Lecture 19

Overlay and spatial query

GEOG 489 SPRING 2020

Overlay and spatial query

Source layer (x): the layer that is used to determine the selection from the target layer based on some spatial relationship.

Target layer (y): the layer to extract data FROM

R function: over(x, y)

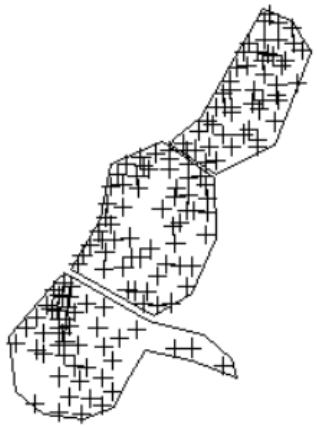
x: geometry (loctions) of the queries

y: layer from which the geometries or attributes are queried

Overlay and spatial query

over(meuse, srdf)

```
> meuse
        coordinates cadmium copper lead zinc   elev       dist   om ffreq soil lime landuse dist.m
1   (181072, 333611)    11.7     85  299 1022  7.909 0.00135803 13.6     1    1    1      Ah     50
2   (181025, 333558)     8.6     81  277 1141  6.983 0.01222430 14.0     1    1    1      Ah     30
3   (181165, 333537)     6.5     68  199  640  7.800 0.10302900 13.0     1    1    1      Ah    150
4   (181298, 333484)     2.6     81  116  257  7.655 0.19009400  8.0     1    2    0      Ga    270
5   (181307, 333330)     2.8     48  117  269  7.480 0.27709000  8.7     1    2    0      Ah    380
6   (181390, 333260)     3.0     61  137  281  7.791 0.36406700  7.8     1    2    0      Ga    470
7   (181165, 333370)     3.2     31  132  346  8.217 0.19009400  9.2     1    2    0      Ah    240
8   (181027, 333363)     2.8     29  150  406  8.490 0.09215160  9.5     1    1    0      Ab    120
9   (181060, 333231)     2.4     37  133  347  8.668 0.18461400 10.6     1    1    0      Ab    240
10  (181232, 333168)     1.6     24   80  183  9.049 0.30970200  6.3     1    2    0       W    420
> srdf
An object of class "SpatialPolygonsDataFrame"
Slot "data":
   X1 X2
r1  1  5
r2  2  4
r3  3  3
> over(meuse,srdf)
   X1 X2
1   1  5
2   1  5
3   1  5
4   1  5
5   1  5
```



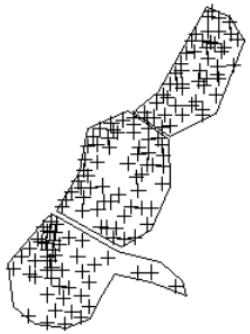
The output is, for each point, the data frame of the polygon it intersected.

Overlay and spatial query

over(srdf, meuse, fn = mean)

```
> meuse
        coordinates cadmium copper lead zinc   elev       dist   om ffreq soil lime landuse dist.m
1   (181072, 333611)    11.7     85  299 1022  7.909 0.00135803 13.6     1    1    1      Ah     50
2   (181025, 333558)     8.6     81  277 1141  6.983 0.01222430 14.0     1    1    1      Ah     30
3   (181165, 333537)     6.5     68  199  640  7.800 0.10302900 13.0     1    1    1      Ah    150
4   (181298, 333484)     2.6     81  116  257  7.655 0.19009400  8.0     1    2    0      Ga    270
5   (181307, 333330)     2.8     48  117  269  7.480 0.27709000  8.7     1    2    0      Ah    380
6   (181390, 333260)     3.0     61  137  281  7.791 0.36406700  7.8     1    2    0      Ga    470
7   (181165, 333370)     3.2     31  132  346  8.217 0.19009400  9.2     1    2    0      Ah    240
8   (181027, 333363)     2.8     29  150  406  8.490 0.09215160  9.5     1    1    0      Ab    120
9   (181060, 333231)     2.4     37  133  347  8.668 0.18461400 10.6     1    1    0      Ab    240
10  (181232, 333168)     1.6     24   80  183  9.049 0.30970200  6.3     1    2    0       W    420
```

```
> srdf
An object of class "SpatialPolygonsDataFrame"
Slot "data":
   X1 X2
r1  1  5
r2  2  4
r3  3  3
```



```
> over(srdf, meuse,fn = mean)
    cadmium   copper     lead     zinc     elev      dist       om ffreq soil lime landuse   dist.m
r1 4.036000 44.48000 147.2600 475.8800 8.225760 0.1791637       NA    NA   NA   NA      NA 237.4000
r2 2.910526 40.22807 145.4035 452.0702 8.485649 0.3207399 7.219298    NA   NA   NA      NA 361.2281
r3 2.820833 36.08333 169.1667 484.2500 7.722208 0.2075471 7.350000    NA   NA   NA      NA 261.2500
```

