Aalto University
School of Engineering

YYT-C3002
Application Programming in Engineering

GIS I

Anas Altartouri
Otaniemi 22.1.2019

Overview: GIS lectures & exercise

We will deal with GIS application development in two lectures.

Because of the versatility of GIS data models and formats, GIS software applications, and GIS application domains and analyses, we will discuss the development of GIS application *broadly*.

Overview: GIS lectures & exercise

Using software libraries for computation, developing application software and information systems, and working with databases are important topics not only in the field of Geoinformatics.

You will learn:

- To manipulate data with command-line programs in Bash (Unix shell)
- To geoprocess with free and open-source software libraries
- To work with spatial databases and SQL
- To create simple geospatial web services
- Basic architecture of an information system

We will also talk about

- Writing a program for a geoprocessing task and developing a QGIS plugin for it (a desktop GIS extension)

What is a GIS application?

Application software that allows creation, storage, manipulation, analyzing, management, and presenting geospatial data.

Examples: ArcGIS (proprietary), QuantumGIS (FOSS)

Web Map Application

- Often developed for a specific case (limited functionality in comparison to a desktop GIS)
- Distributed systems that serve spatial data/information and utilize remote resources

Source: http://docs.qgis.org/2.8/en/docs/gentle_gis_introduction/introducing_gis.html

Spatial operations and analyses (examples)

Geometry operations

- Intersection, Union, Difference, Buffer, Clipping, etc.
- Intersects, Disjoint, Contains, Touches, Within, etc.

Spatial prediction

- Regression; geographically weighted regression (GWR)
- Interpolation; Inverse Distance Weighted (IDW), Kriging

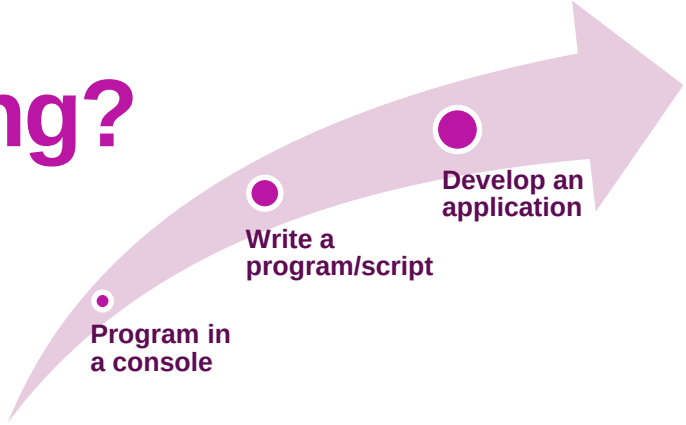
Map algebra

- Local
- Focal
- Zonal

Network analysis

- Routing (e.g., road networks, sewer networks)

What kind of programming?



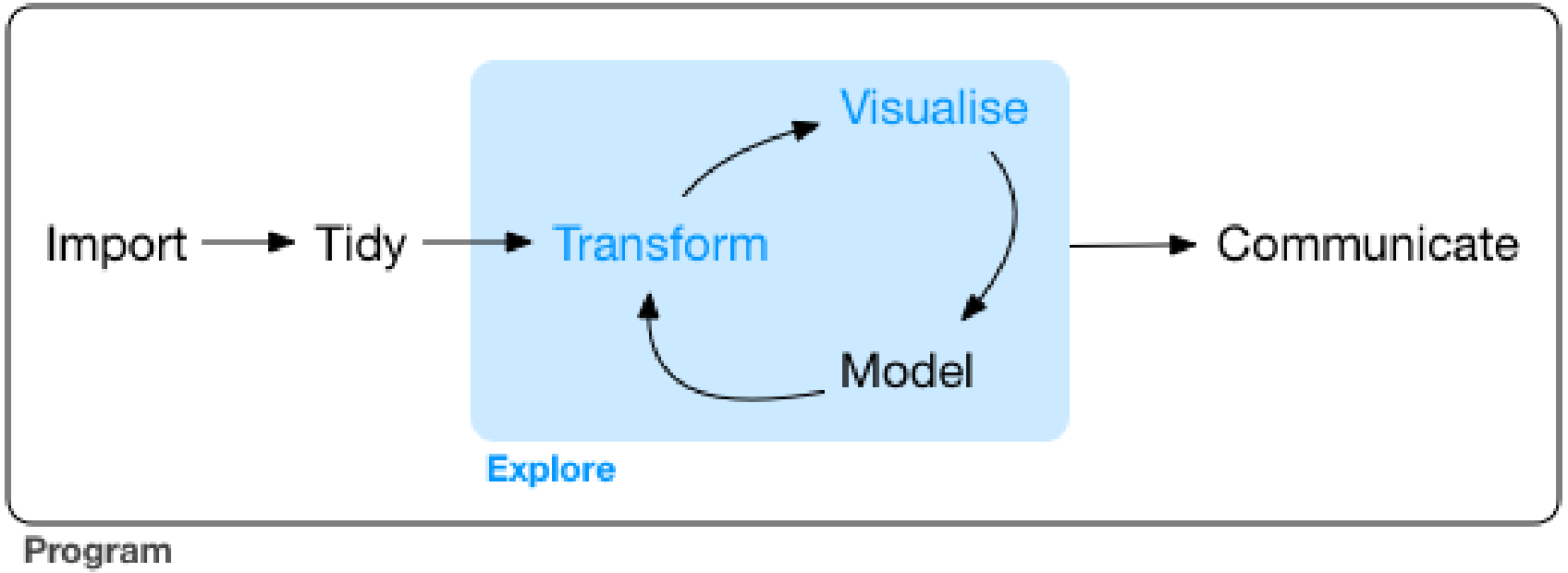


Image sources: Grolemund & Wickham (2017). Available from: http://r4ds.had.co.nz/explore-intro.html

GIS application development

Desktop

Extending a GIS application
- ArcGIS add-in
- QGIS plugin

Web-based

Web map application
- Database/spatial DB
- Server-side application, geospatial web services
- Client-side application

Geoprocessing
- Reading/writing libraries
- Vector data geoprocessing libraries
- Raster data geoprocessing libraries

Programming Languages for GIS

A long list! (see for example: https://www.e-education.psu.edu/geog583/node/67)

Python

R

JavaScript

SQL

C++

Java

https://www.codecademy.com/
http://www.learnpython.org/
http://learnpythonthehardway.org/book/
http://www.macwright.org/2012/10/31/gis-with-python-shapely-fiona.html

Geoprocessing with open source libraries

GEOS

Geometry Engine

Shapely

Python package for manipulation and analysis of planar geometric objects (GEOS bindings)

GDAL

Geospatial Data Abstraction Library (Python, Perl, and other bindings)

Fiona

GDAL Python binding

NumPy

Python extension that provides support for large, multi-dimensional arrays and matrices

PyGRASS

Python API for GRASS GIS (wide range of vector and raster analysis functions)

Exercise walkthrough

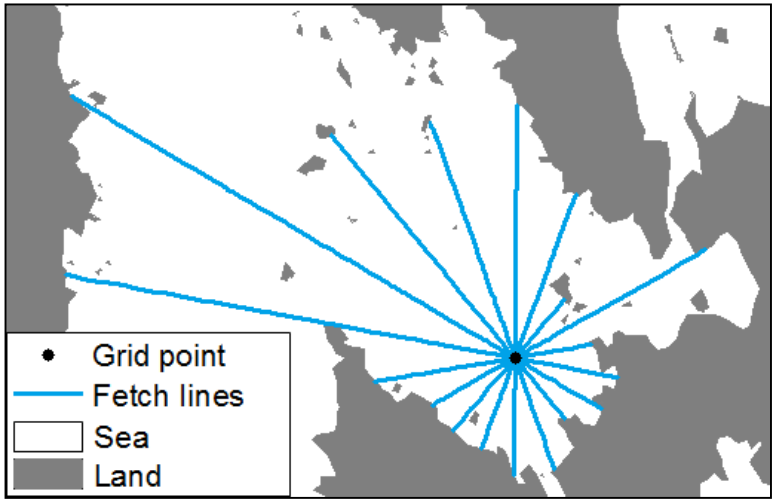
The exercise following the GIS lectures is divided in two parts.

Most of the tasks of the exercise deal with a specific case, but there are also ‘generic’ tasks that aim to introduce the databases, spatial databases, and SQL.

