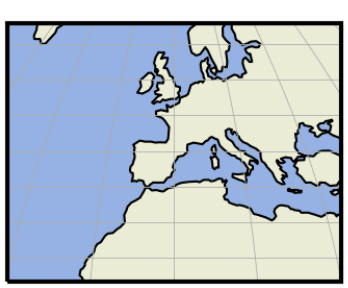


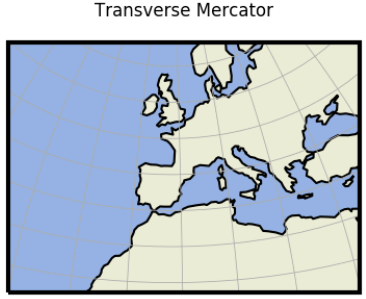
PROJECTIONS

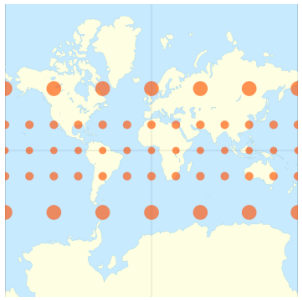
Projections

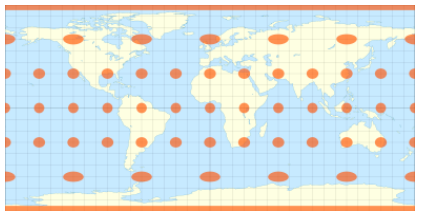


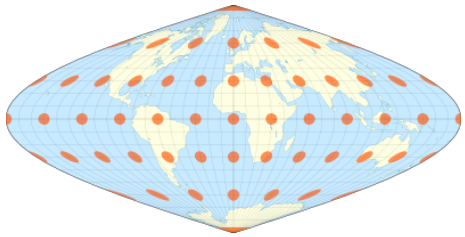


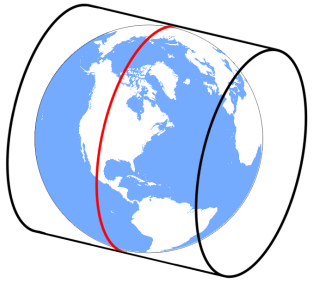












PYPROJ

Pyproj

This is a Python package for cartographic projections and coordinate transformations.

This is usually imported as:

```
import pyproj
```

It uses PROJ.4 notation to define projections. However, unless we are defining our own projections, we can use EPSG (http://epsg.io) strings for standard projections.

```
wgs84 = pyproj.Proj("+init=EPSG:4326")    # LatLon with WGS84 datum used by
                                          # GPS units and Google Earth
osgb36 = pyproj.Proj("+init=EPSG:27700") # UK Ordnance Survey, 1936 datum
```

To transform coordinates from WSG to BNG we can simply use the method transform:

```
pyproj.transform(wgs84, osgb36, latitude, longitude)
```

VECTOR REPRESENTATION

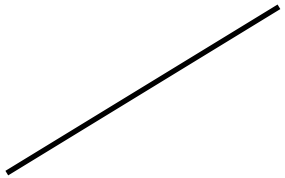
Vector Representation

A vector is a line segment (edge, wire) defined by its end points (vertices).

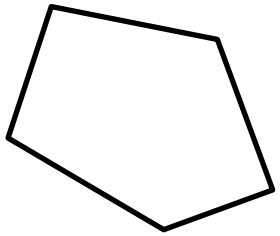
Line segments may be curved.

A 0D geometric entity

A 1D geometric entity.



A 2D geometric entity.