The output is, for each polygon, the fn (mean) value of the data frame of the points falling within it.

Vector spatial geoprocessing

rgeos library is a wrapper to the Interface to Geometry Engine Open Source (GEOS).

rgeos links sp objects to GEOS' processing.

This has most of the vector spatial geoprocessing you are used to. For instance:

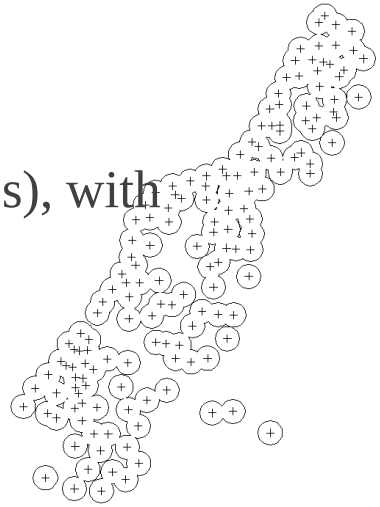
1) Buffer: ?gBuffer
2) Distance between objects: ?gDistance
3) Intersect: ?gIntersection()
4) Are within a distance of: ?gWithinDistance()
5) Are within: ?gCoveredBy()
6) Are completely within: ?gWithin()
7) Contain: ?gContains()
8) Completely contain: ?gCovers()
9) Have their centroid in: ?gCentroid() + ?gIntersection()
10) Share a line segment with: ?gTouches()
11) Touch the boundary of: ?gTouches()
12) Are identical to: ?identical()
13) Are crossed by the outline of: ?gCrosses()

Vector spatial geoprocessing

Buffer:

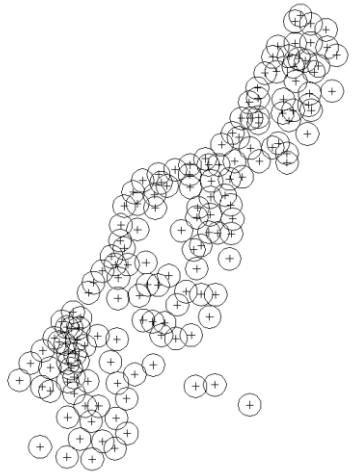
Let's buffer the spatial points meuse out 100 unit (units of coordinates), with overlapping polygons dissolved:

meuse_buffer <- gBuffer(meuse,width=100)



Each polygon is separate.

meuse_buffer_individual <- gBuffer(meuse,width=100,byid=TRUE)



Vector spatial geoprocessing

Distance between objects:

distance_matrix <- gDistance(meuse,meuse,byid=TRUE)

	1	2	3	4	5	6	7	8
1	0.00000	70.83784	118.8486	259.2393	366.3141	473.6296	258.3215	252.0496
2	70.83784	0.00000	141.5662	282.8516	362.6403	471.1995	234.4014	195.0103
3	118.84864	141.56624	0.0000	143.1712	251.0239	356.8669	167.0000	222.0811
4	259.23927	282.85155	143.1712	0.0000	154.2628	242.1570	175.1713	296.7861
5	366.31407	362.64032	251.0239	154.2628	0.0000	108.5772	147.5263	281.9379
6	473.62960	471.19953	356.8669	242.1570	108.5772	0.0000	250.4496	377.3301
7	258.32151	234.40137	167.0000	175.1713	147.5263	250.4496	0.0000	138.1774
8	252.04960	195.01026	222.0811	296.7861	281.9379	377.3301	138.1774	0.0000

