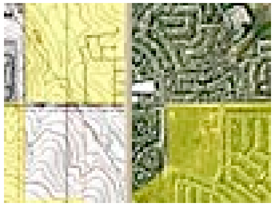
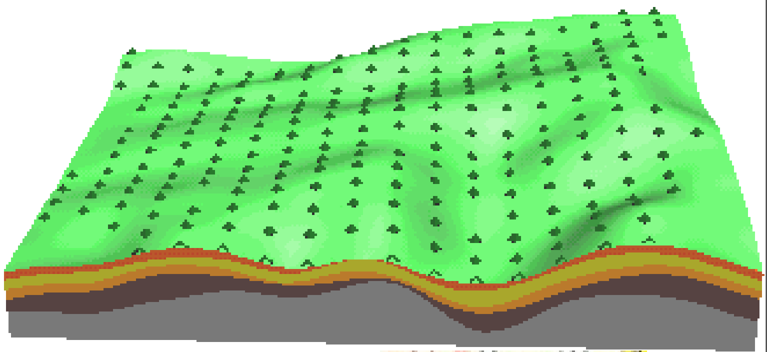
CSci 350: A Computing Perspective on GIS

RDBMS and Spatial data

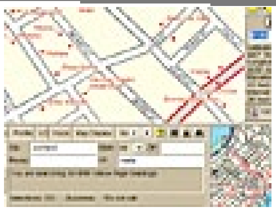
Data

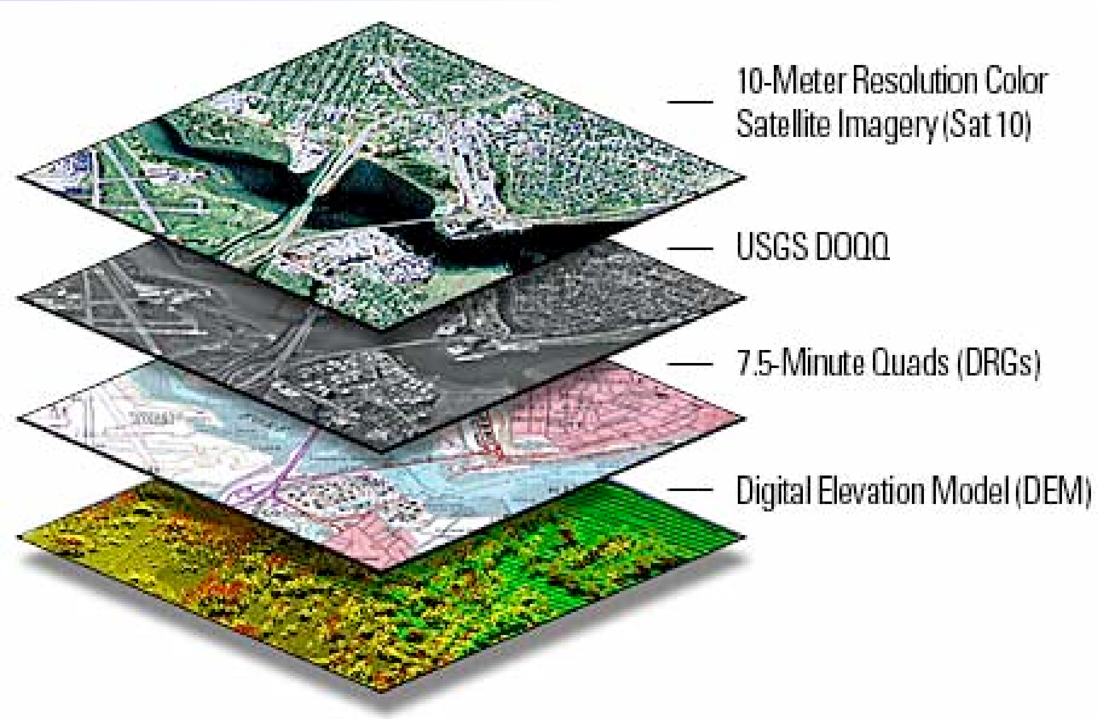


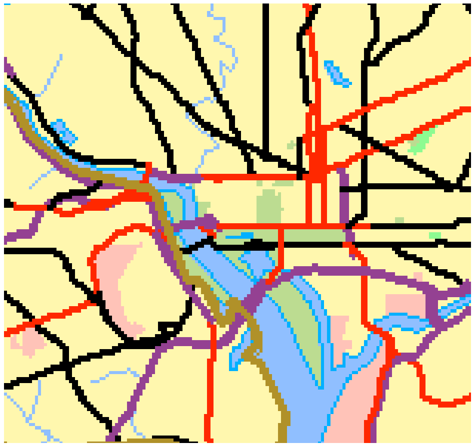
- GIS deals with geospatial data



- Huge amounts of digital geospatial data available
  - from remote sensing, satellites, aerial photogr
  - from existing cartographic maps







GIS data

- GIS data handles geographic objects
  - data has descriptive attribute + spatial component
- organized into "themes" (layers)
  - a theme consists of objects of the same type
  - e.g. river theme, road theme, etc
  - each theme has a schema and instances
  - e.g. theme city
    - city = { name, population, location}
  - theme country
    - country = {name, capital, population, region}
  - theme language
    - language = {language, region}
- operations on themes
  - projection
  - selection
  - union
  - overlay
  - geometric selection

- overlay of spatial data (spatial join)
  - T1 X T2
  - essentially computes the intersections of the two themes. It produces a new theme where each object in the intersected map is labeled with its attributes from both themes
  - e.g. country X language
- Questions:
  - size of output?
  - how to compute intersections?
  - efficiency?...

- geometric selection
  - window query
  - point query
  - [others]
- other theme operations
  - topological: what countries are adjacent to belgium?
  - geometric: what’s the distance paris-berlin?
  - interpolation: estimate an attribute at a given point

Storing spatial data

- At the beginning, GIS were built directly on top of the file systems.
  - data is stored in files, controlled by the application
  - problems with security, concurrency, etc

- Store spatial data in a DBMS
- RDBMS suitable for spatial data?
  - not too flexible (hard to define spatial types)
  - no data independence (formulation queries requires knowledge of how data is stored)
  - efficiency is questionable
  - how to express geometric/topologic computations with relations?
    - e.g. adjacency test, or point query
  - indexing structures not appropriate

