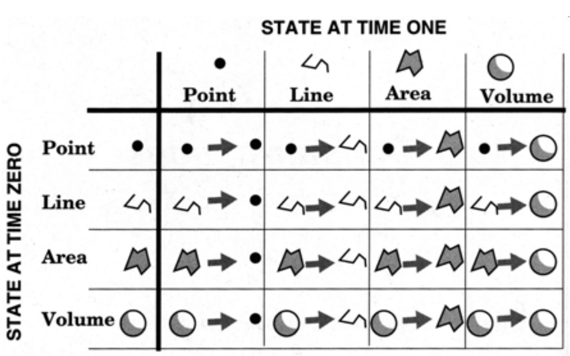
Analytical and Computer Cartography

Lecture 9: Geometric Map Transformations

Cartographic Transformations

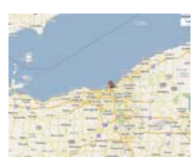
- Attribute Data (e.g. classification)
- Locational properties (e.g. projection)
- Graphics (e.g. symbolization)
- Information content of maps (e.g. data structure conversion)

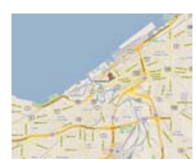
Dimensional Transformations



Geometric map transformations

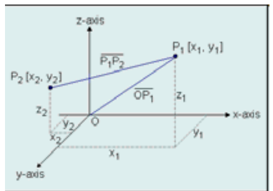
- Same dimension: e.g. point to point in a projection
- Change structure: e.g. TIN to grid in a DEM
- Change scale: e.g. Area to point as a city is generalized

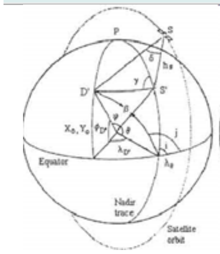


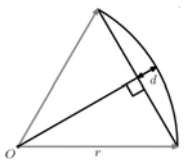


Planar geometries

- Cartesian
- Spherical (ellipsoidal)
- Radial







Analysis

- Input: geometric feature (Point(s), line(s), Area(s), Surface(s), Volumes(s)
- Output: Reduce feature to single dimension: e.g. centroid, length, perimeter, area in square unit lengths
- Output: A scalar, numerical value reflecting quantity "Collapse"
- E.g. shape, sinuosity, network metric
- Can compute using algorithms

Uncertainty in Geometric features

- Assume infinite thinness
- Assume exact location
- Assume unambiguous ontology (definition) e.g. wetlands

MIT International Review | web.mit.edu/mitir

Web-published essay installment for 17 February 2009

Finding Osama bin Laden:

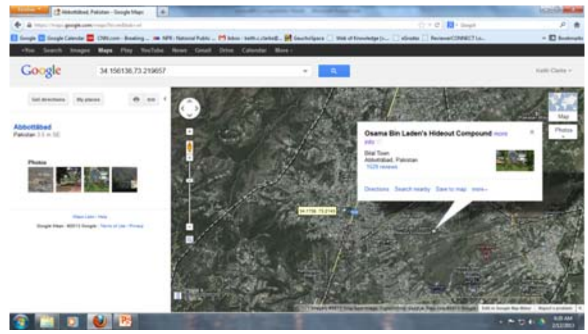
An Application of Biogeographic Theories and Satellite Imagery

Thomas W. Gillespie and John A. Agnew are professors of geography at UCLA. They may be contacted respectively at tg@geog.ucla.edu and jagnew@geog.ucla.edu. Erika Mariano, Scott Mossler, Nolan Jones, Matt Braughton, and Jorge Gonzalez are undergraduates in UCLA's geography department. They may be contacted respectively at erikmari@ucla.edu, smossler@ucla.edu, nolanjones@ucla.edu, mbraught@ucla.edu, and jorgon@ucla.edu.[1]

Likely location and setting (3 given)

Actually at: 34.156136,73.219657

Abbottabad



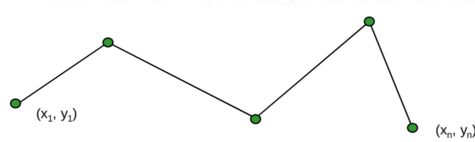
Assuming crisp features: Points, lines, areas, volumes