Geoprocessing

Case

Calculate openness of seawater at coastal areas and inlets

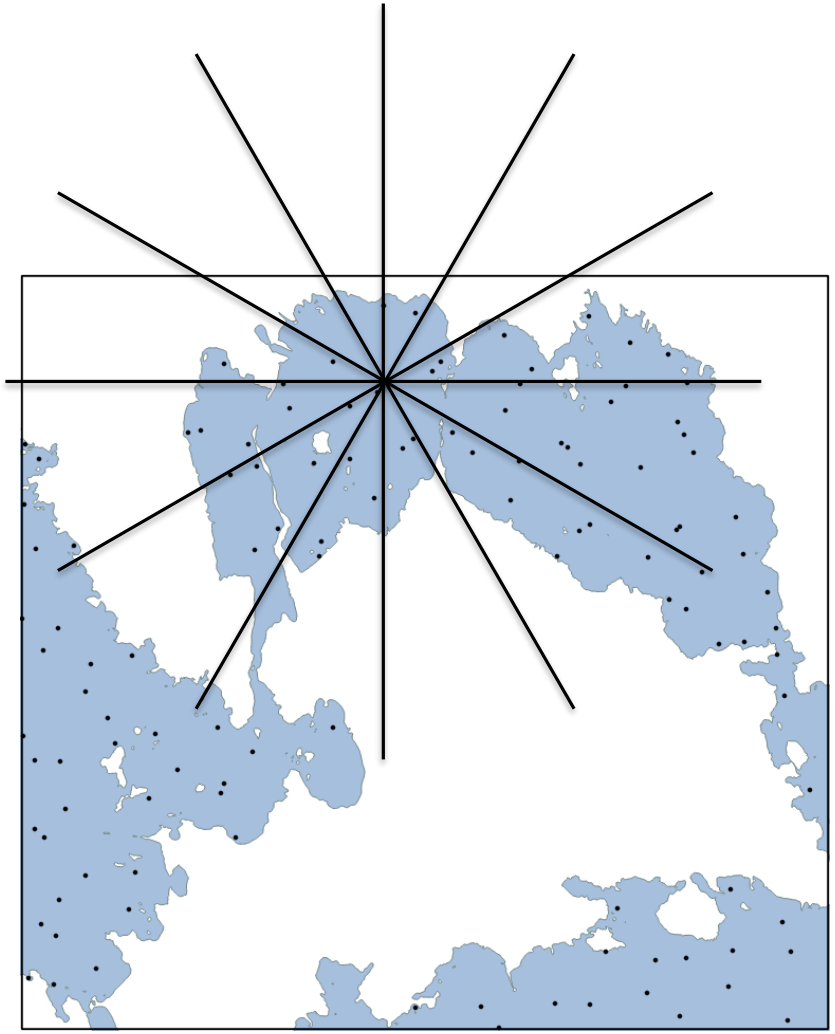


Geoprocessing

Case

Datasets we need:

1. The extent of the study area (a polygon vector layer)
2. Sea areas (a polygon vector layer)
3. A grid of points distributed across the sea in the study area (a point vector layer)
4. Lines radiating from each point in the grid (a line vector layer)



Geoprocessing

Procedure

- Download the topographic data
- Create polygons of the sea area
- Create a grid of points
- Create radiating lines

Prepare the data
- Command-line tools
- Software library

- Clip and clean the radiating lines
- Aggregate the fetch lengths around each grid point
- Interpolate an openness grid

Do the computations
- Spatial database

Demo…

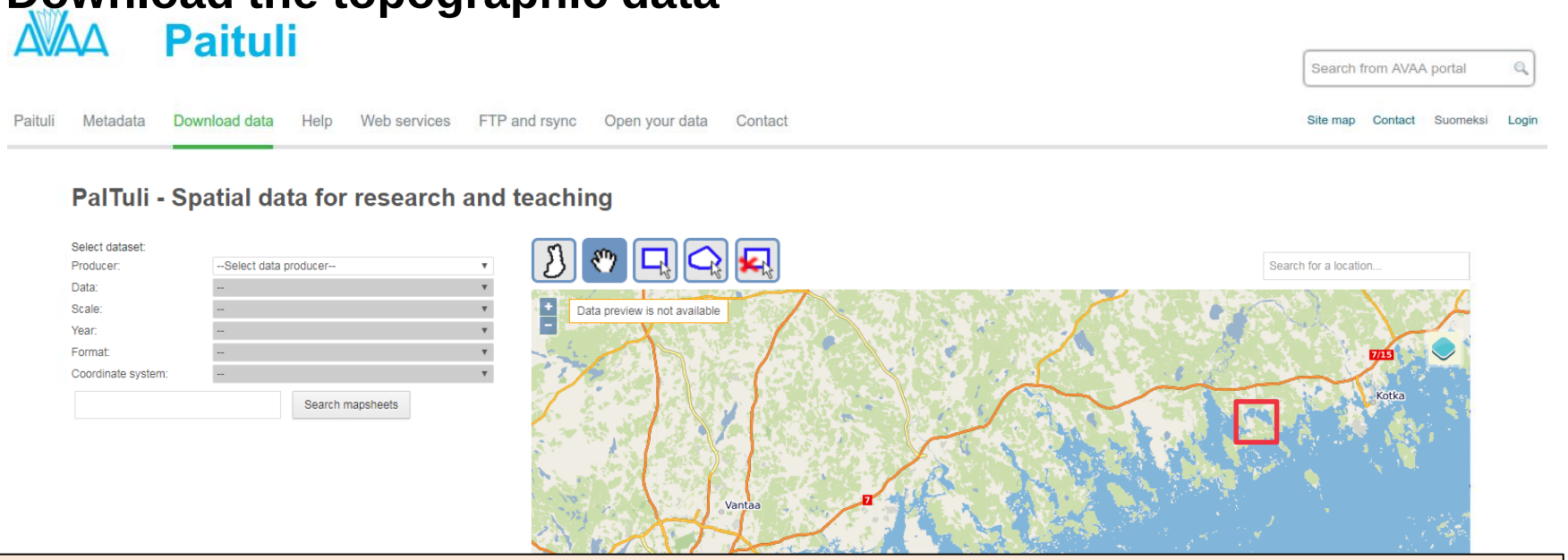
Working with Bash, a Unix shell

Geoprocessing

Procedure

This is ready for you

Download the topographic data




```
$ wget ftp://ftp.funet.fi/pub/sci/geo/geodata/mml/maastotietokanta/2016/L4/L43
```

Geoprocessing

Procedure

Create a polygon of the sea area

We need a polygon that represents the sea area to clip the radiating lines (to create fetch lines).

```
.
├── mxxxxxxp
│   ├── mL4332Rp.cpg
│   ├── mL4332Rp.dbf
│   ├── mL4332Rp.prj
│   ├── mL4332Rp.shp
│   ├── mL4332Rp.shx
│   ├── mL4334Lp.cpg
│   ├── mL4334Lp.dbf
│   ├── mL4334Lp.prj
│   ├── mL4334Lp.shp
│   ├── mL4334Lp.shx
│   ├── mL4334Rp.cpg
│   ├── mL4334Rp.dbf
│   ├── mL4334Rp.prj
│   ├── mL4334Rp.shp
│   ├── mL4334Rp.shx
│   ├── mL4341Rp.cpg
│   ├── mL4341Rp.dbf
│   ├── mL4341Rp.prj
│   ├── mL4341Rp.shp
│   ├── mL4341Rp.shx
│   ├── mL4342Rp.cpg
│   ├── mL4342Rp.dbf
│   ├── mL4342Rp.prj
│   ├── mL4342Rp.shp
│   ├── mL4342Rp.shx
│   ├── mL4343Lp.cpg
│   ├── mL4343Lp.dbf
│   ├── mL4343Lp.prj
│   ├── mL4343Lp.shp
│   ├── mL4343Lp.shx
│   ├── mL4343Rp.cpg
│   ├── mL4343Rp.dbf
│   ├── mL4343Rp.prj
│   ├── mL4343Rp.shp
│   ├── mL4343Rp.shx
│   ├── mL4344Lp.cpg
│   ├── mL4344Lp.dbf
│   ├── mL4344Lp.prj
│   ├── mL4344Lp.shp
│   ├── mL4344Lp.shx
│   ├── mL4344Rp.cpg
│   ├── mL4344Rp.dbf
│   ├── mL4344Rp.prj
│   ├── mL4344Rp.shp
│   └── mL4344Rp.shx
└── paituli_downloads
```

```
$ cd /home/user/YourDir/gis/data/input/topodb/
```

```
$ cd /home/user/YourDir/gis/data/input/topodb/paituli_downloads/
```

```
$ for zip in *; do unzip $zip -d extracts/${zip%%.*}; done
```

```
$ for dir in extracts/*; do for file in $dir/m*p.*; do cp $file
../mxxxxxxp/m$(basename $dir)p.${file##*.}; done; done
```

```
h_L4343R_p.shp
h_L4343R_t.shp
h_L4343R_v.shp
j_L4343R_s.shp
j_L4343R_v.shp
k_L4343R_s.shp
k_L4343R_t.shp
k_L4343R_v.shp
l_L4343R_s.shp
l_L4343R_t.shp
l_L4343R_v.shp
m_L4343R_p.shp
m_L4343R_s.shp
m_L4343R_t.shp
m_L4343R_v.shp
n_L4343R_p.shp
n_L4343R_s.shp
n_L4343R_t.shp
n_L4343R_v.shp
r_L4343R_p.shp
r_L4343R_s.shp
r_L4343R_t.shp
r_L4343R_v.shp
s_L4343R_p.shp
s_L4343R_s.shp
s_L4343R_t.shp
s_L4343R_v.shp
u_L4343R_p.shp
u_L4343R_v.shp
```

```
m_L4343R_p.cpg
m_L4343R_p.dbf
m_L4343R_p.prj
m_L4343R_p.shp
m_L4343R_p.shx
```

