9. Databases

Motivation:

Computers are often used

- for dealing with **large amounts of data**
- and in situations where **data integrity** is important for the survival of an organization.

Examples:

- Banking
- e-commerce (commercial transactions via WWW – e.g., **`amazon.com`** or **`ebay.com`**)
- meteorological measurements
- booking systems (trains, airlines...)
- ERP systems (Enterprise resource planning)
- telecommunication (phone numbers, fax numbers, mobile phone data...)

Main problems:

- How can large amounts of data be organized so that they can be accessed quickly?
- How can data be organized so that hardware and software failures do not lead to a desaster?
- How can data be changed by several agents in parallel without interference?

Today these problems are being dealt with on the conceptual basis of **relational database management systems** (RDBMS), typically using some dialect of **SQL** (structured query language) as notation for definition and manipulation of data.

In these slides: Only very basic concepts are discussed.

Introduction using an example

Simplistic example: public library. Data organized in **tables**.

- table “Users” with columns `UserID, Name, Address, BirthDate`
- table “Books” with columns `BookID, Title, Author, Keywords`
- table “BorrowedBooks” with columns `UserID, BookID, BorrowedSince, BorrowedUntil`

Principles of database tables

- Relational databases hold the data in (typically several) **tables**.
- Each row represents one **record**.
- The number and meanings of the **columns** of a table is (more or less) **fixed**.
- The number of **rows** of a table is **variable**.

"Entity relationship model":

- Each table describes one kind of **entities** or a **relation** (typically between several entities)
- a model of a certain part of reality based on the concepts of entities and their relationships is called an entity-relationship model.

In our example: tables "Books", "Users" represent entities,
table "BorrowedBooks" represents a relation between these entities.

Attributes, key candidates and keys

Columns in a table are called **attributes**. Some attributes or attribute combinations **characterize** entities. Such attributes or attribute combinations are **key candidates**. One of the key candidates is designated as **primary key**. The primary key of an entity is used in order to refer to it from other entities or from relations.

In our example, `UserID` is used as primary key in the “Users” table, and `BookID` is used as primary key in the “Books” table. These attributes are used in “BorrowedBooks” in order to refer to the related entities.

Data definition and data manipulation with SQL

Two kinds of languages for working with relational data bases are distinguished:

data definition language (DDL)

data manipulation language (DML)

DDL and DML are today typically combined in dialects of SQL (structured query language) and supported by producers of database management systems. The different dialects are based on similar principles. We will give examples.

Data definition consists in the definition of the structure or tables and their interrelations.

During data definition, it must be defined for each table:

- which **attributes** it contains,
- how each attribute is to be **represented** (a data type must be chosen),
- which attributes form the **primary key** of the table, and
- which attributes refer as **keys** to other tables.

A notation which allows to define tables in this way is called a **data definition language** (DDL).

Data manipulation consists in adding, changing and deleting table rows and in the selection of data from the data base.

A DDL only alows to describe the structure of a data base, not to change its content in any way.

A notation which allows to manipulate tables is called a **data manipulation language** (DML).

Data definition

The “Users” table from the public library example could be defined like this:

```
CREATE TABLE Users (
  UserID      INT(10) NOT NULL,
  Name        CHAR(100),
  Address     CHAR(100),
  Birthdate   DATE,
  PRIMARY KEY (UserID)
)
```

This instruction creates a table names “Users” with the four already described columns. `UserID` is represented a ten-digit decimal number, `Name` and `Address` are represented as 100 characters, `Birthdate` as a date, and `UserID` is the primary key of the table.

For `UserID`, a value must be given for each row in the table – for the other three columns, a standard value (`NULL`) might be used in order to designate that the value of the attribute is not known.

The table “Books” might be defined similarly, only the attribute `Keywords` presents problems. Which amount of memory should we reserve for the keywords of a book if we do not want to restrict the number of keywords beforehand?

One solution consists in the definition of an extra table “Keywords”:

```
CREATE TABLE Keywords (
  BookID  INT(10),
  Keyword CHAR(100)
)
```

Key words have a maximal length of 100 characters, but the number of key words which can be given for a book is not restricted, since the same book can occur any number of times in the table.

The “Books” table could be declared like this:

```
CREATE TABLE Books (
  BookID INT(10) NOT NULL,
  Title  CHAR(100),
  Author CHAR(100),
  PRIMARY KEY (BookID)
)
```

The table representing currently borrowed books might be declared like this:

```
CREATE TABLE BorrowedBooks (
  UserID      INT(10),
  BookID      INT(10),
  BorrowedSince DATE,
  BorrowedUntil DATE
)
```

Data manipulation

The following operations can be used to manipulate the data in the tables:

- The **SELECT** command selects information from the data base.
- The **INSERT** command inserts rows into a table.
- The **UPDATE** command changes the content of existing rows in a table.
- The **DELETE** command removes rows from a table.

SELECT

The list of overdue books can be determined as follows:

```
SELECT b.BookID, b.Author, b.Title, l.BorrowedSince
FROM Books AS b, BorrowedBooks AS l
WHERE     b.BookID = l.BookID
      AND l.BorrowedUntil < TODAY
```

This statement is also called a **query** (the data base system is queried for some data).

This query returns a **table with four columns**. Each row represents an overdue book; the first column contains the book id, the second the author, the third the book title, and the last column the date when the book was borrowed.

A query has the following form:

- After the keyword **FROM**, the tables are listed from which data is to be collected. We use all combinations of rows from "Books" and "BorrowedBooks", and we abbreviate "Books" as "b" and "BorrowedBooks" as "l" elsewhere in the query.

- The **WHERE** keyword defines a filter: only those combination of rows from the **FROM** clause are kept which fulfill the condition given behind the **WHERE**: The book ids of the two entries must match, and the date until which the book must be given back must lie in the past.

- The **SELECT** keyword introduces a list of expressions which are evaluated for each row combination filtered out by the **WHERE**. In the example, these are simply some of the attributes.

Tasks: (a) Change the query so that in addition to the overdue book, the result also contains the person who has borrowed the book? (b) Which query determines which books have to be given back in the next two weeks? Assume that from addition of a date and a number, a date results which lies the given number of days after the given date.