- What simple transformations can be performed on PLAV in Euclidean space?
- In C language
- typedef struct POINT { int point_id, double x, double y};
- POINT Point[100]; in npts;
- Create a loop
  - double sumx = 0.0
  - for (i=0; I <= npts; i++) sumx += Point[i].x;
  - meanx = sumx / npts;

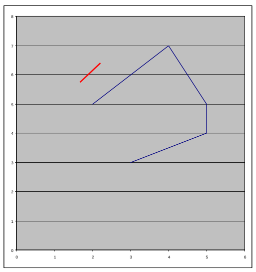
Planar Map Transformations on Points
- Length of a line

- Repetitive application of point-to-point distance calculation
- For n points, algorithm/formula uses n-1 segments

$$\text{length} = \sum_{i=1}^{npts} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$



Length of a Line



Length of a generic line

i	x_i	y_i	x_i-x_{i-1}	y_i-y_{i-1}	$(x_i-x_{i-1})^2$	$(y_i-y_{i-1})^2$	$(x_i-x_{i-1})^2+(y_i-y_{i-1})^2$	$(x_i-x_{i-1})^2+(y_i-y_{i-1})^2$	$\text{sqrt}[(x_i-x_{i-1})^2+(y_i-y_{i-1})^2]$
1	3	3	--	--	--	--	--	--	--
2	5	4	2	1	4	1	5	5	2.236
3	5	5	0	1	0	1	1	1	1.000
4	4	7	-1	2	1	4	5	5	2.236
5	2	5	-2	-2	4	4	8	8	2.828

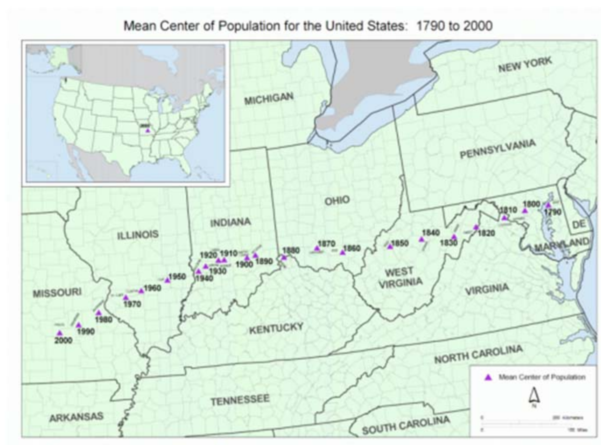
Sum =

8.3 units

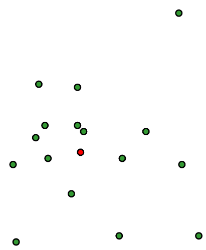
Planar Map Transformations on Points
- Centroids

- Multiple point or line or area to be transformed to single point
- Point can be "real" or representative
- Can use weightings or populations
- Mean center simple to compute but may fall outside point cluster or polygon
- Can use point-in-polygon to test for inclusion

$$\bar{x} = \frac{\sum_{i=1}^{npts} P_i x_i}{\sum_{i=1}^{npts} P_i} \qquad \bar{y} = \frac{\sum_{i=1}^{npts} P_i y_i}{\sum_{i=1}^{npts} P_i}$$



Mean point vs. centroid



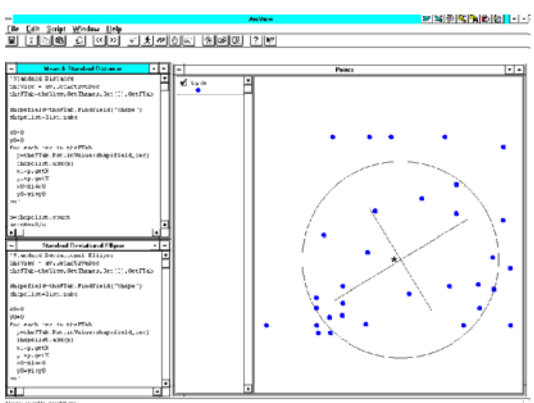
Planar Map Transformations on Points
- Standard Distance