$ tree topodb

```
.
├── mxxxxxxp
└── paituli_downloads
    └── extracts
        ├── L4332R.shp.zip
        ├── L4334L.shp.zip
        ├── L4334R.shp.zip
        ├── L4341R.shp.zip
        ├── L4342R.shp.zip
        ├── L4343L.shp.zip
        ├── L4343R.shp.zip
        ├── L4344L.shp.zip
        └── L4344R.shp.zip

3 directories, 9 files
```

```
.
├── mxxxxxxp
└── paituli_downloads
    ├── extracts
    │   ├── L4332R
    │   ├── L4334L
    │   ├── L4334R
    │   ├── L4341R
    │   ├── L4342R
    │   ├── L4343L
    │   ├── L4343R
    │   ├── L4344L
    │   └── L4344R
    ├── L4332R.shp.zip
    ├── L4334L.shp.zip
    ├── L4334R.shp.zip
    ├── L4341R.shp.zip
    ├── L4342R.shp.zip
    ├── L4343L.shp.zip
    ├── L4343R.shp.zip
    ├── L4344L.shp.zip
    └── L4344R.shp.zip

12 directories, 9 files
```

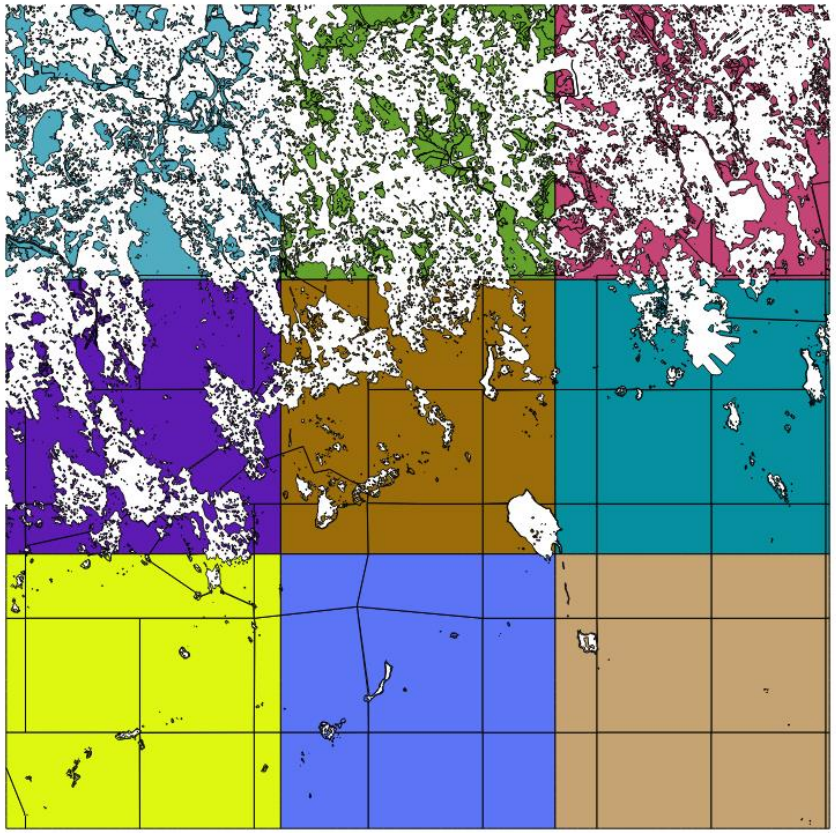
```
mxxxxxxp
├── mL4332Rp.cpg
├── mL4332Rp.dbf
├── mL4332Rp.prj
├── mL4332Rp.shp
├── mL4332Rp.shx
├── mL4334Lp.cpg
├── mL4334Lp.dbf
├── mL4334Lp.prj
├── mL4334Lp.shp
├── mL4334Lp.shx
├── mL4334Rp.cpg
├── mL4334Rp.dbf
├── mL4334Rp.prj
├── mL4334Rp.shp
├── mL4334Rp.shx
├── mL4341Rp.cpg
├── mL4341Rp.dbf
├── mL4341Rp.prj
├── mL4341Rp.shp
├── mL4341Rp.shx
├── mL4342Rp.cpg
├── mL4342Rp.dbf
├── mL4342Rp.prj
├── mL4342Rp.shp
├── mL4342Rp.shx
├── mL4343Lp.cpg
├── mL4343Lp.dbf
├── mL4343Lp.prj
├── mL4343Lp.shp
├── mL4343Lp.shx
├── mL4343Rp.cpg
├── mL4343Rp.dbf
├── mL4343Rp.prj
├── mL4343Rp.shp
├── mL4343Rp.shx
├── mL4344Lp.cpg
├── mL4344Lp.dbf
├── mL4344Lp.prj
├── mL4344Lp.shp
├── mL4344Lp.shx
├── mL4344Rp.cpg
├── mL4344Rp.dbf
├── mL4344Rp.prj
├── mL4344Rp.shp
└── mL4344Rp.shx
paituli_downloads
```

```
$ cd /home/user/YourDir/gis/data/input/topodb/
```

```
$ cd /home/user/YourDir/gis/data/input/topodb/paituli_downloads/
```

```
$ for zip in *; do unzip $zip -d extracts/${zip%%.*}; done
```

```
$ for dir in extracts/*; do for file in $dir/m*p.*; do cp $file
../mxxxxxxp/m$(basename $dir)p.${file##*.}; done; done
```



```
mxxxxxxp
├── mL4332Rp.cpg
├── mL4332Rp.dbf
├── mL4332Rp.prj
├── mL4332Rp.shp
├── mL4332Rp.shx
├── mL4334Lp.cpg
├── mL4334Lp.dbf
├── mL4334Lp.prj
├── mL4334Lp.shp
├── mL4334Lp.shx
├── mL4334Rp.cpg
├── mL4334Rp.dbf
├── mL4334Rp.prj
├── mL4334Rp.shp
├── mL4334Rp.shx
├── mL4341Rp.cpg
├── mL4341Rp.dbf
├── mL4341Rp.prj
├── mL4341Rp.shp
├── mL4341Rp.shx
├── mL4342Rp.cpg
├── mL4342Rp.dbf
├── mL4342Rp.prj
├── mL4342Rp.shp
├── mL4342Rp.shx
├── mL4343Lp.cpg
├── mL4343Lp.dbf
├── mL4343Lp.prj
```