INSERT

When a book is borrowed, a row has to be added to table `BorrowedBooks`. The following instruction adds a row with `UserID` `1053465`, `43565` as `BookID`, `TODAY` as `BorrowedSince` and `TODAY+14` as `BorrowedUntil`. The order of the arguments is the same as the order of the columns in the table declaration.

```
INSERT INTO BorrowedBooks
VALUES (1053465, 43565, TODAY, TODAY+14)
```

The general form is the following: After the keywords `INSERT INTO` and the name of the table, the keyword `VALUES` starts a list of values representing the row to be inserted.

UPDATE

In order to lengthen the borrowing time of the book with id 43565 by a week, the following command could be executed:

```
UPDATE BorrowedBooks
SET BorrowedUntil = BorrowedUntil + 7
WHERE BookID = 43565
```

After `UPDATE`, the name of the table to be changed is given. The `WHERE` predicate defines which rows are affected by the change, and after `SET` it is defined which columns in the rows to be changed are updated, and to which value.

DELETE

When a book is brough back by a used, its entry has to be taken out of the “BorrowedBooks” table:

```
DELETE FROM BorrowedBooks
WHERE BookID = 43565
```

Further elements of the SQL language

Above we have only seen the most elementary SQL language elements. Many SQL dialects present many more features. Examples:

- **Integrity constraints** can be used in order to define *conditions on the content* of a database which shall never be violated during manipulations.
- **Foreign key relations** are used in order to make explicit that values in a column are keys of some other table. They are a special case of integrity constraints.
- **Index declarations** are used in order to accelerate searching in tables.
- **Stored procedures** are used in order to store instructions which are to be executed by the database.
- Further **table operations**: *set union*, *set difference*, *set intersection*, *grouping* of results, *sorting* of results.
- **Views** allow to shield the users of a database from the internal representation of the data.
- **Database administration** consists in deciding how tables etc. are represented and which users get which kind of access to the database.
- **Invariants** and **triggers** are language elements which ensure the fulfillment of integrity constraints independently of the application programme.
- **Transactions** are language elements which ensure that a sequence of changes is either executed *completely* or *not at all*, even in the case of hardware or software failures.

Conceptual database design

The **conceptual design** of a relational database often proceeds according to the following steps:

- First the **entities** relevant in the application area are collected and their types are determined. (Example: books, users)
- Then the relevant **relationships** between entities are determined. (Example: BorrowedBooks)
- For each entity type and each relationship type, the **attributes** and their data types are determined.
- Finally, **integrity conditions** for the database are specified. (Example: BorrowedUntil must not be earlier than BorrowedSince)

On the basis of this design it is decided how entities, relationships, attributes are represented in a specific database management system.

Normalization:

Redundant data in a data base might lead to **inefficiencies** and **inconsistencies**: Updates of redundantly held information have to be performed at several locations instead of at only one, and if this is forgotten, an inconsistency results.

Normalisation of a database consists in the reduction of redundancies, typically via splitting tables.

Architecture of database applications

Database applications often use a *three-layer* architecture:

- A **DBMS** operates as the kernel of the system. It ensures data persistency, data integrity etc.

- An **application layer** provides application-specific functionality. In our example, it would provide the functions “borrow a book”, “lengthen borrowing time”, “register new user” etc.

- A **presentation layer** defines the user interface, which today is often graphical, and not seldom with an alternative using the WWW.

These three components might run as **three different programs** on different computers: A **web-browser** runs the presentation layer, the web-server dispatches the user input to an **application program**, and the application program accesses a **relational database** on a dedicated database server.

Geographical Information Systems

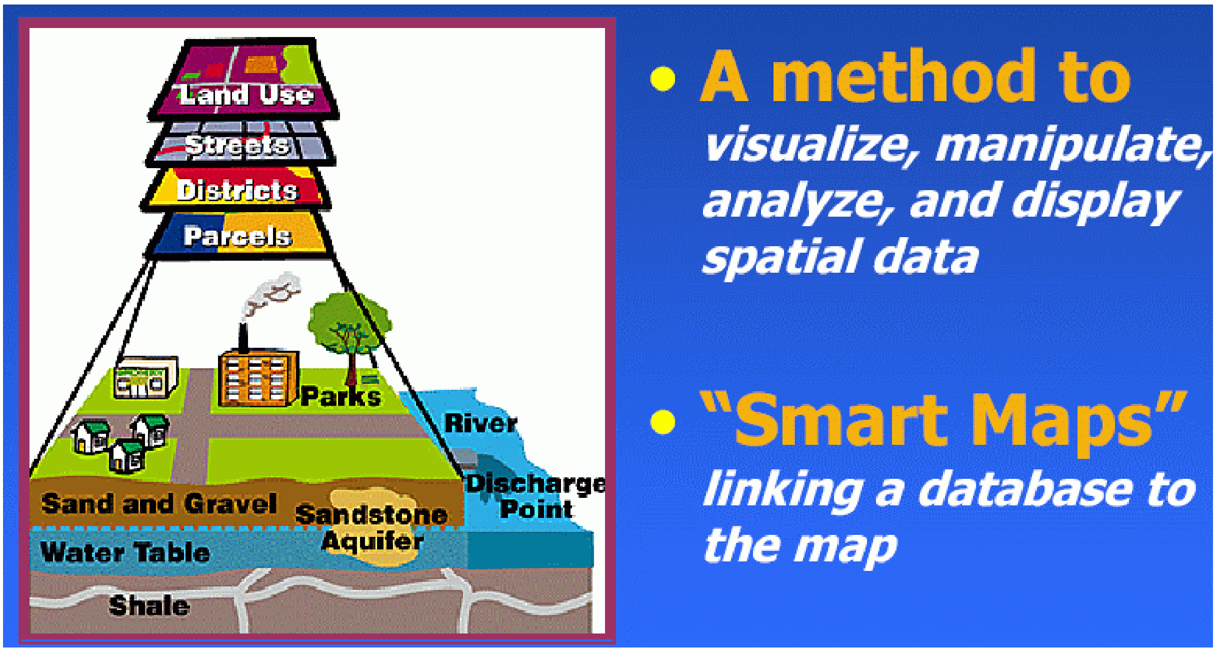
What is a Geographical Information System (GIS)?