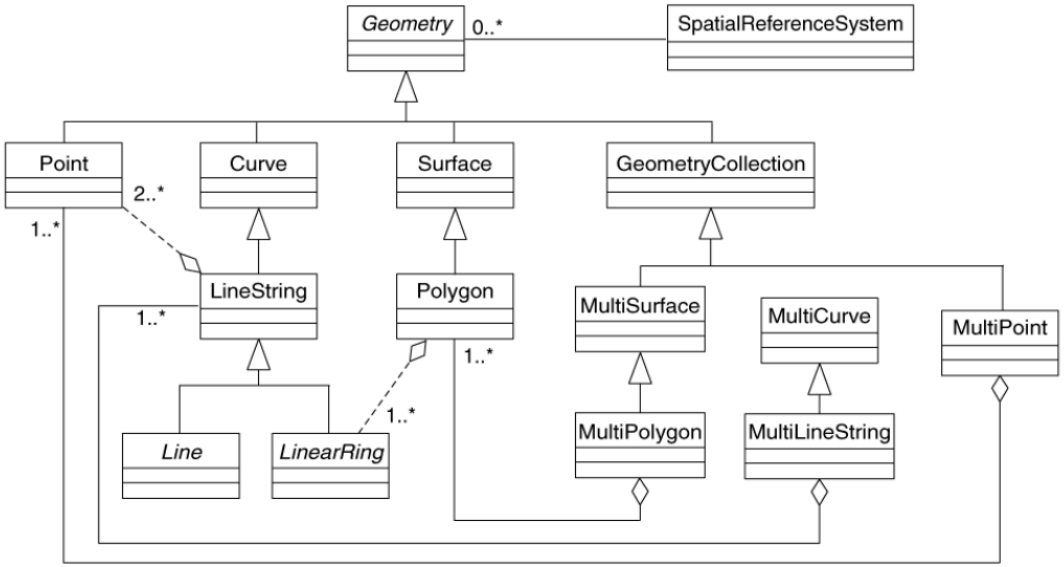


ISO 19125 Simple Feature Access

This standard has been developed by the Open Geospatial Consortium

Standardized by the International Standards Organization (ISO).

A standard class relationship among, points, line strings, and polygons.



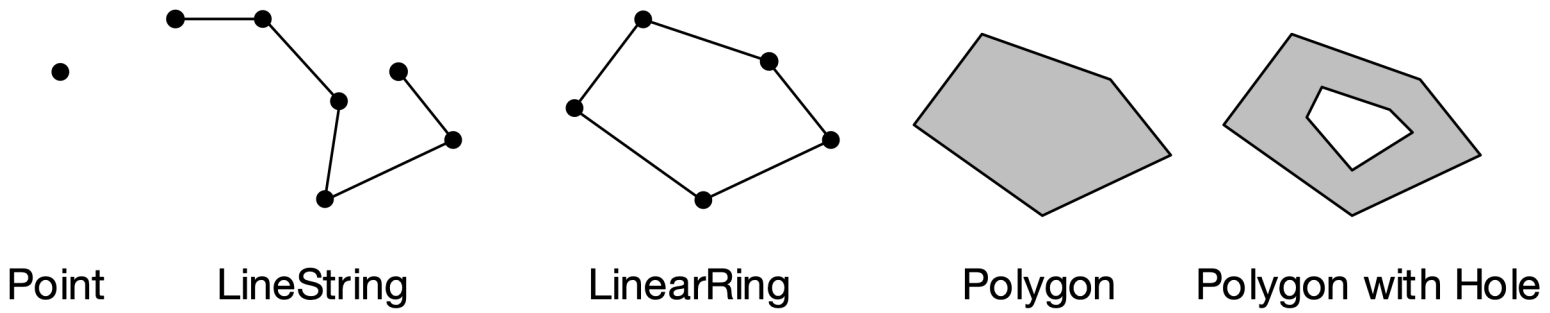
ISO 19125 Simple Feature Access (II)

Geometry is a parent class from which all other classes are inherited.

LineString is defined as an array of vertices (points).

LinearRing is a specialized class of LineString which last vertex is the same as the first one.

Polygon is a class defined by an external LinearRing and a list of internal LinearRings.



SHAPELY

Points

We can represent points in Python using the shapely geometry package.

```
from shapely.geometry import Point
```

We can create a point by passing the coordinates x and y:

```
pt1 = Point(0, 0)
```

Operations

A fundamental geospatial operation on point objects is to compute the distance to another point.

Using shapely we can compute the Euclidean (or Cartesian) distance between two points:

```
pt2 = Point(1, 1)

pt1.distance(pt2)
```

Operations (II)

Every geometry object in Shapely defines the following attributes.

To get the length of a geometric object:

```
pt1.length
```

To get the area:

```
pt1.area
```

To get the minimum bounding rectangle:

```
pt1.bounds
```

Operations (III)

To return the geometry type:

```
pt1.geom_type
```

To compute the distance from the two furthest points of the objects:

```
pt1.hausdorff_distance
```

To get the coordinates as an array:

```
pt1.xy
```

Line Strings

Using the shapely geometry, we can use the LineString class to represent a series of segments all connected to each other.

```
from shapely.geometry import LineString
```

To create a segment we can pass a list of two points as coordinates:

```
LineString([(0, 0), (1, 1)])
```

Or, two points:

```
LineString([pt1, pt2])
```

Operations

Since the LineString class inherits from the Geometry class, it also has the same operators presented for the Point class.