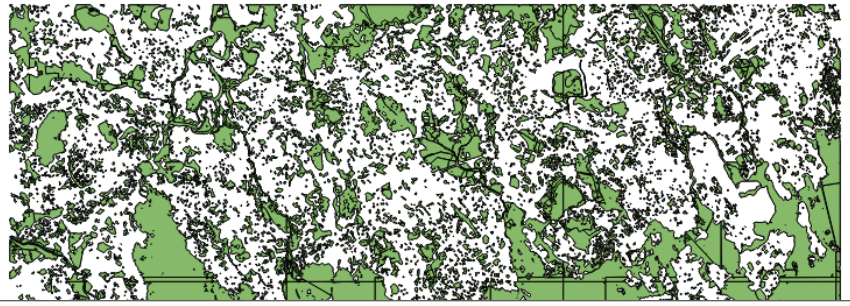
```
$ cd /home/user/YourDir/gis/data/input/topodb/
```

```
$ cd /home/user/YourDir/gis/data/input/topodb/paituli_downloads/
```

```
$ for zip in *; do unzip $zip -d extracts/${zip%%.*}; done
```

```
$ for dir in extracts/*; do for file in $dir/m*p.*; do cp $file
../mxxxxxxp/m$(basename $dir)p.${file##*.}; done; done
```

```
$ cd ../mxxxxxxp
$ for shapefile in *.shp; do ogr2ogr -update -append merger.shp
$shapefile -f "ESRI Shapefile"; done
```



merger :: Features total: 9768, filtered: 9768, selected: 0

	TEKSTI	RYHMA	LUOKKA	TASTAR	KORTAR	KORARV	KULKUTAPA	KOHDEOSO	AINLAHDE	SYNTYHETKI	KUOLHETKI	KART
0	NULL	64	34300	20000	0	0.0	0	387538977	1	20131112	NULL	
1	NULL	64	34300	20000	0	0.0	0	387546854	1	20131107	NULL	
2	NULL	64	34300	20000	0	0.0	0	387546804	1	20131107	NULL	
3	NULL	64	34300	20000	0	0.0	0	387546844	1	20131107	NULL	
4	NULL	64	34300	20000	0	0.0	0	387546889	1	20131107	NULL	
5	NULL	64	34100	20000	0	0.0	0	387537813	1	20131107	NULL	
6	NULL	64	34100	20000	0	0.0	0	387537803	1	20131108	NULL	
7	NULL	64	34100	20000	0	0.0	0	387537808	1	20131107	NULL	
8	NULL	64	34100	20000	0	0.0	0	387537798	1	20131108	NULL	
9	NULL	64	34100	20000	0	0.0	0	387537828	1	20131108	NULL	
10	NULL	64	34100	20000	0	0.0	0	387537908	1	20131108	NULL	

```
$ cd /home/user/YourDir/gis/data/input/topodb/
```

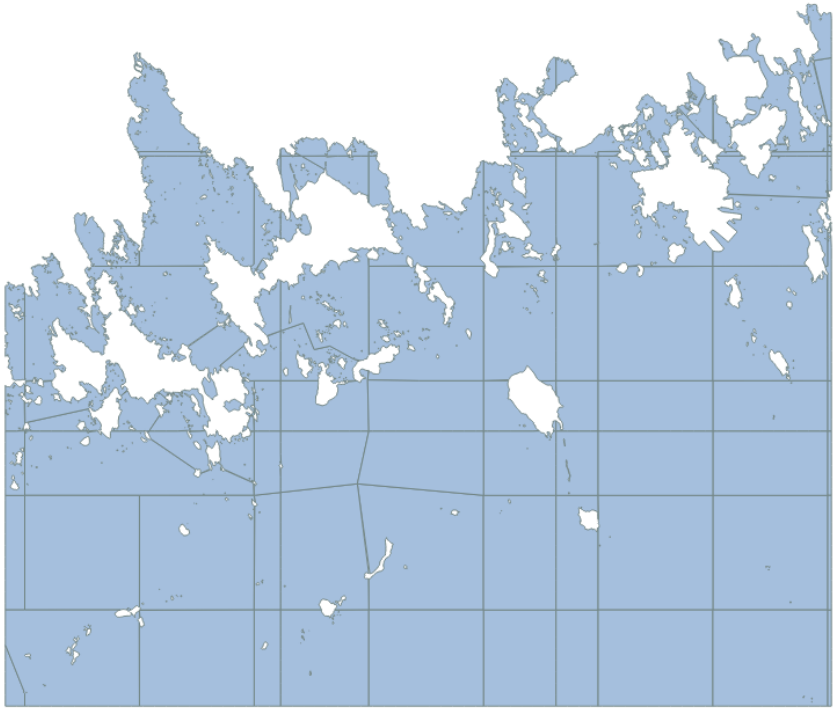
```
$ cd /home/user/YourDir/gis/data/input/topodb/paituli_downloads/
```

```
$ for zip in *; do unzip $zip -d extracts/${zip%%.*}; done
```

```
$ for dir in extracts/*; do for file in $dir/m*p.*; do cp $file
../mxxxxxxp/m$(basename $dir)p.${file##*.}; done; done
```

```
$ cd ../mxxxxxxp
$ for shapefile in *.shp; do ogr2ogr -update -append merger.shp
$shapefile -f "ESRI Shapefile"; done
```

```
$ ogr2ogr -sql "SELECT * FROM merger WHERE LUOKKA=36211"
sea_water_LUOKKA_36211.shp merger.shp
```



```
$ cd /home/user/YourDir/gis/data/input/topodb/
```

```
$ cd /home/user/YourDir/gis/data/input/topodb/paituli_downloads/
```

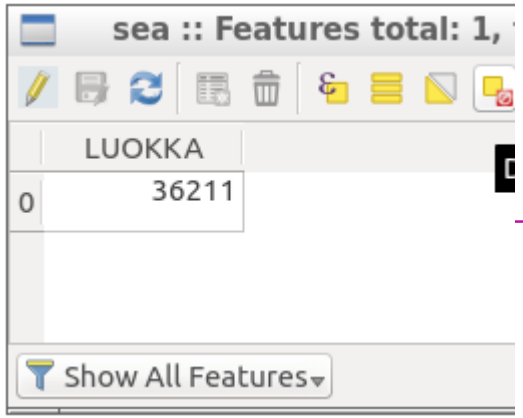
```
$ for zip in *; do unzip $zip -d extracts/${zip%%.*}; done
```

```
$ for dir in extracts/*; do for file in $dir/m*p.*; do cp $file
../mxxxxxxp/m$(basename $dir)p.${file##*.}; done; done
```

```
$ cd ../mxxxxxxp
$ for shapefile in *.shp; do ogr2ogr -update -append merger.shp
$shapefile -f "ESRI Shapefile"; done
```

```
$ ogr2ogr -sql "SELECT * FROM merger WHERE LUOKKA=36211"
sea_water_LUOKKA_36211.shp merger.shp
```

```
$ ogr2ogr -dialect sqlite -sql "SELECT ST_Union(Geometry),
LUOKKA FROM sea_water_LUOKKA_36211 GROUP BY LUOKKA"
/home/user/YourDir/gis/data/output/sea_
sea_water_LUOKKA_3
```