- Software, hardware and data to help manipulate, analyse and present information that is tied to *spatial locations* (usually geographical locations).

Estimates are that 80 % of all data stored worldwide has a *spatial* component (Source: www.gis.com).

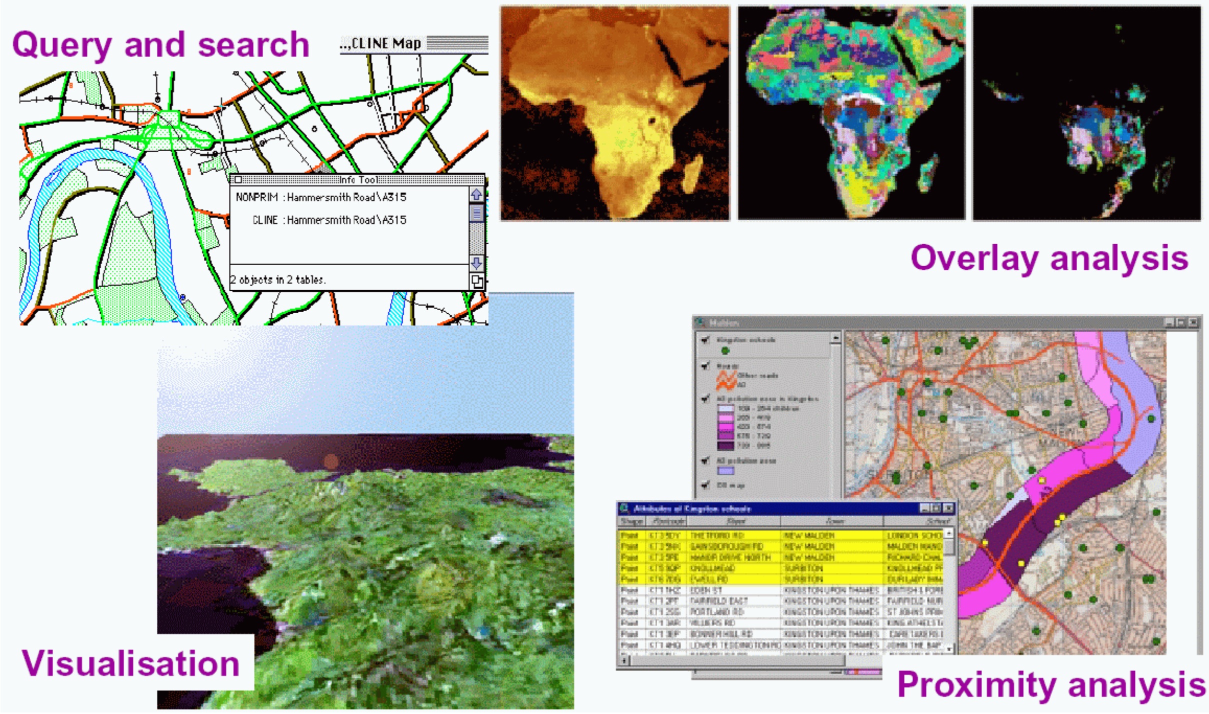
A GIS contains a classical database, but extends its functionality by methods adapted to spatial information.

Particularly, a GIS provides...



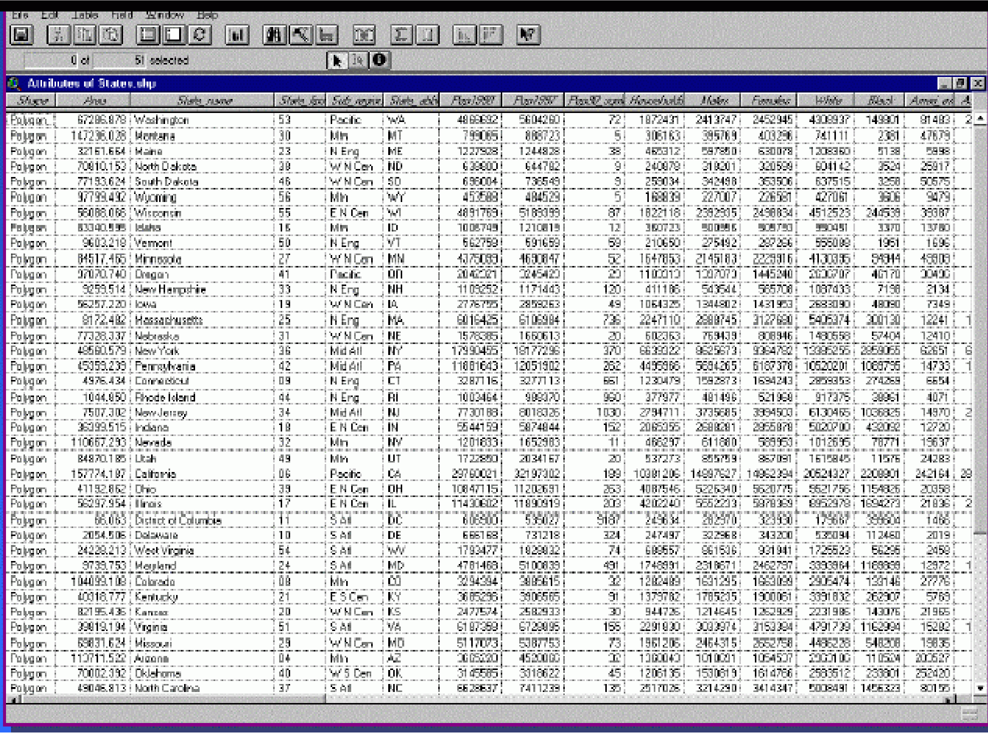
- special forms of *query*, designed to extract information with spatial properties from a database (e.g., taking neighbourhoods into account)
- special forms of *data analysis* (e.g., geostatistics)
- special forms of *integrity checking* adapted to spatial data.

What can a GIS do?