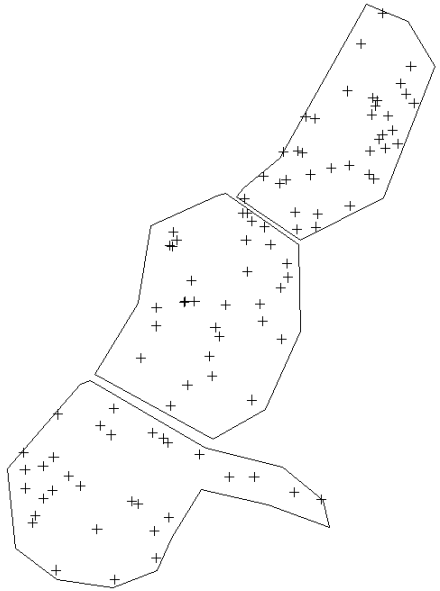
Vector spatial geoprocessing

Spatial sampling: If you are planning on doing spatial field work, proper spatial sampling is an important skill to have.

spsample: This allows you to perform spatial sampling of point locations within a bounding box, grid, polygon, or on a line; given random or regular sampling.

Plots N=100, random points falling within the meuse.sr SpatialPolygon.

```
plot(srdf) # plot the polygon
points(spsample(srdf, n = 100, "random"), pch = 3)
```



Raster Modification

'raster' provided a suite of tools for modifying the spatial configuration of a Raster* object.

- reproject a raster
- crop and expand a raster
- change pixel sizes
- raster algebra

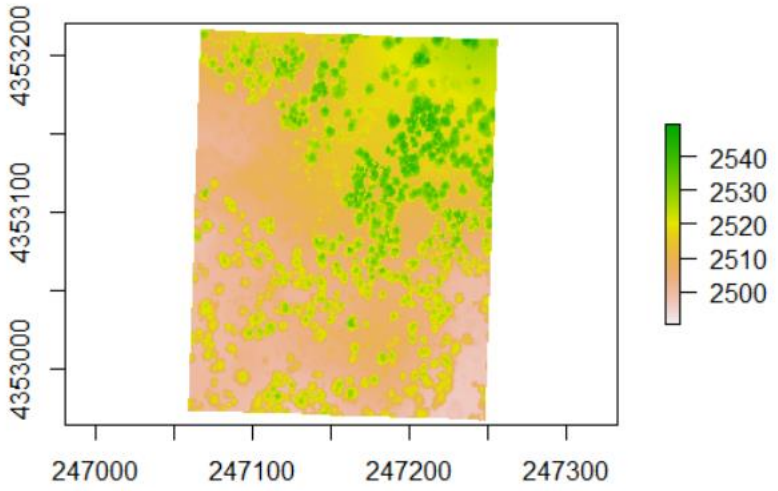
Raster Modification

reproject a raster: projectRaster()

```
# Say we want to project our Lidar data to UTM zone 11:
highest_hit_raster <- raster("tahoe_lidar_highesthit.tif")

# Let's create a CRS string:
utm_zone_11_crs <- CRS("+proj=utm +zone=11
+ellps=WGS84 +datum=WGS84 +units=m +no_defs")

# We can project a raster using this CRS:
highest_hit_raster_utm <-
projectRaster(from=highest_hit_raster,
crs=utm_zone_11_crs)
```