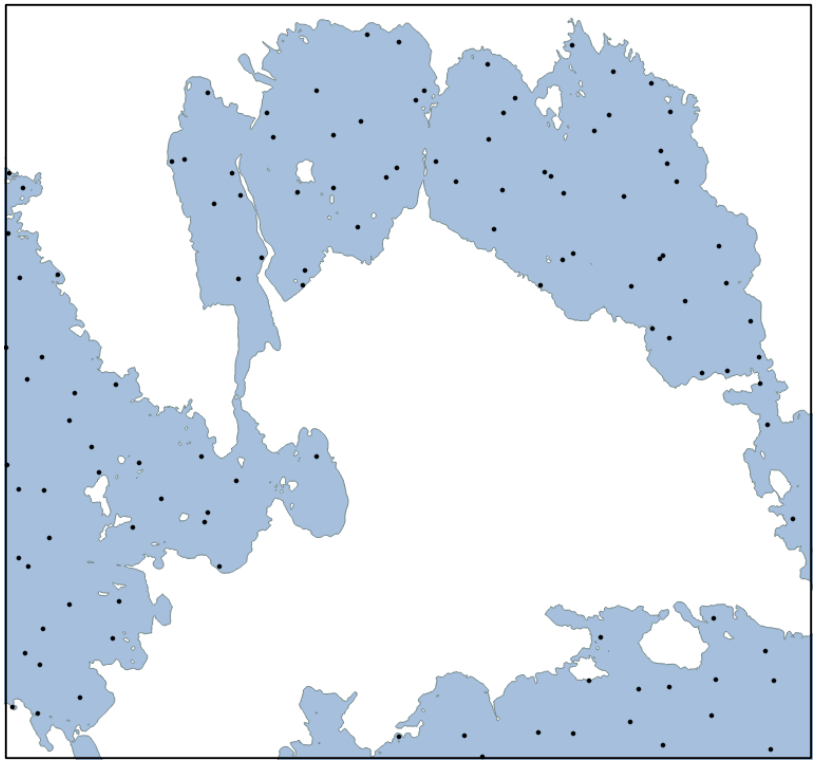


Geoprocessing

Procedure

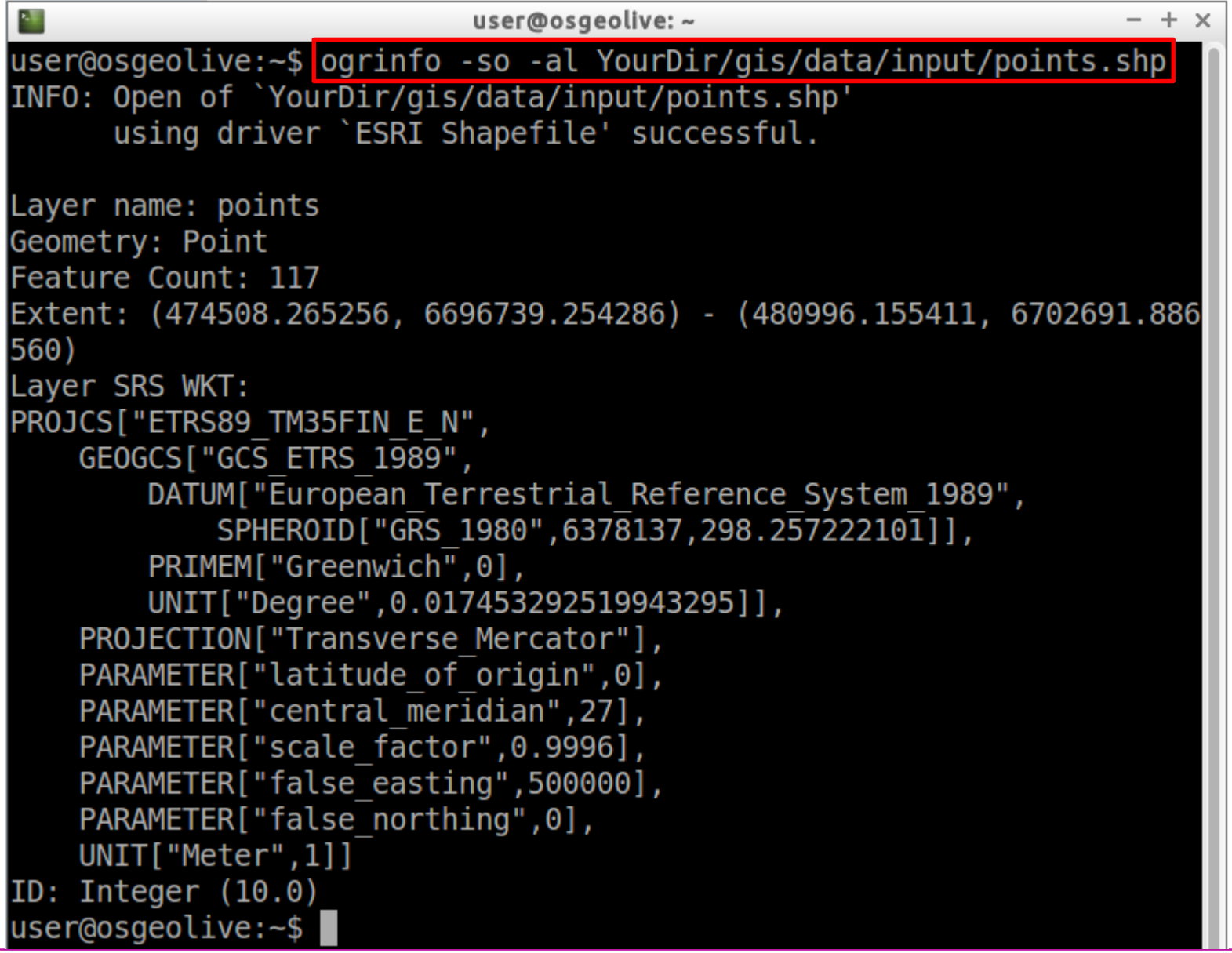
This is ready for you

Create a grid of points



Attribute table

	ID
20	118
21	119
22	121
23	122
24	123
25	125
26	126
27	127
28	128
29	144
30	147
31	148
32	149
33	150
34	151
35	152
36	153
37	154
38	155
39	156
40	157


user@osgeolive: ~
user@osgeolive:~$ ogrinfo -so -al YourDir/gis/data/input/points.shp
INFO: Open of `YourDir/gis/data/input/points.shp'
using driver `ESRI Shapefile' successful.
Layer name: points
Geometry: Point
Feature Count: 117
Extent: (474508.265256, 6696739.254286) - (480996.155411, 6702691.886560)
Layer SRS WKT:
PROJCS["ETRS89_TM35FIN_E_N",
GEOGCS["GCS_ETRS_1989",
DATUM["European_Terrestrial_Reference_System_1989",
SPHEROID["GRS_1980",6378137,298.257222101]],
PRIMEM["Greenwich",0],
UNIT["Degree",0.0174532925199433295]],
PROJECTION["Transverse_Mercator"],
PARAMETER["latitude_of_origin",0],
PARAMETER["central_meridian",27],
PARAMETER["scale_factor",0.9996],
PARAMETER["false_easting",500000],
PARAMETER["false_northing",0],
UNIT["Meter",1]]
ID: Integer (10.0)
user@osgeolive:~$

Geoprocessing

Procedure

Create radiating lines

- Twelve radiating lines from each point in the grid
- Each line is 10000 m in length

class **pysal.cg.shapes.Point**(*loc*) [source]

Geometric class for point objects.

class **pysal.cg.shapes.Ray**(*origin*, *second_p*) [source]

Geometric representation of ray objects.

o

Point – Origin (point where ray originates)

p

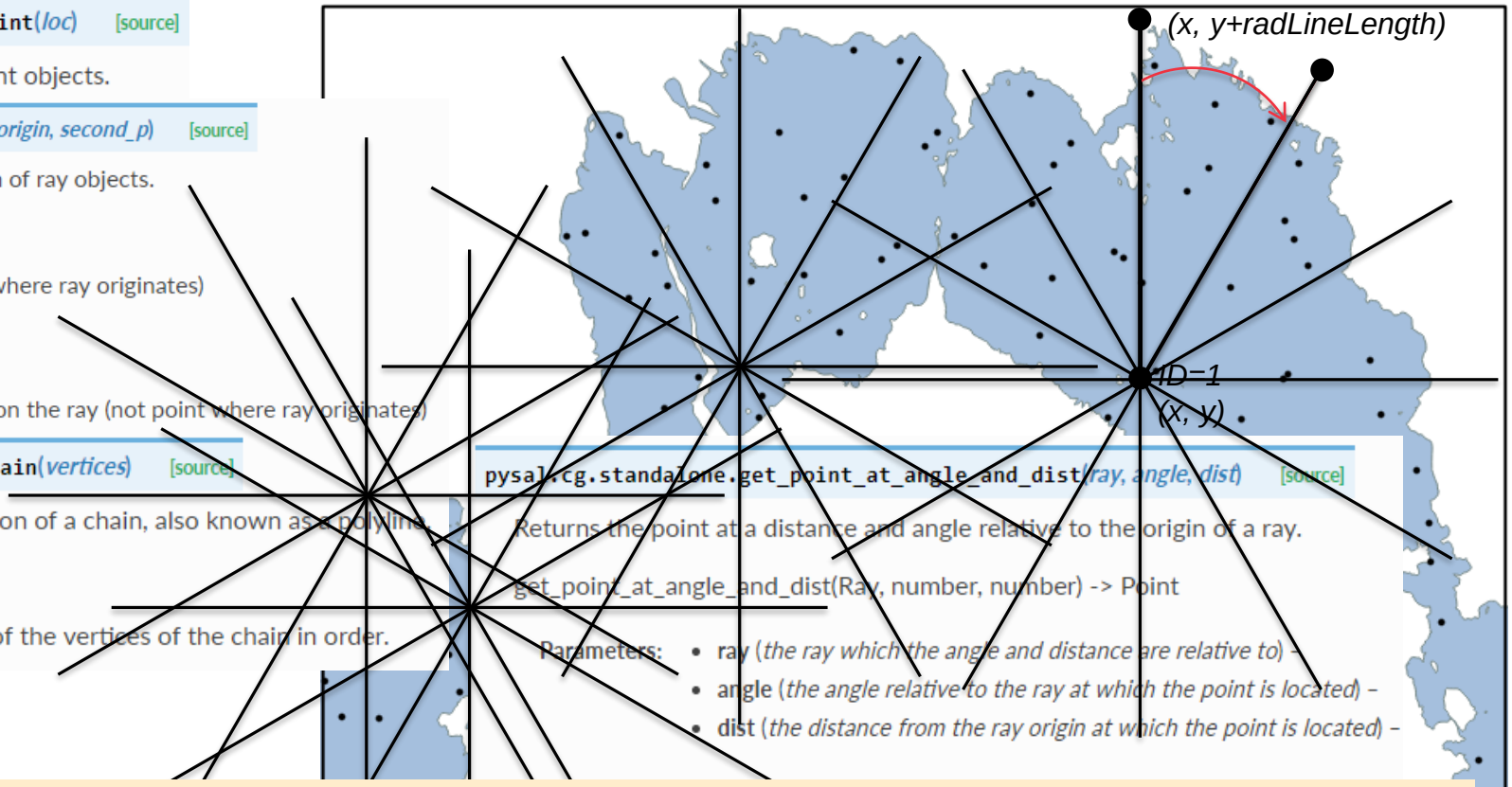
Point – Second point on the ray (not point where ray originates)

class **pysal.cg.shapes.Chain**(*vertices*) [source]

Geometric representation of a chain, also known as a polyline.

vertices

list – List of Points of the vertices of the chain in order.