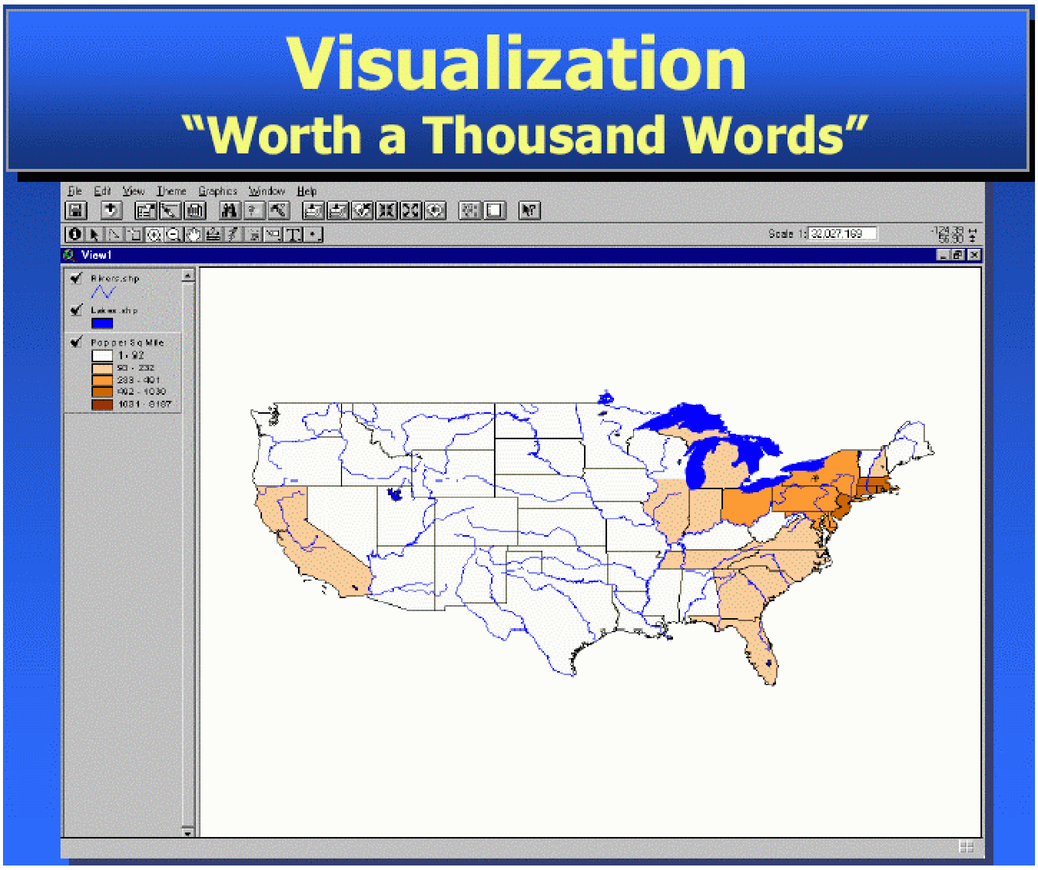


Database

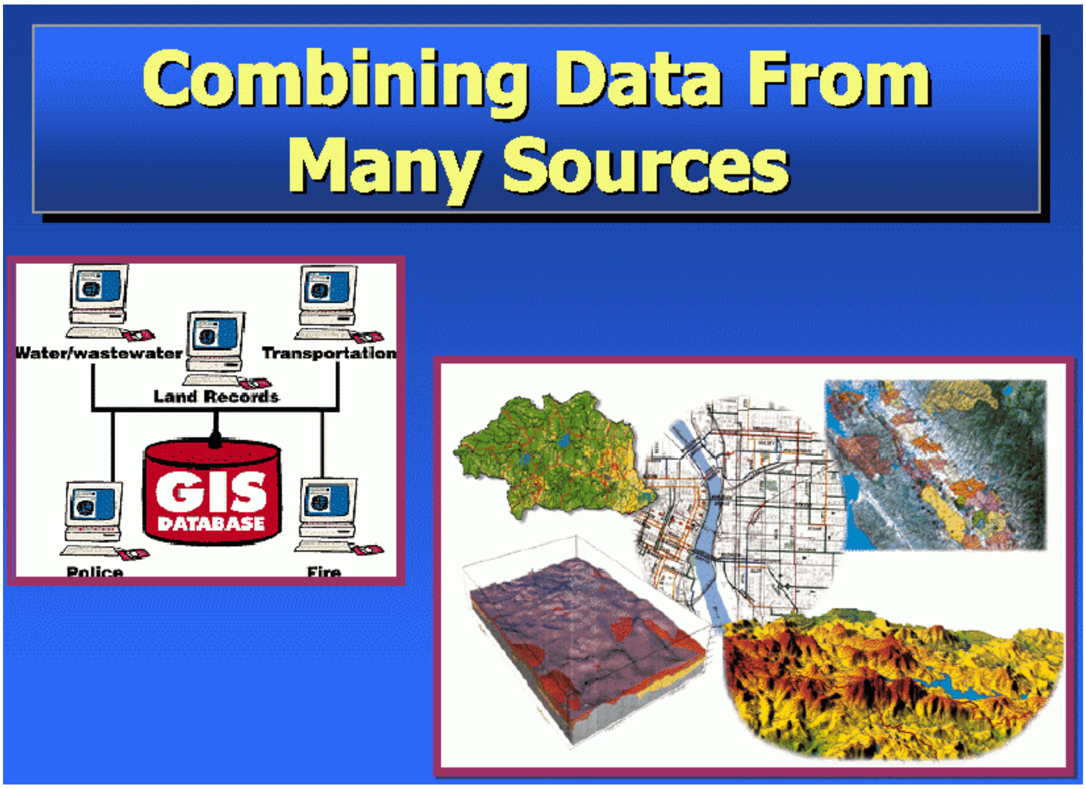
"Not Easy to Interpret"