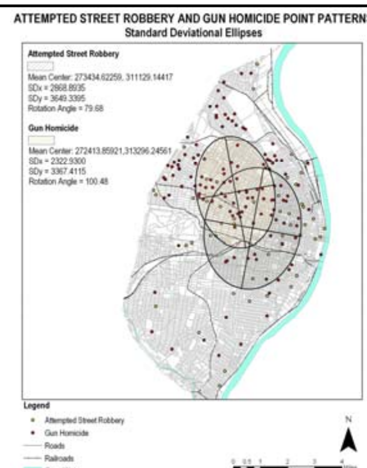
- Just as centroid is an indication of representative location, standard distance is mean dispersion
- Equivalent of standard deviation for an attribute, mean variation from mean
- Around centroid, makes a "radius" tracing a circle

$$s_x = \sqrt{\frac{\sum_{i=1}^{npts} (x_i - \bar{x})^2}{npts}} \qquad s_y = \sqrt{\frac{\sum_{i=1}^{npts} (y - \bar{y})^2}{npts}}$$

$$s = \sqrt{(s_x^2 + s_y^2)}$$

Standard distance

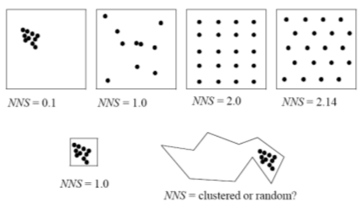




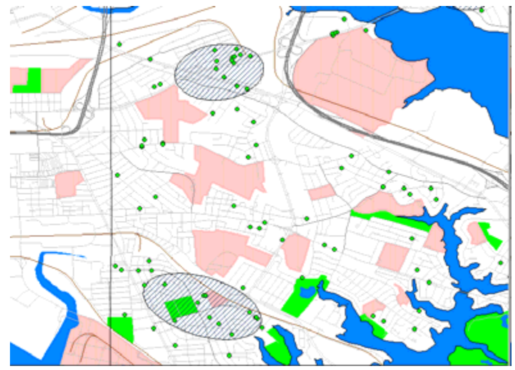
Planar Map Transformations on Points
- Nearest Neighbor Statistic

- NNS is a single dimensionless scalar that measures the pattern of a set of point (point-> scalar)
- Computes nearest point-to-point separation as a ratio of expected given the area
- Highly sensitive to the area chosen

$$NNS = 2 \times \frac{\sum_{i=1}^{npts} d_i}{npts \times \sqrt{\frac{A}{npts}}}$$

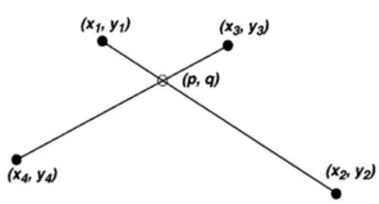


Local NNS: Cluster busting

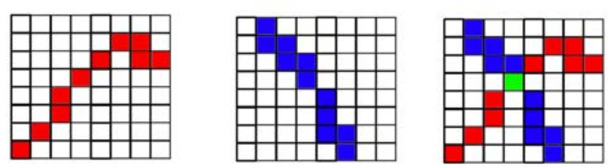


Planar Map Transformations Based on Lines
- Intersection of two lines

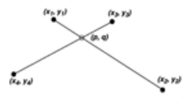
- Absolutely fundamental to many mapping operations, such as overlay and clipping.
- In raster mode it can be solved by layer overlay.
- In vector mode it must be solved geometrically.
- Lines (2) to point transformation



Raster solution: add binary arrays



Planar Map Transformations Based on Lines - Intersection of two lines (cnt.)



- When using this algorithm, a problem exists when b2 - b1 = 0 (divide by zero)
- Special case solutions or tests must be used
- These can increase computation time greatly
- Computation time can be reduced by pre-testing, e.g. based on bounding box.