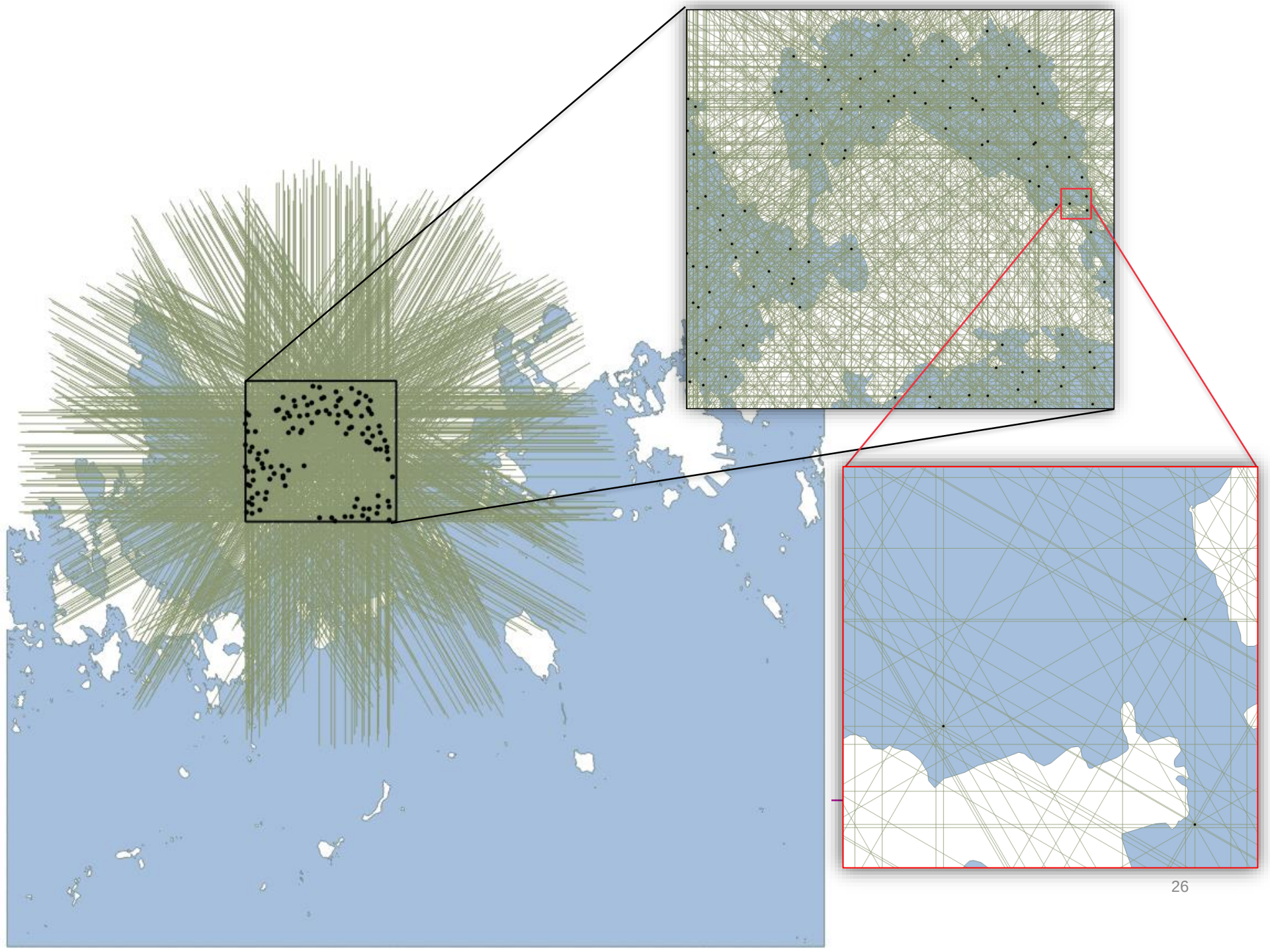
pysal.cg.standalone.get_point_at_angle_and_dist(*ray*, *angle*, *dist*) [source]

Returns the point at a distance and angle relative to the origin of a ray.

get_point_at_angle_and_dist(Ray, number, number) -> Point

Parameters:
- **ray** (*the ray which the angle and distance are relative to*) –
- **angle** (*the angle relative to the ray at which the point is located*) –
- **dist** (*the distance from the ray origin at which the point is located*) –

```
for p in range(len(pt)):
    id = ptAtt.read_record(p)
    origin = pt.get(p)
    lineEnd = pysal.cg.shapes.Point((pt.get(p)[0], pt.get(p)[1] + radLineLength))
    ray = pysal.cg.shapes.Ray(origin, lineEnd)
    line = pysal.cg.shapes.Chain([origin, lineEnd])
    radLines.write(line)
    radLinesAtt.write(id)
    for r in range(numRadLines-1):
        lineEnd = pysal.cg.standalone.get_point_at_angle_and_dist(ray, angle, radLineLength)
        ray = pysal.cg.shapes.Ray(origin, lineEnd)
        line = pysal.cg.shapes.Chain([origin, lineEnd])
        radLines.write(line)
        radLinesAtt.write(id)
```



```
# Import necessary libraries
import pysal # To read and write Shapefiles and do basic geometry calculations
import math # To use the number Pi for calculating the angle in Radian
# Full path to input Shapefile (a grid of points)
inFile = "/home/user/YourDir/gis/data/input/points.shp"
# Full path to output Shapefile (radiating lines)
outFile = "/home/user/YourDir/gis/data/output/rad_lines.shp"
# Length of a radiating line
radLineLength = 10000
# Number of lines to be created around each point
numRadLines = 12
# Calculate the deviating angle between consecutive lines based on 'numRadLines'
angle = (math.pi * 2) / numRadLines # in Radian
# Open (read only) the input Shapefile '.shp' and its attribute table '.dbf'.
pt = pysal.open(inFile.split(".")[0] + ".shp","r")
ptAtt = pysal.open(inFile.split(".")[0] + ".dbf","r")
# Open (for writing ) the output Shapefile '.shp' and its attribute table '.dbf'
radLines = pysal.open(outFile.split(".")[0] + ".shp","w")
radLinesAtt = pysal.open(outFile.split(".")[0] + ".dbf","w")
# Assign for the output attribute table the same header and properties of the input attribute table
radLinesAtt.header = ptAtt.header
radLinesAtt.field_spec = ptAtt.field_spec
# Loop over each point in the grid
for p in range(len(pt)):
    # Read the ID of the point (to assign it to all lines around the point)
    id = ptAtt.read_record(p)
    origin = pt.get(p) # Get the point in question
    # Create the first radiating line
    lineEnd = pysal.cg.shapes.Point((pt.get(p)[0], pt.get(p)[1] + radLineLength))
    ray = pysal.cg.shapes.Ray(origin, lineEnd)
    line = pysal.cg.shapes.Chain([origin, lineEnd])
    # Write the line and its ID to the output Shapefile
    radLines.write(line)
    radLinesAtt.write(id)
    # Now that the first line is created, start creating the remaining lines around the point.
    # Loop over the remaining lines
    for r in range(numRadLines-1):
        lineEnd = pysal.cg.standalone.get_point_at_angle_and_dist(ray, angle, radLineLength)
        ray = pysal.cg.shapes.Ray(origin, lineEnd)
        line = pysal.cg.shapes.Chain([origin, lineEnd])
        # Write the line and its ID to the output Shapefile
        radLines.write(line)
        radLinesAtt.write(id)
# Close all files
pt.close()
ptAtt.close()
radLines.close()
radLinesAtt.close()
```