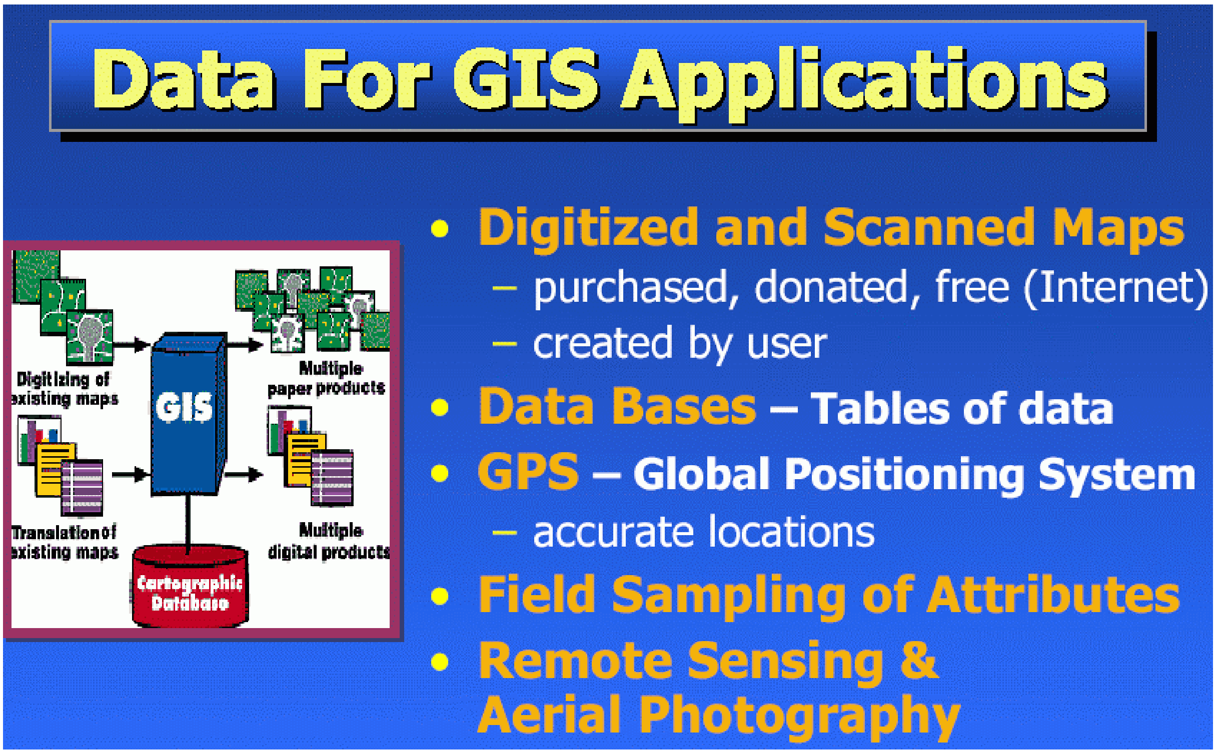


Shape	Area	State_name	State_fips	Sub_region	State_abbr	Pop1990	Pop1997	Pop90_sqmi	Households	Males	Females	White	Black	Ameri_es	A
Polygon	67286.878	Washington	53	Pacific	WA	4866692	5604260	72	1872431	2413747	2452945	4308937	149801	81483	2
Polygon	147236.028	Montana	30	Mtn	MT	799065	888723	5	306163	395769	403296	741111	2381	47679	
Polygon	32161.664	Maine	23	N Eng	ME	1227928	1244828	38	465312	597850	630078	1208360	5138	5998	
Polygon	70810.153	North Dakota	38	W N Cen	ND	638800	644782	9	240878	318201	320599	604142	3524	25917	
Polygon	77193.624	South Dakota	46	W N Cen	SD	696004	736549	9	259034	342498	353506	637515	3258	50575	
Polygon	97799.492	Wyoming	56	Mtn	WY	453588	484529	5	168839	227007	226581	427061	3606	9479	
Polygon	56088.066	Wisconsin	55	E N Cen	WI	4891769	5169099	87	1822118	2392935	2498834	4512523	244539	39387	
Polygon	83340.598	Idaho	16	Mtn	ID	1006749	1210819	12	360723	500956	505790	950451	3370	13780	
Polygon	9603.218	Vermont	50	N Eng	VT	562758	591659	59	210650	275492	287266	555088	1951	1696	
Polygon	84517.465	Minnesota	27	W N Cen	MN	4375099	4690847	52	1647853	2145183	2229916	4130395	94944	49909	
Polygon	97070.740	Oregon	41	Pacific	OR	2842321	3245420	29	1103313	1397073	1445248	2636787	46178	38496	
Polygon	9259.514	New Hampshire	33	N Eng	NH	1109252	1171443	120	411186	543544	565708	1087433	7198	2134	
Polygon	56257.220	Iowa	19	W N Cen	IA	2776755	2859263	49	1064325	1344802	1431953	2683090	48090	7349	
Polygon	8172.482	Massachusetts	25	N Eng	MA	6016425	6106984	736	2247110	2888745	3127680	5405374	300130	12241	1
Polygon	77328.337	Nebraska	31	W N Cen	NE	1578385	1660613	20	602363	769439	808946	1480558	57404	12410	
Polygon	49560.579	New York	36	Mid Atl	NY	17990455	18177296	370	6639322	8625673	9364782	13385255	2859055	62651	6
Polygon	45359.239	Pennsylvania	42	Mid Atl	PA	11881643	12051902	262	4495966	5694265	6187378	10520201	1089795	14733	1
Polygon	4976.434	Connecticut	09	N Eng	CT	3287116	3277113	661	1230479	1592873	1694243	2859353	274269	6654	
Polygon	1044.850	Rhode Island	44	N Eng	RI	1003464	989370	960	377977	481496	521968	917375	38861	4071	
Polygon	7507.302	New Jersey	34	Mid Atl	NJ	7730188	8018326	1030	2794711	3735685	3994503	6130465	1036825	14970	2
Polygon	36399.515	Indiana	18	E N Cen	IN	5544159	5874844	152	2065355	2688281	2855878	5020700	432092	12720	
Polygon	110667.293	Nevada	32	Mtn	NV	1201833	1652983	11	466297	611880	589953	1012695	78771	19637	
Polygon	84870.185	Utah	49	Mtn	UT	1722850	2034167	20	537273	855759	867091	1615845	11576	24283	
Polygon	157774.187	California	06	Pacific	CA	29760021	32197302	189	10381206	14897627	14862394	20524327	2208801	242164	28
Polygon	41192.862	Ohio	39	E N Cen	OH	10847115	11202691	263	4087546	5226340	5620775	9521756	1154826	20358	
Polygon	56297.954	Illinois	17	E N Cen	IL	11430602	11890919	203	4202240	5552233	5878369	8952978	1694273	21836	2
Polygon	66.063	District of Columbia	11	S Atl	DC	606900	535027	9187	249634	282970	323930	179667	399604	1466	
Polygon	2054.506	Delaware	10	S Atl	DE	666168	731218	324	247497	322968	343200	535094	112460	2019	
Polygon	24228.213	West Virginia	54	S Atl	WV	1793477	1829832	74	688557	861536	931941	1725523	56295	2458	
Polygon	9739.753	Maryland	24	S Atl	MD	4781468	5100839	491	1748991	2318671	2462797	3393964	1189899	12972	1
Polygon	104093.108	Colorado	08	Mtn	CO	3294394	3885615	32	1282489	1631295	1663099	2905474	133146	27776	
Polygon	40318.777	Kentucky	21	E S Cen	KY	3685296	3906585	91	1379782	1785235	1900061	3391832	262907	5769	
Polygon	82195.436	Kansas	20	W N Cen	KS	2477574	2582933	30	944726	1214645	1262929	2231986	143076	21965	
Polygon	39819.194	Virginia	51	S Atl	VA	6187358	6729895	155	2291830	3033974	3153384	4791739	1162994	15282	1
Polygon	69831.624	Missouri	29	W N Cen	MO	5117073	5387753	73	1961206	2464315	2652758	4486228	548208	19835	
Polygon	113711.522	Arizona	04	Mtn	AZ	3665228	4520000	32	1368843	1810691	1854537	2963186	110524	203527	
Polygon	70002.392	Oklahoma	40	W S Cen	OK	3145585	3318622	45	1206135	1530819	1614766	2583512	233801	252420	
Polygon	49046.813	North Carolina	37	S Atl	NC	6628637	7411239	135	2517026	3214290	3414347	5008491	1456323	80155	