If (x_1, y_1) and (x_2, y_2) lie on the same line, then

$$y_1 = a_1 + b_1 x_1$$
$$y_2 = a_1 + b_1 x_2$$

Similarly, if (x_3, y_3) and (x_4, y_4) lie on the same line, then

$$y_3 = a_2 + b_2 x_3$$
$$y_4 = a_2 + b_2 x_4$$

If there exists an intersection point, (p, q) that lies on both lines, then

$$q = a_1 + b_1 p$$
$$q = a_2 + b_2 p$$

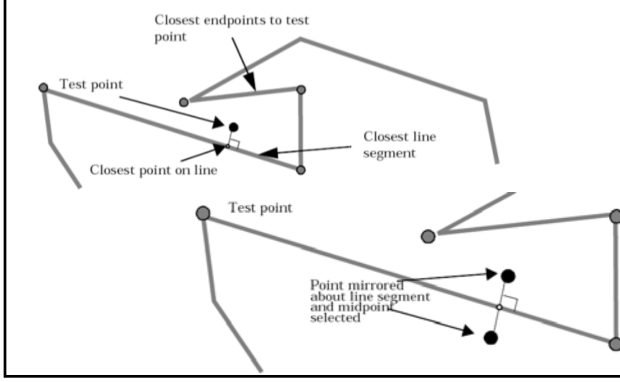
By subtracting the former from the latter,

$$q - q = a_1 - a_2 + p\,(b_1 - b_2)$$

and rearranging, we obtain

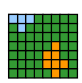
$$a_1 - a_2 = p\,(b_2 - b_1)$$

Planar Map Transformations Based on Lines - Distance from a Point to a Line



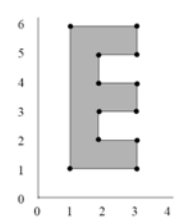
Planar Map Transformations Based on Areas

- Computing the area of a vector polygon (closed)
- Manually, many methods are used, e.g. cell counts, point grid.
- For a raster, simply count the interior pixels
- Vector Mode more complex



$$A = \frac{1}{2}\left|\sum_{i=1}^{npts+1} (x_i y_{i-1}) - (x_{i-1} y_i)\right|$$

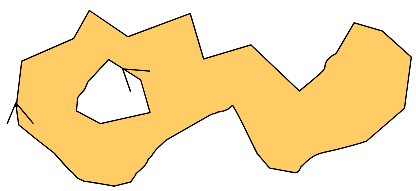
Planar Map Transformations Based on Areas



Vertex	x	y	A	B	Difference (A–B)
1	1	1	—	—	—
2	1	6	1	6	–5
3	3	6	18	6	12
4	3	5	18	15	3
5	2	5	10	15	–5
6	2	4	10	8	2
7	3	4	12	8	4
8	3	3	12	9	3
9	2	3	6	9	–3
10	2	2	6	4	2
11	3	2	6	4	2
12	3	1	6	3	3
13	1	1	1	3	–2
					16

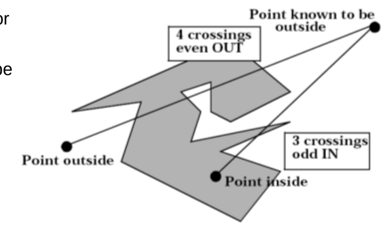
Polygon direction

- Clockwise: Sum is positive
- Counter-clockwise: Negative
- So make holes counter clockwise, include in area calculation, then add



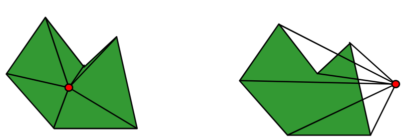
Planar Map Transformations Based on Areas - Point-in-Polygon

- Again, a basic and fundamental test, used in many algorithms.
- For raster mode, use overlay.
- For vector mode, many solutions.
- Most commonly used is the Jordan Arc Theorem
- Tests every segment for line intersection.
- Test point selected to be outside polygon.