However, the logic of these methods changes accordingly. For example, `distance` computes the minimum distance between the series of segments defining the LineString and a point:

```
line.distance(pt1)
```

Or, the attribute xy returns the series of coordinates as an array:

```
line.xy
```

LineString (II)

We can build more complicated LineString objects by passing more than two points. This will create multiple segments sequentially connected.

```
LineString([(0, 0), (1, 1), (1, 0), (0.6, 0.4)])
```

Polygon

Shapely also enables the creation of polygons.

```
from shapely.geometry import Polygon

Polygon([(0, 0), (1,1), (1,0)])
```

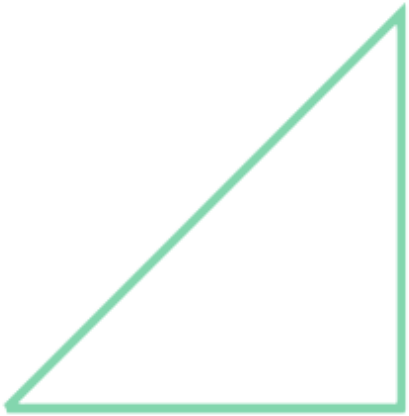
Operations

Like Line Strings, all attributes presented so far are still available for Polygons.

However, polygons implement a new attribute called exterior.

This returns the polygon's perimeter as a LineString:

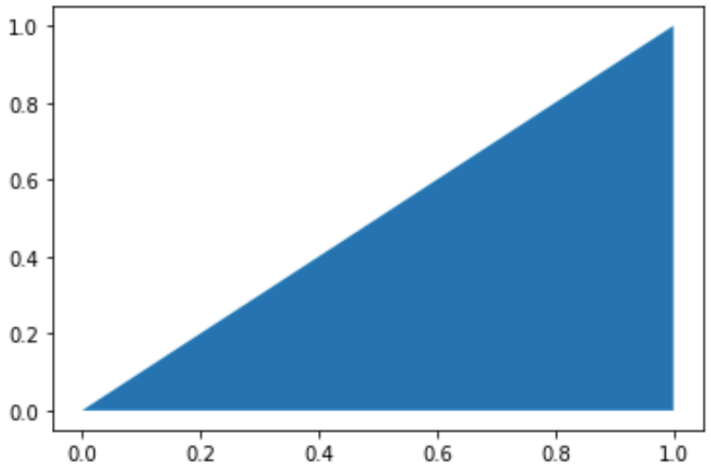
```
polygon.exterior
```



Plotting a Polygon in Matplotlib

We can easily plot a polygon using Matplotlib like:

```
x, y = polygon.exterior.xy
plt.fill(x, y);
```



Topological Relationships

Topological relationships describe qualitative properties that characterize the relative position of spatial objects: disjoint, meet, overlap, and inside are few examples.

We can use methods implemented by shapely classes to query the topological relationship between shapely objects.

To check if a point touches the polygon:

```
polygon.touches(pt1)
```

To check if a point is contained in a polygon:

```
polygon.contains(pt1)
```

Geometric Operations

A buffering operation returns a Polygon around an object that is offset by a given distance:

```
pt1.buffer(1)
```

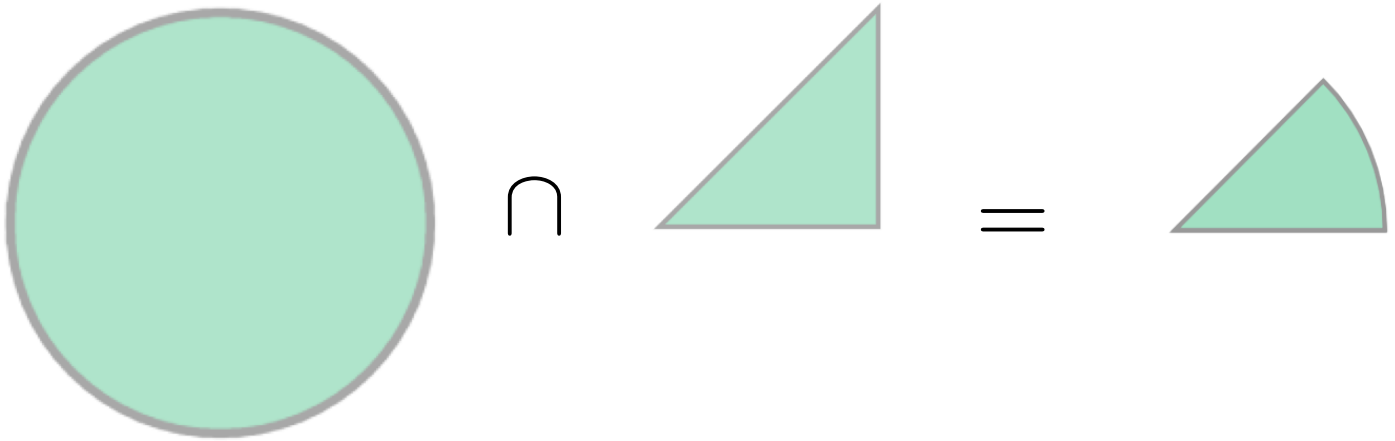
```
line.buffer(1)
```

```
polygon.buffer(1)
```