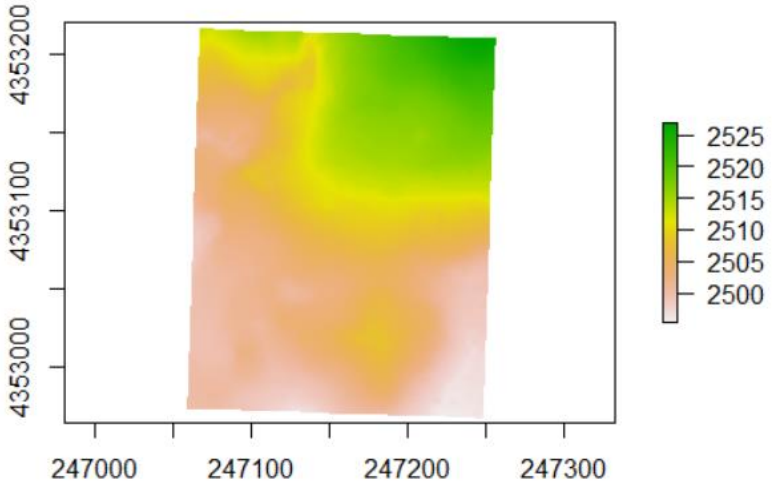


Raster Modification

reproject a raster: projectRaster()

```
# Reference raster
highest_hit_raster <- raster("tahoe_lidar_highesthit.tif")
# raster to be reprojected
bareearth_raster <- raster("tahoe_lidar_bareearth.tif")
```

```
# We can also use a reference raster to project.
highest_hit_raster_ll <-
projectRaster(from=bareearth_raster,
                        to=highest_hit_raster_utm)
```



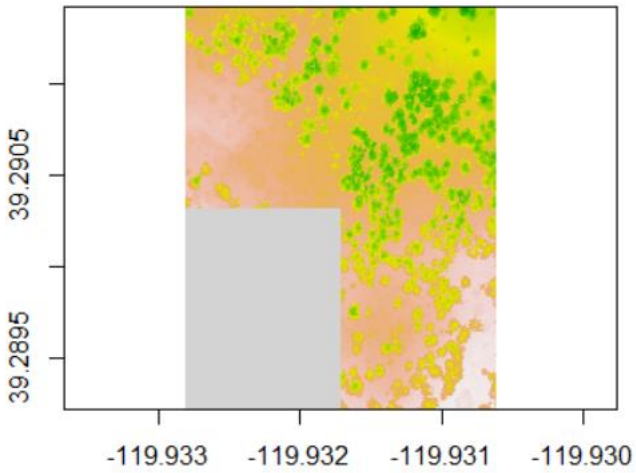
Raster Modification

Crop a raster: crop()

```
# We need to use an "Extent" object to define the crop rectangle:
hh_extent <- extent(highest_hit_raster)

# Let's crop out the bottom left:
middle_x <- mean(c(hh_extent@xmin,hh_extent@xmax))
middle_y <- mean(c(hh_extent@ymin,hh_extent@ymax))
bottomleft <- hh_extent
bottomleft@xmax <- middle_x
bottomleft@ymax <- middle_y

hh_bottomleft <- crop(highest_hit_raster,bottomleft)
```



Raster Modification

Extend a raster: extend()

```
# Let's add 10 pixels to each side.  First we need to know the resolution:
hh_res <- res(highest_hit_raster)     # 400*400 pixels

# So to add 10 pixels, we need to add/subtract a buffer of:
hh_10_pixels <- hh_res * 10
hh_extent <- extent(highest_hit_raster)

hh_extent_10_pixels <- hh_extent
hh_extent_10_pixels@xmin <- hh_extent@xmin - hh_10_pixels[1]
hh_extent_10_pixels@xmax <- hh_extent@xmax + hh_10_pixels[1]
hh_extent_10_pixels@ymin <- hh_extent@ymin - hh_10_pixels[2]
hh_extent_10_pixels@ymax <- hh_extent@ymax + hh_10_pixels[2]

hh_extend <- extend(highest_hit_raster,hh_extent_10_pixels)    # 420*420 pixels
```

Raster Modification

Change pixel sizes: aggregate() and disaggregate()

To make pixels larger, use aggregate().

```
# Aggregate higher resolution (smaller) cells into the lower resolution (larger cell).
The size of the raster changes from 400*400 to 100*100

highest_hit_raster_lower <- aggregate(x=highest_hit_raster, fact=4, fun=mean)
```

To make pixels smaller, use disaggregate()

```
# The size of the raster changes from 400*400 to 800*800

highest_hit_raster_higher <- disaggregate(x=highest_hit_raster, fact=2)
```

Raster Modification

Raster algebra: The basic rule of thumb is that any operation is done on a pixel-by-pixel basis. Mixing different Raster* objects in a statement requires, generally that they be the same number of rows and columns.