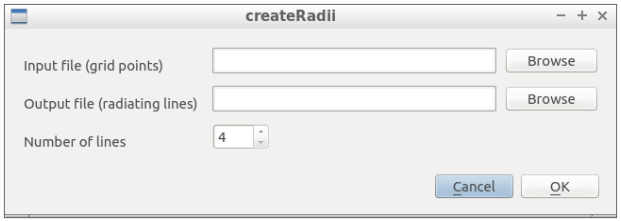
Adding a user-interface

Command-line

- `raw_input()`
- `optparse`
  (https://docs.python.org/2/library/optparse.html)

GUI

- QGIS Plugin
- ArcGIS Add-in



Adding a user-interface

radlines.py

```
from optparse import OptionParser
from create_radiating_lines import createRadiatingLines

parser = OptionParser(usage='Create radiating lines around points.')
parser.add_option('-i', '--input', type=str, help='A point vector layer (.shp)')
parser.add_option('-o', '--output', type=str, help='Name of the output Shapefile (radiating lines)')
parser.add_option('-n', '--lines', type=int, default=12, help='Number of lines to be created')
opt, args = parser.parse_args()

createRadiatingLines(opt.input, opt.output, opt.lines)
print "Done!"
```

create_radiating_lines.py

```
import pysal
import math
def createRadiatingLines(input, output, lines):
        inFile = input
        outFile = output
        numRadLines = lines
        radLineLength = 10000
        # Deviating angle between consecutive lines in Radian
        angle = (math.pi * 2) / nu
        pt = pysal.open(inFile.spl
        ptAtt = pysal.open(inFile.
        radLines = pysal.open(outF
        radLinesAtt = pysal.open(o
        radLinesAtt.header = ptAtt
        radLinesAtt.field_spec = p
        # Loop over each point in
        for p in range(len(pt)):
            id = ptAtt.read_record
            origin = pt.get(p) # G
            lineEnd = pysal.cg.sha
            ray = pysal.cg.shapes.
            line = pysal.cg.shapes
            # Write the line and i
            radLines.write(line)
            radLinesAtt.write(id)
            # Create the remaining
            for r in range(numRadL
                lineEnd = pysal.cg.standalone.get_point_at_angle_and_dist(ray, angle, radLineLength)
                ray = pysal.cg.shapes.Ray(origin, lineEnd)
                line = pysal.cg.shapes.Chain([origin, lineEnd])
                # Write the line and its ID to the output Shapefile
                radLines.write(line)
                radLinesAtt.write(id)
        # Close all files
        pt.close()
        ptAtt.close()
        radLines.close()
        radLinesAtt.close()
```

```
$ python radlines.py --help
Usage: Create radiating lines around points.

Options:
  -h, --help            show this help message and exit
  -i INPUT, --input=INPUT
                        A point vector layer (.shp)
  -o OUTPUT, --output=OUTPUT
                        Name of the output Shapefile (radiating lines)
  -n LINES, --lines=LINES
                        Number of lines to be created
```

```
$ python radlines.py -i points.shp -o radlines_36.shp -n 36
```

Extending a desktop GIS

QGIS plugin

An example plugin

Resources for developers:

http://docs.qgis.org/2.8/pdf/en/QGIS-2.8-PyQGISDeveloperCookbook-en.pdf
http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/index.html
http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/plugins.html
http://www.qgistutorials.com/en/docs/building_a_python_plugin.html
http://nathanw2.github.io/dmsnz-pyqgis-workshop/started/

Plugin Builder

Plugin Reloader

PyQt4: A cross-platform application and UI framework

Qt Designer

Extending a desktop GIS

QGIS plugin

Plugin files¶

Here's the directory structure of our example plugin

```
PYTHON_PLUGINS_PATH/
  MyPlugin/
    __init__.py     --> *required*
    mainPlugin.py   --> *required*
    metadata.txt    --> *required*
    resources.qrc   --> *likely useful*
    resources.py    --> *compiled version, likely useful*
    form.ui         --> *likely useful*
    form.py         --> *compiled version, likely useful*
```

What is the meaning of the files:

- `__init__.py` = The starting point of the plugin. It has to have the `classFactory()` method and may have any other initialisation code.
- `mainPlugin.py` = The main working code of the plugin. Contains all the information about the actions of the plugin and the main code.
- `resources.qrc` = The .xml document created by Qt Designer. Contains relative paths to resources of the forms.
- `resources.py` = The translation of the .qrc file described above to Python.
- `form.ui` = The GUI created by Qt Designer.
- `form.py` = The translation of the form.ui described above to Python.
- `metadata.txt` = Required for QGIS >= 1.8.0. Containts general info, version, name and some other metadata used by plugins website and plugin infrastructure. Since QGIS 2.0 the metadata from `__init__.py` are not accepted anymore and the `metadata.txt` is required.

Source: https://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/plugins/plugins.html#plugin-files

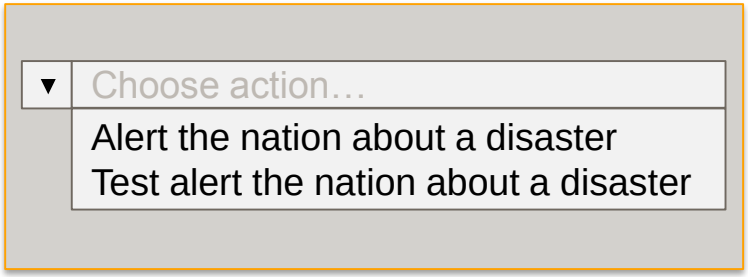
Extending a desktop GIS

ArcGIS add-in

An example add-in: Shape Metrics toolbox (http://clear.uconn.edu/tools/Shape_Metrics/download.htm)

A free online course on customizing ArcGIS and developing an ESRI Add-In is available from the Department of Geography, Penn State University, at: https://www.e-education.psu.edu/geog489/

User-interface development



User interface

- Command-line
- Graphical

Usability Engineering[1] (not only for the UI)

- A system should be easy to learn and remember, be efficient in performing tasks, have low error rates and easy recovery from errors, and be pleasant to use[1]

Human-computer interaction[2]

Designing a user interface (for geospatial applications)[3]

[1] Nielsen, J. (1993). Usability Engineering. San Diego: Academic press, San diego, CA. 362 p.
[2] Skarlatidou, A. (2010). Web-mapping applications and HCI considerations for their design. In: M. Haklay (Ed.), Interacting with Geospatial Technologies. Wiley Blackwell, UK, pp. 245–264.
[3] Nivala, A. M., Sarjakoski, L. T., Sarjakoski, T. (2007). Usability methods' familiarity among map application developers. International journal of human-computer studies, 65(9), 784-795.