Geometric Operations (II)

We can intersect multiple shapes together to create a new shape:

```
pt1.buffer(1).intersection(polygon)
```



GEOPANDAS

GeoPandas

It is an open source project to make working with geospatial data in Python easier.

```
import geopandas as gpd
```

GeoPandas extends the datatypes used by pandas to allow spatial operations on geometric types.

Geometric operations are performed by Shapely. And, plotting by matplotlib.

GeoPandas implements two main data structures **GeoSeries** and **GeoDataFrame**. These are subclasses of pandas Series and DataFrame.

GeoSeries

A GeoSeries is a vector where each entry in the vector is a set of shapes, corresponding to one observation. An entry may consist of only one shape.

GeoPandas supports shapely objects: Point, Lines, and Polygons.

Note that like for pandas' Series, all entries of a GeoSeries need to be of the same type.

Attributes

We can compute the area of each entry:

```
series.area
```

We can compute their bounds:

```
series.bounds
```

Its total bounds:

```
series.total_bounds
```

To get the shapely type:

series.geom_type

Methods

We can compute the distance to another object:

```
series.distance(pt1)
```

Returns a GeoSeries of centroids:

```
series.centroids(pt1)
```

Return a GeoSeries of points that are within each geometry:

```
series.representantive_points()
```

To change coordinate system:

```
series.to_crs()
```

To plot a series:

```
series.plot()
```

GeoDataFrame

A GeoDataFrame is a tabular data structure that contains GeoSeries.

The most important property of a GeoDataFrame is that it always has one GeoSeries column that holds a special status. This GeoSeries is referred to as **geometry**.

When a spatial method is applied to a GeoDataFrame or a special attribute like area is called, this commands will always act on the **geometry** column.

We can have multiple series in a GeoDataFrame acting as geometry but only one can be considered at all time.

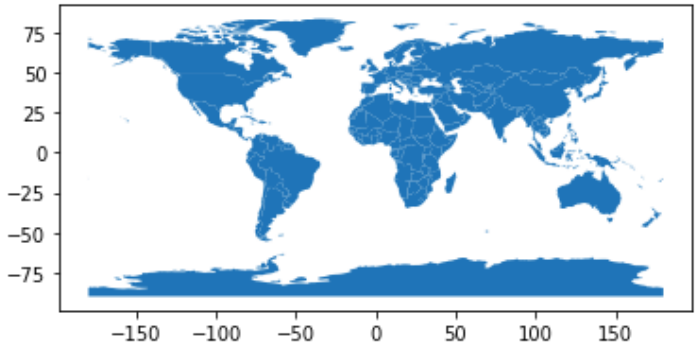
GeoDataFrame (II)

```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
```

```
world.head()
```

	pop_est	continent	name	iso_a3	gdp_md_est	geometry
0	920938	Oceania	Fiji	FJI	8374.0	MULTIPOLYGON (((180.00000 -16.06713, 180.00000...
1	53950935	Africa	Tanzania	TZA	150600.0	POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
2	603253	Africa	W. Sahara	ESH	906.5	POLYGON ((-8.66559 27.65643, -8.66512 27.58948...
3	35623680	North America	Canada	CAN	1674000.0	MULTIPOLYGON (((-122.84000 49.00000, -122.9742...
4	326625791	North America	United States of America	USA	18560000.0	MULTIPOLYGON (((-122.84000 49.00000, -120.0000...

```
world.plot()
```




```
world.geometry.name
```

FILE FORMATS FOR VECTOR REPRESENTATION

GeoJSON

It is an open standard format designed for representing simple geographical features.

The features include: points, lines, polygons, and combinations of them.

We can associate to each item attributes.

Coordinate reference system is the World Geodetic System 1984 (WGS 84) datum.

```
{
   "type": "Feature",
   "geometry": {
       "type": "Point",
       "coordinates": [125.6, 10.1]
    },
   "properties": {
       "name": "Dinagat Islands"
    }
}
```

Documentation: https://geojson.org/

Shapefile

It is a geospatial vector data format for GIS software.