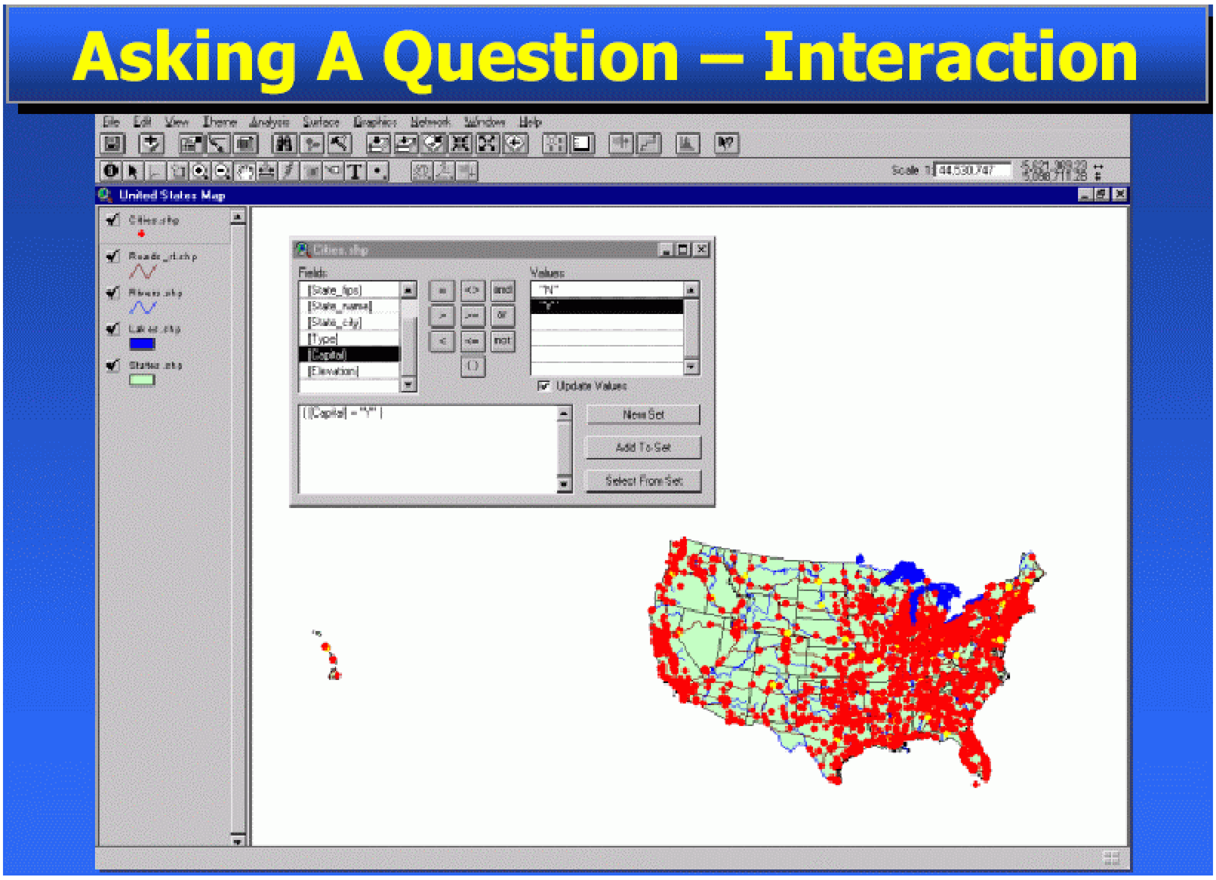


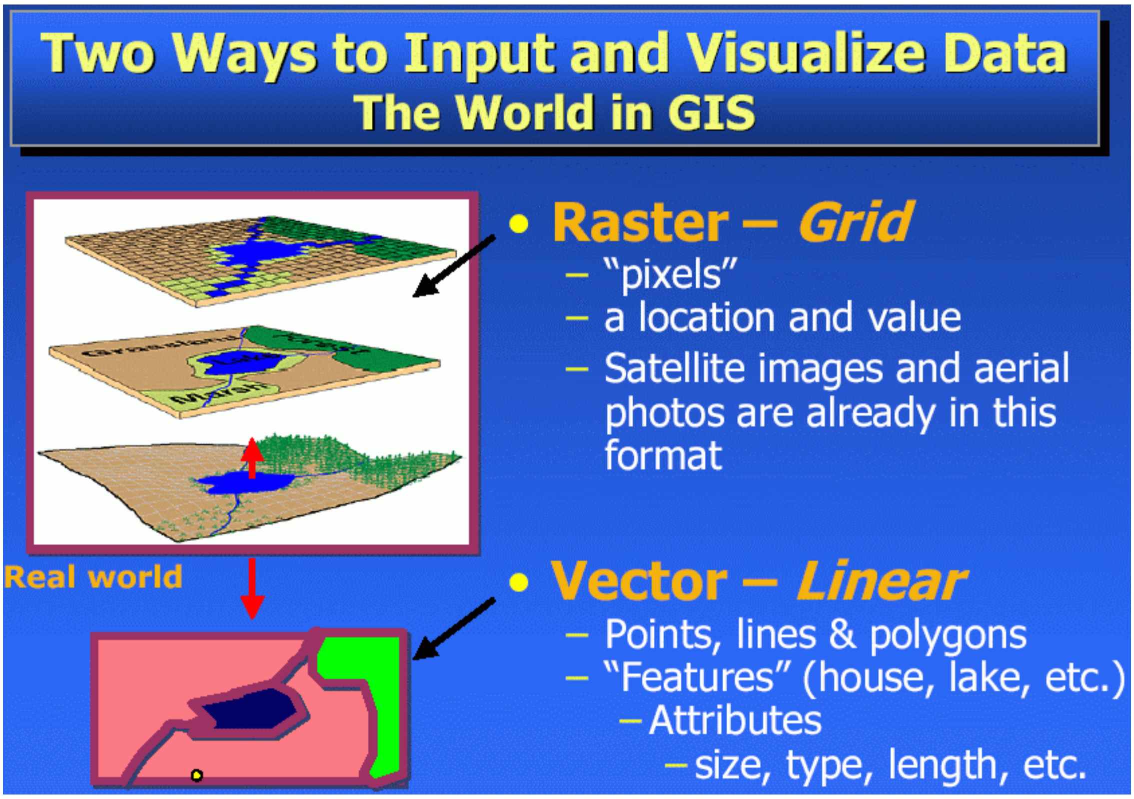
One of the main advantages of GIS over classical geographical maps:





Further advantage: Easy interaction, visualization, manipulation of maps

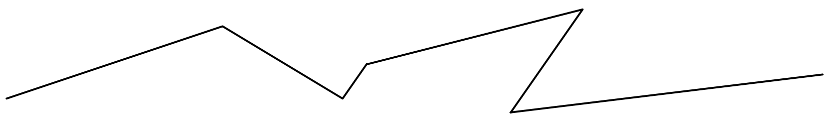




The vector representation is more appropriate for senseful queries (and is more exact)
– basis for relational database representation of geographical data

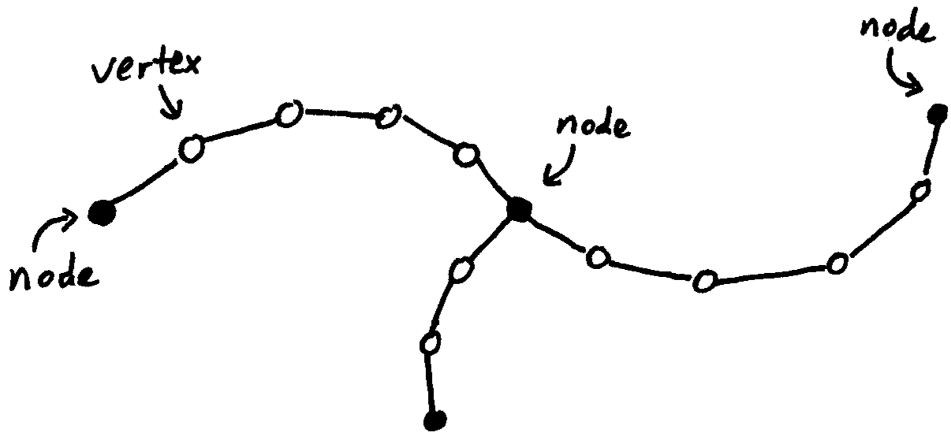
Typical *entities* of a GIS:

- *Points*
- *Tics* (= special points for which the exact real-world coordinates are known, used to fit a digital map into a global coordinate system)
- *Lines*, also called *arcs* (more precisely: Multilines, i.e. consisting of several linear segments)



- *Polygons* (closed multilines, possibly with additional attributes)
- *Annotations* (text objects associated with points).