Geoprocessing

Procedure

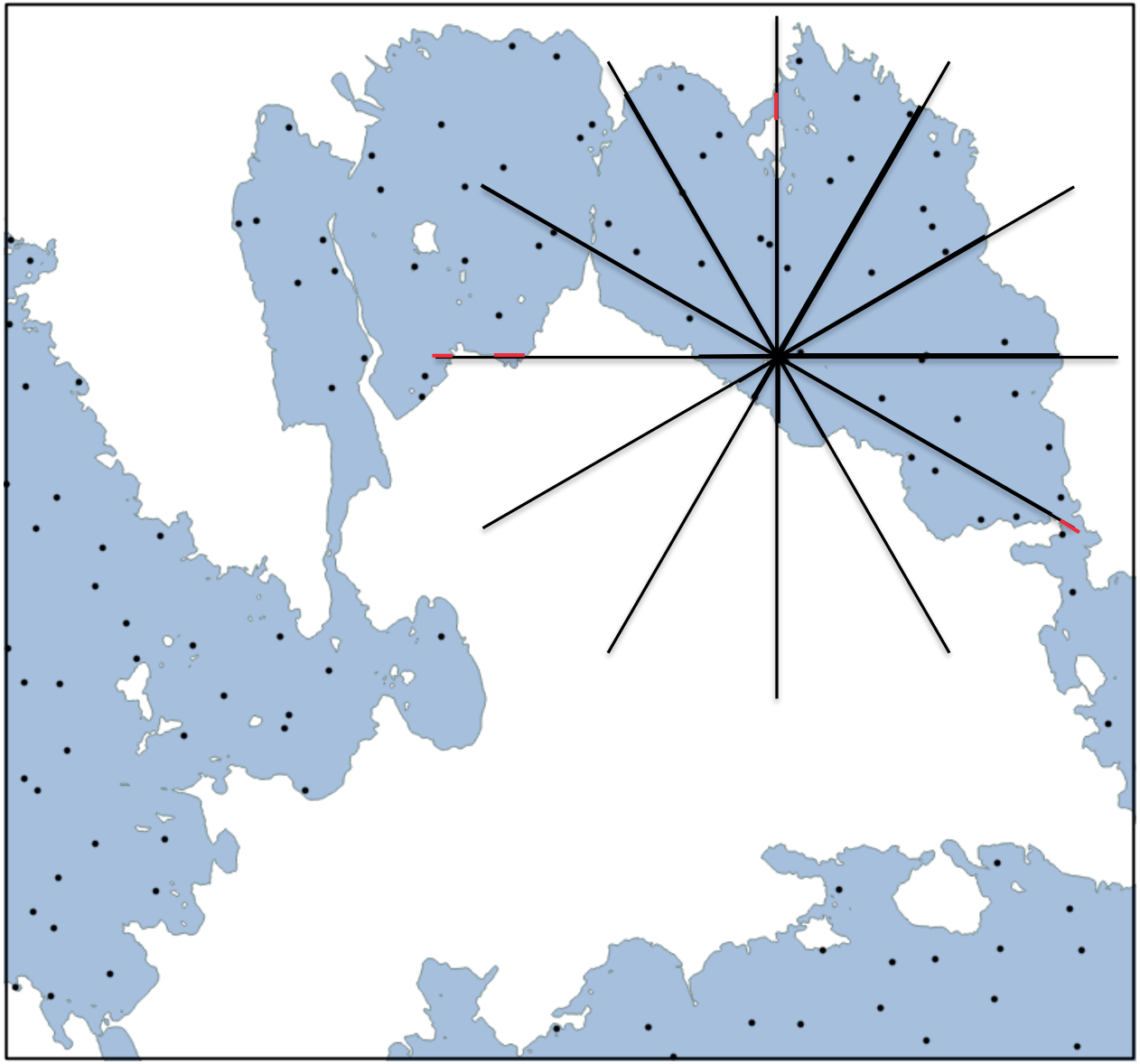
- Download the topographic data
- Create polygons of the sea area
- Create a grid of points
- Create radiating lines

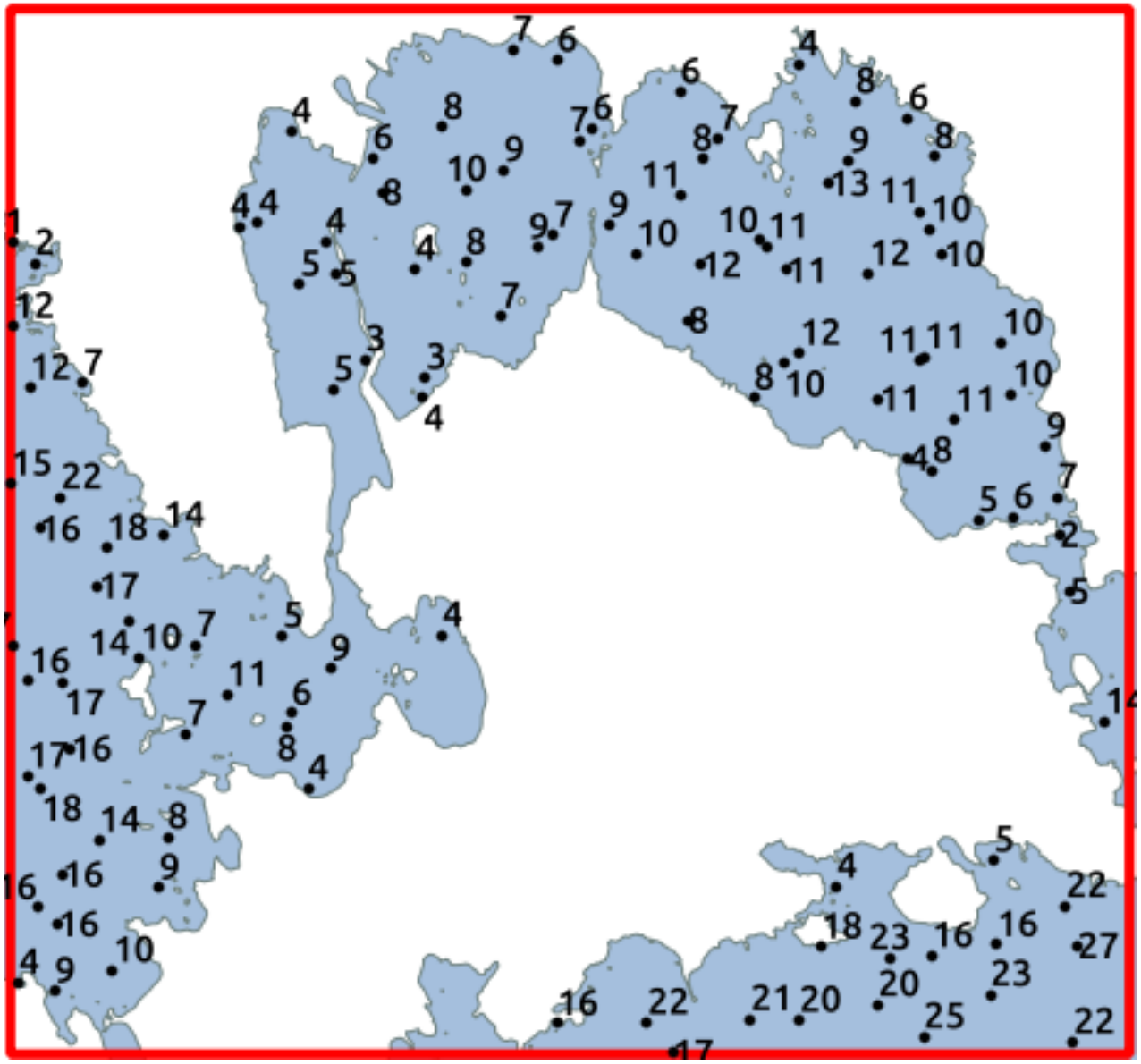
- Clip and clean the radiating lines
- Aggregate the fetch lengths around each grid point
- Interpolate an openness grid

Prepare the data
- Command-line tools
- Software library

Part II – next lecture & exercise

Do the computations
- Spatial database





Geoprocessing

Software

We will use OSGeo-Live VM

"A self-contained bootable DVD, USB thumb drive or Virtual Machine based on Lubuntu, that allows trying a wide variety of open source geospatial software without installing anything"[1]

All software applications and tools needed for our exercise (and many more) are available in this virtual machine.

[1] Source: https://live.osgeo.org

Geoprocessing

Software

Bash (Unix shell) command-line interface

`cd`	Changes location to another directory
`mkdir`	Creates a new directory
`for`	Runs a command (or more) for a set of items in a list
`unzip`	Extracts compressed files from a ZIP archive
`cp`	Copies files or directories

GDAL/OGR – A translator library for raster and vector spatial data formats

ogr2ogr A command-line program that allows manipulating vector data and converting between various file formats

QGIS – An open-source desktop GIS

Geoprocessing

Software

Python & libraries

PySAL	A Python library for spatial analysis

PostgreSQL & PostGIS – An open source object-relational database management system with a spatial extension

pgAdmin	An administration platform for PostgreSQL with GUI
SQL	Structured Query Language – a standard language for accessing and managing databases

GeoServer – An open-source server for publishing geospatial

WMS	Web Map Service
WFS	Web Feature Service

Exercise material

gis	This is the main directory.
|__code	Here you find the code in all listings in this document.
|__data	
| |__input	Here you find the input data.
| |__output	[*empty*] Here you can save the output data.
|__libraries	Here you find the PySAL library that we will install and use for geoprocessing.
|__ready	Here you find readymade data: the data you need to produce yourself in some steps during the exercise. You should use the data you produce, but if you find yourself stuck at one point, you can use the data in this directory to proceed with the exercise.

Next lecture & exercise …

In the next lecture, we will talk about web map applications and their design and architecture;

Web GIS planning and design

Database (geospatial data queries)

Server-side application (geospatial web services)

Client-side application (geospatial data loading and rendering)

In the next exercise, we will work with *PostGIS* and perform spatial queries and analysis. We will also set spatial data services using *Geoserver*.