It is developed and regulated by the Environmental Systems Research Institute (Esri).

It describes points, lines and polygons.

We can associate attributes to each item.

This is a multi-file format and it requires at least the following 3 files:

- .shp the shape file containing the feature geometry itself
- .shx the shape index containing a positional index for quicker access
- .dbf the attribute file, which defines columnar attributes for each shape

The files are stored in binary (not readable by a standard text editor).

Shapefile (II)

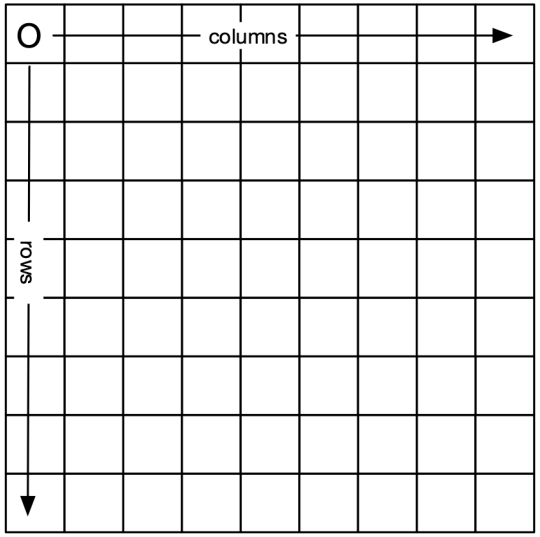
Limitations of this file format are:

- Poor support for Unicode field names or field storage;
- Maximum length of field names is 10 characters;
- Maximum number of fields is 255;
- Supported field types are:
  - floating point (13 characters);
  - integer (4 or 9 characters);
  - date (no time storage; 8 characters), and;
  - text (maximum 254 characters).
- Floating point numbers may contain rounding errors since they are stored as text.

RASTER REPRESENTATION

Raster Representation

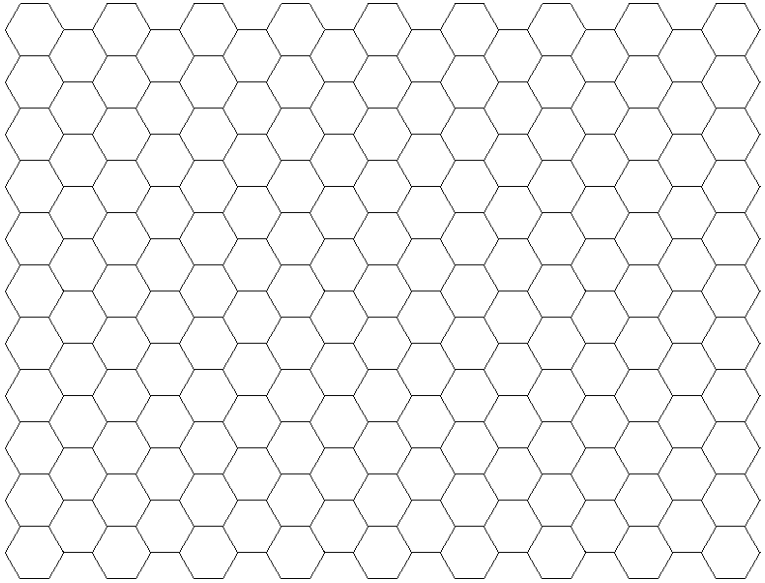
In the raster model a surface is broken down into a continues regular arrangement of discrete cells known as pixels.



Pixel in Area

Raster Representation (II)

The regular pattern could either be a grid of equilateral triangles or hexagons:

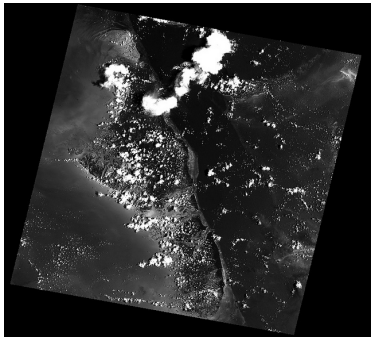


In practice, 99% of the times rasters are modelled as grid of squares.

Practical Applications

The Raster model is widely used to represent geospatial information.