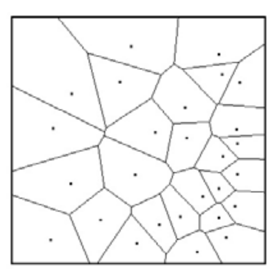
Another Point-in-Polygon Algorithm

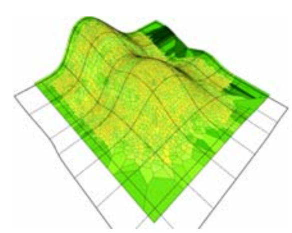
- Calculate area of polygon
- Find test point
- Calculate area for all triangles made by center and two sequential exterior points
- If sum of areas is the area of the polygon, point is inside polygon



Planar Map Transformations Based on Areas - Theissen Polygons

- Often called proximal regions or Voronoi diagrams
- Often used for contouring terrain, climate, interpolation, etc





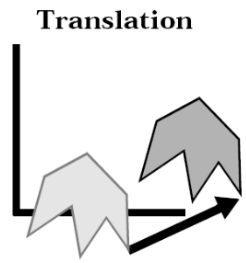
http://en.wiki.mcneel.com/default.aspx/McNeel/PointsetReconstruction.html

Affine Transformations

- These are transformation of the fundamental geometric attributes, i.e. location.
- Influence absolute location, not relative or topological
- Necessary for many operations, e.g. digitizing, scanning, geo-registration, and display
- Affine Transformations take place in three steps (TRS) in order
  - Translation
  - Rotation
  - Scaling

Affine Transformations - Translation

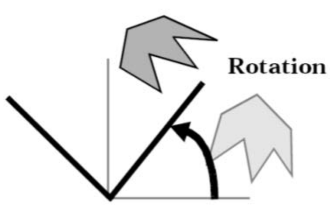
- Movement of the origin between coordinate systems



$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_0 & -y_0 & 1 \end{bmatrix} = \begin{bmatrix} x - x_0 & ,y - y_0 & ,1 \end{bmatrix}$$

Affine Transformations - Rotation

- Rotation of axes by an angle theta

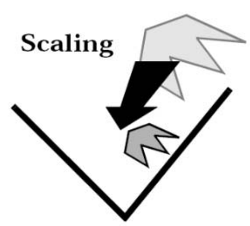


$$\begin{bmatrix} x - x_0 & y - y_0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos\theta(x - x_0) - \sin\theta(y - y_0) & \sin\theta(x - x_0) - \cos\theta(y - y_0) & 1 \end{bmatrix}$$

Affine Transformations - Scaling

- The numbers along the axes are scaled to represent the new space scale

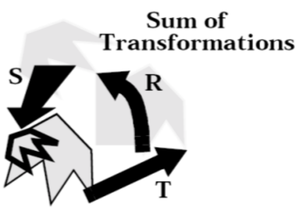


$$\begin{bmatrix} \cos\theta(x - x_0) - \sin\theta(y - y_0) & \sin\theta(x - x_0) - \cos\theta(y - y_0) & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} S_x[\cos\theta(x - x_0) - \sin\theta(y - y_0)] & S_y[\sin\theta(x - x_0) - \cos\theta(y - y_0)] & 1 \end{bmatrix}$$

Affine Transformations

- Possible to use matrix algebra to combine the whole transformation into one matrix multiplication.
- Step must then be applied to every point

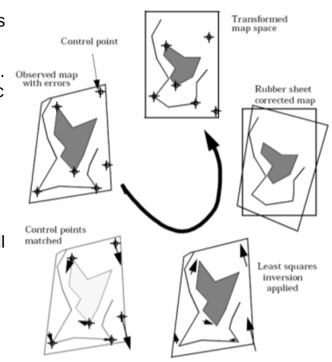


$$\begin{bmatrix} x & y & 1 \end{bmatrix} TRS = \begin{bmatrix} x' & y' & 1 \end{bmatrix}$$

$$x' = s_x[\cos\theta(x - x_0) - \sin\theta(y - y_0)]$$
$$y' = s_y[\sin\theta(x - x_0) + \cos\theta(y - y_0)]$$

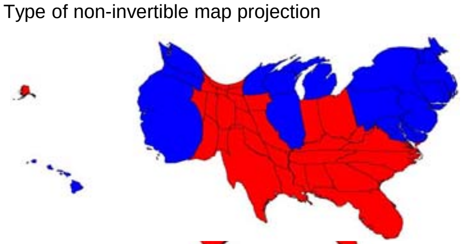
Statistical Space Transformations - Rubber Sheeting