The endpoints of a line (and possible branching points) are called *nodes*.
Intermediate points (without branching) are called *vertices*.



Tables in the underlying relational database:

- Tic table
- boundary table (represents the spatial extent of the map – a surrounding rectangle)
- arc attribute table (AAT)
- polygon attribute table (PAT).

E.g., a *polygon* is represented as a line in the PAT, with attributes: polygon ID, nodes, arcs, a label point (in the interior), further attributes (e.g., area, slope, population density...). Details differ between different GIS.

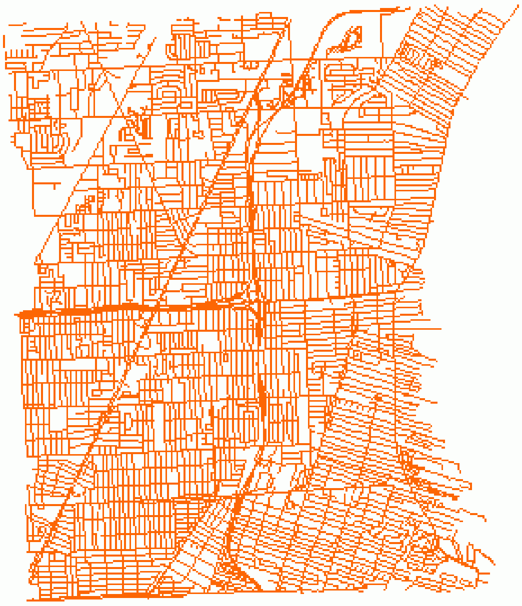
Usually, a GIS does not only contain information for a single map of a region, but *several sorts of information for the same region*:
each sort of information is represented in an extra *coverage* (also called *layer*, *cover* or *theme*).