Relational model and spatial data

- Structure
  - relational tables may be awkward for storing spatial data
  - e.g. imagine storing the segments that form the boundary of a polygon
- Indexes
  - relational DB provide indexing structures that work well with standard tabular data
    - e.g. to provide fast accesses to movie titles, RDMS keeps FILM in a balanced search tree ordered by title
    - BST: insert, delete, search fast
  - spatial data requires specialized indices
    - standard RDMS indexes are not efficient
- Performance
  - spatial data requires many types of joins, which are expensive
  - difficult to achieve good performance with generic join technology
  - need specialized algorithms that work on geometric data

Indexing

- Indexing 1D data
  - Input: A set of n 1D-points S = { x1, x2, x3, ... xn}
  - Store S in a structure to answer efficiently the following types of questions
    - search (x): does point x exist in S
    - nearestNeighbor(x): return the nearest neighbor of point x in S
    - range(a, b): return all the points in S that fall between a and b

- Indexing 2D data
  - Input: A set of n 2D-points S = { p=(x,y) }
  - Store S in a structure to answer efficiently the following types of questions
    - search (p): does point p = (x,y) exist in S
    - nearestNeighbor(p): return the nearest neighbor of point p in S
    - range(x1, x2, y1, y2): return all the points in S that fall in $[x_1 \times x_2] \times [y_1 \times y_2]$

Storing spatial data

- Loosely coupled approach
  - separate DBMS from spatial data
  - have a specific module that handles spatial data
  - e.g. ArcInfo

- Integrated approach
  - build an extension on top of DBMS that handles spatial data
  - many traditional DBMS started to offer a spatial extension
  - e.g. Oracle 8i, Postgres
  - extend SQL to manipulate spatial data
  - adapt DBMS functionality to handle spatial data

Requirements from a spatial DBMS

- Integrate spatial data at the logical level while satisfying data independence
- Integrate new functionality into SQL to capture geometric data
- An efficient physical representation of data
- Efficient indexing structures for spatial data and efficient algorithms.