Raster algebra: Not all math functions will work in a predictable manner, but minimally these functions will: +, -, *, /, logical operators such as >, >=, <, ==, ! and functions such as abs, round, ceiling, or, trunc, sqrt, log, log10, exp, cos, sin, max, min, range, prod, sum, any, all.

```
# Change the original raster from meters to feet
highest_hit_raster_feet <- highest_hit_raster * 3.28084
```

Raster Modification

Use multiple rasters in a statement, such as calculating the lidar height raster.

height_raster <- highest_hit_raster - bareearth_raster

Logical statements come in handy. How about creating a mask of all heights greater than 6 feet (the definition of a "tree"):

tree_mask <- height_raster_feet > 6

You can use replacement functions similar to a vector. Let's make all the trees the same height, 30 feet tall, but leave the rest of the heights alone:

height_raster_feet[tree_mask==1] <- 30

Or just do some basic masking (use NA in masking):

height_raster_feet[tree_mask==1] <- NA

Raster Modification

We can compare layers using functions such as min, max, mean, prod, sum, Median, cv, range, any, all. This is applied PIXEL BY PIXEL.

mean_height <- mean(bareearth_raster, highest_hit_raster)

For a raster with multiple layers, sum all layers pixel by pixel

tahoe_highrez_brick <- brick("tahoe_highrez.tif") # 3 layers

tahoe_highrez_brick_sum <- sum(tahoe_highrez_brick)

tahoe_highrez_brick_sum # Notice it's only one layer.

Assignment 5

Your goal is to:

1) Convert a data frame of x,y coordinates into a SpatialPoints object.

2) Loop through each point using a foreach statement and determine the hemispheres (North/South and East/West) of each point.

3) Create a SpatialPointsDataFrame object with the hemispheres as attributes.

Due Thursday, April 16, 2020 at midnight.

Assignment 5

```
set.seed(10)
n <- 10
df <- data.frame(xpos=runif(n,0,360),ypos=runif(n,-90,90))
df
#        xpos      ypos
#1  182.69215  27.29802
#2  110.43666  12.19280
#3  153.68676 -69.56838
#4  249.51675  17.26655
#5   30.64895 -25.55100
#6   81.15718 -12.81430
#7   98.83099 -80.65740
#8   98.02982 -42.44802
#9  221.69855 -18.21767
#10 154.68175  60.50415
```

Assignment 5

```
outHemisphere <- hemisphereSummary(df=df)
outHemisphere
#                        coordinates EWhemisphere NShemisphere
#1    (182.692, 27.298)           W            N
#2   (110.437, 12.1928)           E            N
#3  (153.687, -69.5684)           E            S
#4   (249.517, 17.2666)           W            N
#5   (30.6489, -25.551)           E            S
#6  (81.1572, -12.8143)           E            S
#7   (98.831, -80.6574)           E            S
#8   (98.0298, -42.448)           E            S
#9  (221.699, -18.2177)           W            S
#10  (154.682, 60.5041)           E            N
```

Assignment 5

```
summary(outHemisphere)
#Object of class SpatialPointsDataFrame
#Coordinates:
#                          min       max
#xpos  30.64895 249.51675
#ypos -80.65740  60.50415
#Is projected: FALSE
#proj4string : [+proj=longlat +datum=WGS84]
#Number of points: 10
#Data attributes:
#EWhemisphere NShemisphere
#E:7          N:4
#W:3          S:6
```