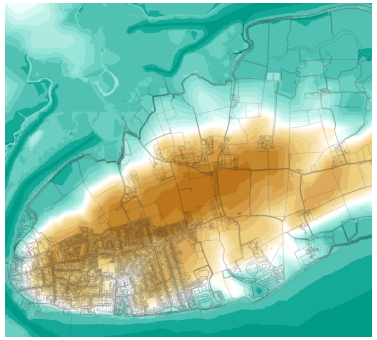
Remote Observation, like satellite imagery:



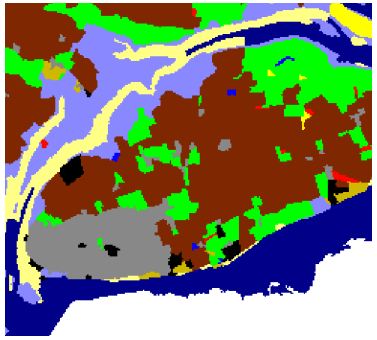
Processed Information, like aerial imagery:

Practical Applications (II)

Modelling Continuous Fields,
like digital elevation models:

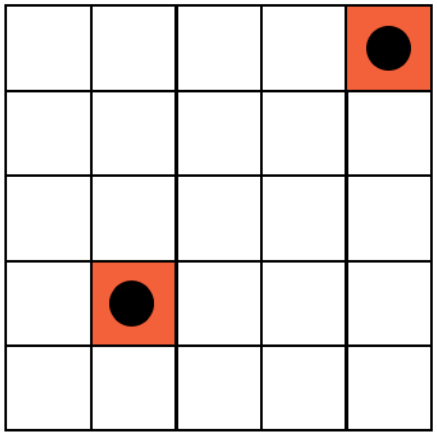


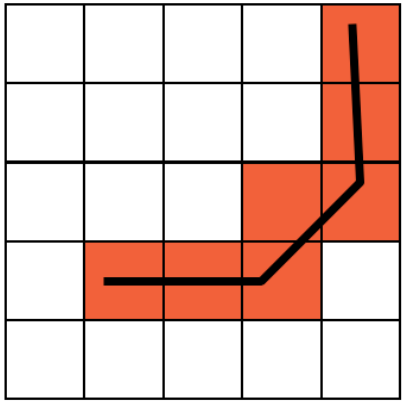
Land Classification data

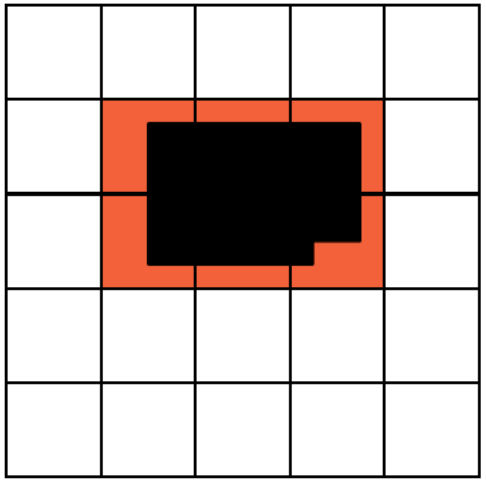


Modeling Discrete Features

Point, Line and Area Features are rasterized and represented as pixels in order to perform overlay operations.