Example: Different coverages of a town area:

Road Centerline/Address (Geographic Township)

- source: 2000 Digital Orthophotography

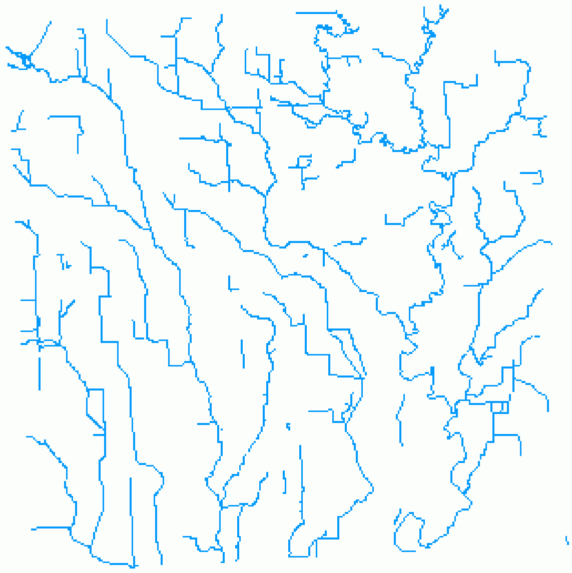
- Road Arcs
- Address Ranges
- Road Names



Hydrology (Geographic Township)

- source: 2000 Digital Orthophotography

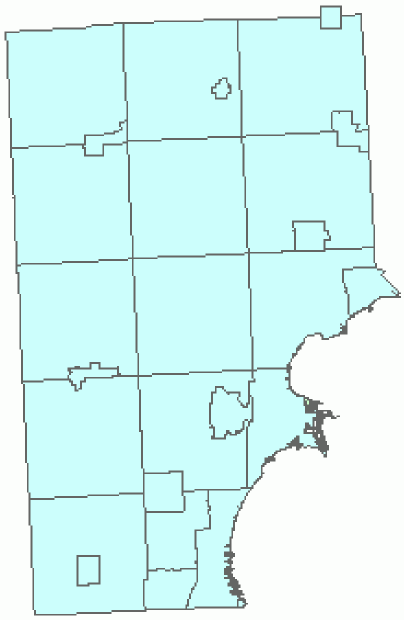
- Water feature arcs (rivers, streams, drains)
- Water feature polygons (lakes, ponds, retention basins)
- Names of County maintained drains



Municipal Boundaries (County)

- source: 2000 Digital Orthophotography

- Community polygons
- Community name

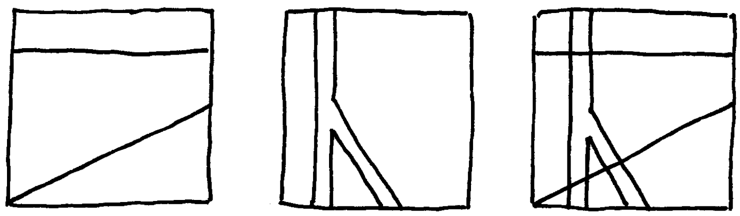


(source: `http://macombcountymi.gov/gis/gis_coverage_samples.htm`)

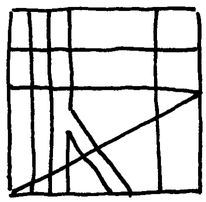
How to combine several coverages?

- *Overlay operation*

From two geometries, the GIS calculates the coarsest common geometry:



a *b* overlay of *a* and *b*

Attention: The following geometry  would also be a common geometry of *a* and *b*, but not the coarsest one!

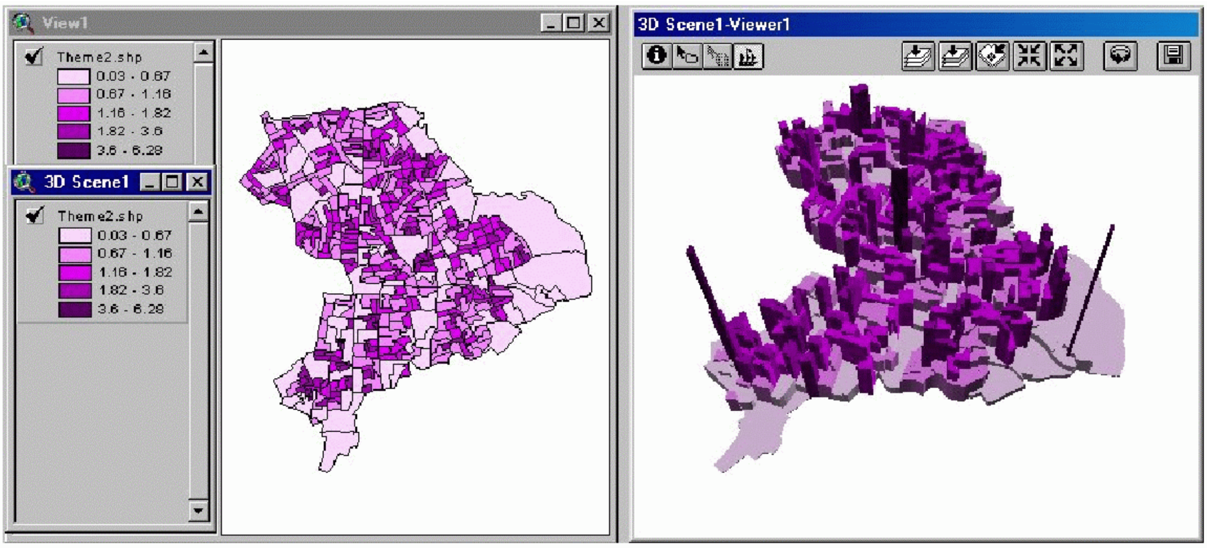
Using overlay, a GIS can give answer to questions like:
"What forest areas of district *x* are within 100 m distance to a road, are stocked with conifers and have a slope < 5 degrees?" (e.g., for a chalking action)

Layers used for this task:

- landuse map (→ forests)
- political district map (→ district *x*)
- road map (→ 100 m neighbourhood to a road)
- forest type map (→ stocked with conifers)
- digital elevation model (→ slope < 5 degrees)

Selection of polygons of the overlay using an "and" operation

Further functionality of GIS: 3D visualization



[redacted] *Representing Attribute Data in 3-D: Population Density in Small Census Areas in the London Borough of Hackney*

Widely used GIS products:

- ESRI ArcView, ArcGIS (licenced commercial software)
- FreeGIS (open source, **`www.freegis.org`**)