- Select points in two geometries that match
- Suitable points are targets, e.g. road intersections, runways etc
- Use least squares transformation to fit image to map
- Involves tolerance and error distribution
- [x y] = T [u v] then applied to all pixels
- May require resampling to higher or lower density



Statistical Space Transformations - Cartograms

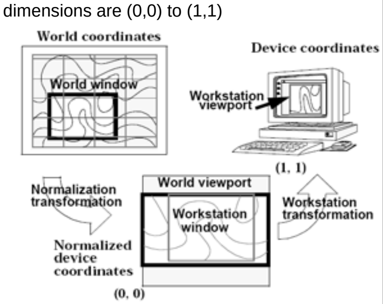
- also known as value-by-area maps and varivalent projections (Tobler, 1986)
- Deliberate distortion of geometry to new "space"
- Type of non-invertible map projection



http://tabacco.blog-city.com/red_vs_blue_big_lie_maps__cartograms_of_2004_presidential_el.htm

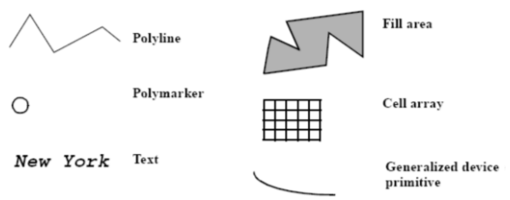
Symbolization Transformations

- Screen coordinates are often reduced to a "standard" device – Normalization Transformation
- Standard Device display dimensions are (0,0) to (1,1)
- World Coordinates-> Normalized Device Coordinates > Device Coordinates

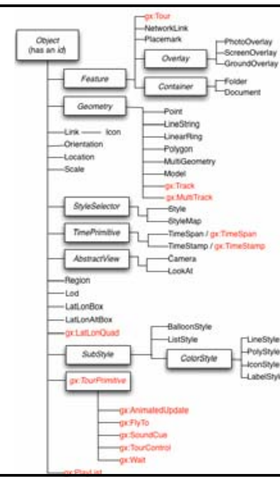


Drawing Objects: Primitives

- Most use model of primitives and attributes
- The Graphical Kernel System (GKS) has six primitives, each has multiple attributes.



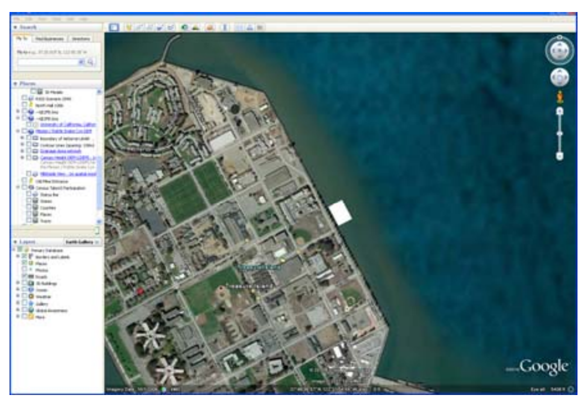
GoogleEarth KML Objects



For example, a LinearRing

- <?xml version="1.0" encoding="UTF-8"?>
  <kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
  <name>LinearRing.kml</name>
  <Polygon>
  <outerBoundaryIs>
  <LinearRing>
  <coordinates> -122.365662,37.826988,0 -122.365202,37.826302,0 -122.364581,37.82655,0 -122.365038,37.827237,0 -122.365662,37.826988,0 </coordinates>
  </LinearRing>
  </outerBoundaryIs>
  </Polygon>
  </Placemark>
  </kml>

Launch GE, bring in KML file



Summary

- Geometry can be crisp or vague, raster or vector
- Operations can work even when information is vague
- Algorithms can apply basic transformations for essential measurements, such as clustering, length, area, density
- Basic features can be ingested into simple programming language data structures
- Functions then can measure length, mean center, dispersal, area
- Can also transform geometry in non-invertible ways, e.g. cartograms
- Last transformation is into normalized device coordinates, then device/viewport coordinates
- Gave examples of simple objects and their rendering objects