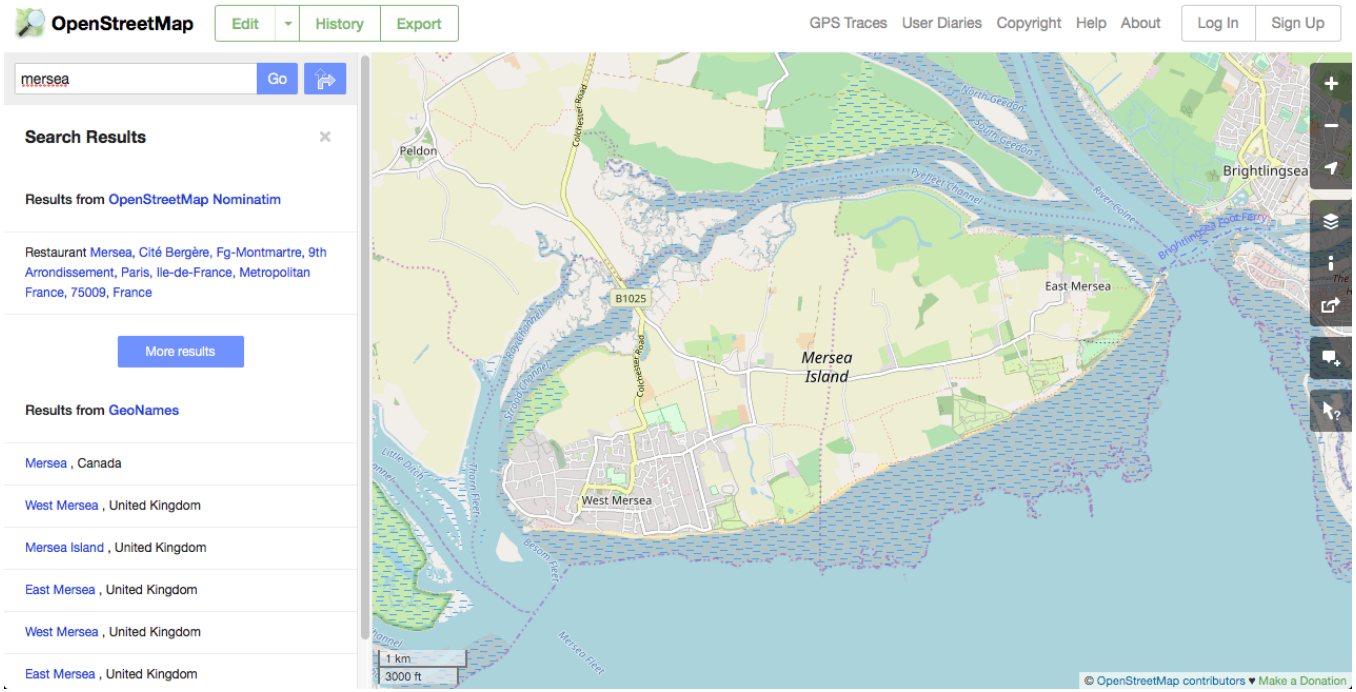
Cartographic Imagery

The Cartographer is an expert at communicating geospatial information to the map user.

Features are manually styled and labelled to provide maximum readability.

Map Tiles

Servers transform vector information and transmit rasterized map tiles to client devices:



Multi Criteria Decision Analysis

This analysis is used for example for site selection.

We first split a large area into discretized cells.

Each cell is given a value that is calculated according to the spatial relationship of that cell to a particular feature.

Then, we weight the value of each cell in accordance with the decision making criteria.

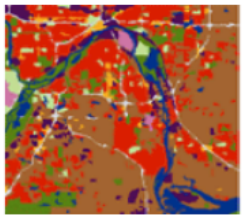
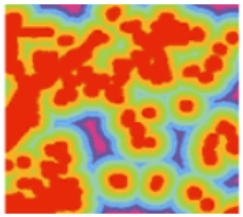
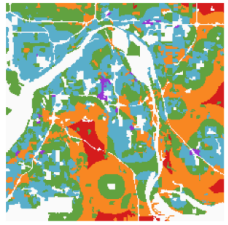
Sum the weighted values.

Map Algebra

Weighted overlay is an operation done with multiple raster layers.

Input 3 raster layers and determine weights:

$$w_1 l_1 + w_2 l_2 + w_3 l_3 = l_t$$

.5  + .15  + .35  = 

Land use input

Distance to parks input

Population density input

The output shows the most suitable areas in red.

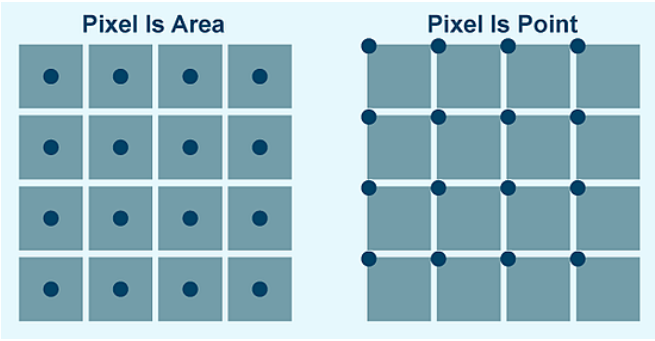
Pixels

Each pixel in a raster is assigned a value that represents a property of the feature being modelled.

These values could be:

- Boolean value
- An integer:
  - Index
  - Ranking
- A floating-point value:
  - Temperature
  - A color component of a color image

Area vs Point



The value of each pixel is either:

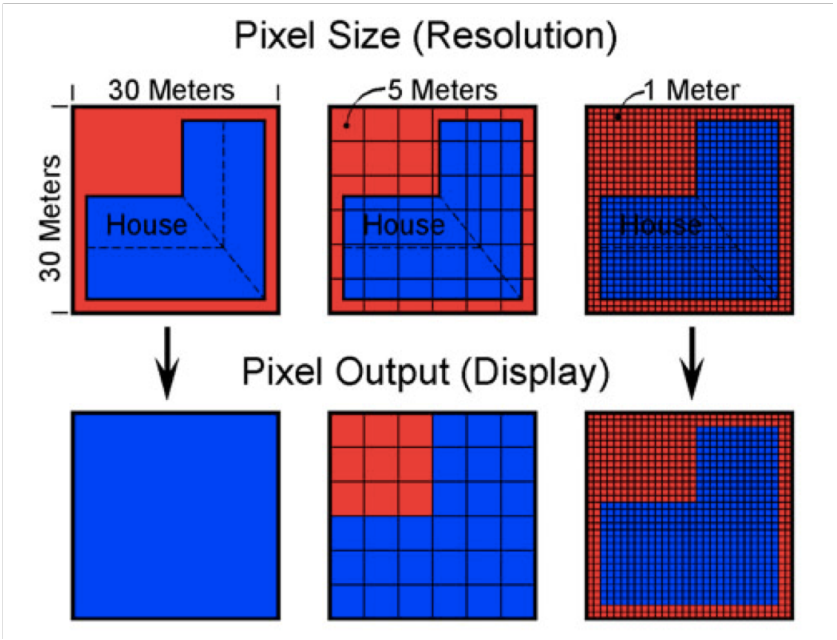
- Pixel-Is-Area
- Pixel-Is-Point

In **Pixel-Is-Area**, the value of the pixel represents a property of the feature that extends across the surface area of the pixel.

- It could be an average, a maximum, minimum.
- It could also be a discrete measurement taken at the center of the pixel.

In **Pixel-Is-Point**, the value of the pixel represent a property of the feature measured at the top-left corner of the pixel.

Resolution and Spatial Referencing



Given i, the row number:

$$y = y_0 + i \cdot res_v$$

Given j, the column number:

$$x = x_0 + j \cdot res_h$$

RASTERIO

Rasterio

Rasterio is a high-level package for reading and writing raster files.

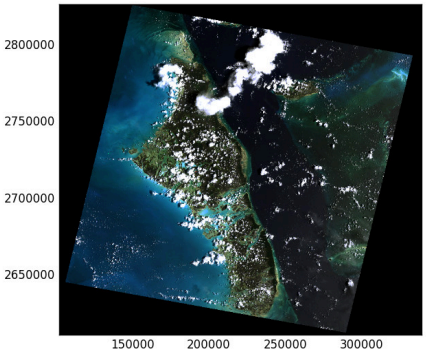
```
import rasterio
```

It provides a Python API based on NumPy arrays.

It can manipulate various raster formats like GeoTiff and ASCII Raster Format.

We can open a raster file like:

```
dataset = rasterio.open("example.tiff")
```



Attributes and Plotting

To get the bounds of the dataset:

```
dataset.bounds
```

To get the data type of the dataset:

```
dataset.dtypes
```

To get the number of bands (layers of rasters):

```
dataset.count
```

The coordinate reference system:

```
dataset.crs
```

To plot a raster:

```
import rasterio.plot
rasterio.plot.show(dataset)
```

FILE FORMATS FOR RASTER REPRESENTATION

GeoTiff

It is a public domain metadata standard which allows georeferencing information to be embedded within a Tiff file. A Tiff file (Tagged Image File Format) is a file format to store raster graphics images.

A Tiff file is composed of one or more bands. A band type is equal to a NumPy type: Byte, UInt16, Int16, etc.

GeoTiff contains the following metadata:

Geo Tiff Version
1.1.0
Gt Raster Type
Pixel Is Area
Gt Citation
OSGB 1936 / British National Grid
Geog Citation
OSGB 1936
Geog Angular Units
Angular Degree
Geog To Wgs84
446.448 -125.157 542.06 0.15 0.247 0.842 -20.489
Projected Cs Type
British National Grid
Proj Linear Units
Linear Meter

ASCII Raster Format

This is an ESRI raster format.

The parameters in the header part of the file must match correctly with the structure of the data file.

The basic structure of the ESRI ASCIII raster has the header information at the beginning of the file.

No CRS information is provided, it is usually stored in a separate metadata file.

```
NCOLS xxx
NROWS xxx
XLLCENTER xxx
YLLCENTER xxx
CELLSIZE xxx
NODATA_VALUE xxx
row 1
row 2
...
row n
```

FILE FORMATS FOR BOTH REPRESENTATIONS

GeoPackage

Defined by the Open Geospatial Consortium (OGS), published in 2014.

It is a n open, non-proprietary, platform independent and standard-based data format for geographic GIS.

It supports both vector and raster representations.

It is based on SQLite 3 database -- The structure of the files is complex.

This format was designed to be as lightweight as possible and be contained in one ready-to-use single file (.gpkg).

Limitation: it only supports one geometry column per table.

See you on Thursday for the Practical

Christopher Ingold Building G20

Do not forget to take your personal laptop with you.