Chapter 1

Introduction

In recent years the commercial importance of spatial information has been increasing consistently. That is due to the many interesting conclusions that can be drawn from the evaluation of information with spatial content. These conclusions may lead to a considerable performance improvement for several businesses. Since company managers have to make important decisions quickly, spatial data will only be taken into account if the important information is available immediately. At the technical level this requires efficient storage and retrieval of spatial information.

Research has been concerned with efficient retrieval of spatial information for quite a while. This has led to many proposals on how to store spatial data in order to be able to retrieve it efficiently. An important point to note is, that the efficiency of any storing method strongly depends on the types of queries that will be used to retrieve it later on. Consequently different storing methods will perform differently on varying queries. Moreover most methods were designed and evaluated as file-based storing methods. Since nowadays (object-)relational databases are used, that support convenient and hopefully efficient data access, the applicability of the proposed methods for current database management systems has to be investigated. This is especially true in the commercial setting where spatial information is only valuable in conjunction with domain-specific data that has been stored in database systems for several years.

As a more concrete motivating example consider the following problem: the secretary of the Tippecanoe county council is preparing a proposal for next years budget plan. In order to be able to estimate the cost for road maintenance she needs to know the total length of all roads in Tippecanoe county. She is provided with a map of all roads and zip code polygons in the state of Indiana (figure 1.1), as a visualization of the contents of some database tables containing that information. She is now confronted with the problem of retrieving all roads that run through Tippecanoe county (marked in red in figure 1.1). More generally she needs to determine all roads from a given table that lie within a given rectangle (or window). This is the case of a *spatial selection* query (or window query). The importance of support for efficient window querying is obvious, if we consider that this query without any index takes 1480 seconds or about 25 minutes

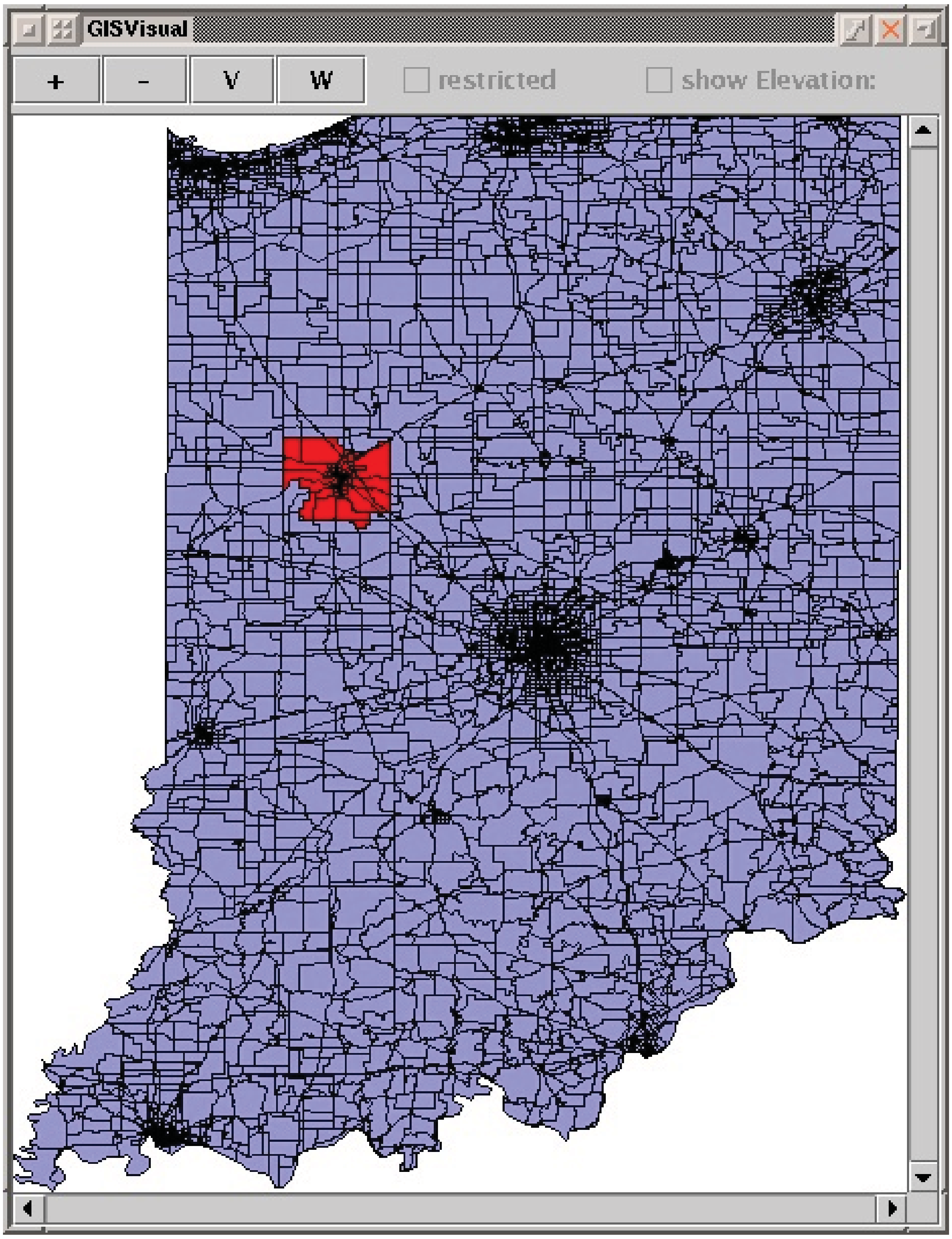

Figure 1.1: *Map of zip code polygons and roads in Indiana*

on the database server used throughout this work[1]. This is clearly not acceptable. By using insights gained from this work the time can be reduced to about 82 seconds for selecting the roads required[2]. Additional time is needed for computing the total length, but this time is in the range of few seconds also. This example shows the importance of efficient spatial selections.

On the next day additional costs have to be added for all bridges to be maintained in Tippecanoe county. The secretary has an additional database table of all rivers in Indiana. Suppose for simplicity that smaller tables of all roads and rivers in Tippecanoe county have been generated by the council's database programmer after the problems caused on the previous day (visualized in figure 1.2). Still the secretary cannot obtain a list or the total length of all the road bridges in the county from just seeing them on the screen. Even by zooming to more interesting parts (see figure 1.3) of the map a pure visualization is not sufficient since no electronic processing of the results is possible. She needs to select all road objects from the database that intersect *any* river in the database and obtain the result in electronically processable form. Required is a *spatial join* query since spatial information from two tables has to be joined with regard to their spatial properties in order to compute the answer. This is a very computationally intensive task which, even if implemented optimally on the current database system, takes time in the range of a few hours if executed on the entire Indiana state dataset. This work will describe a foundation which should simplify the execution of such complex queries by using user-defined extensions to recent database systems.

Another task to be solved by the secretary could be to determine all roads in the county that have a length between 500 yards and 1500 yards since there is a special state-wide support program for such streets. With current methods one can either spatially index the geometry and then let the system check all retrieved roads for their length or index the length of the roads and then let the system check which of the roads of the required length lie in Tippecanoe county. Each option is suboptimal, and this work develops the basis[3] for a better way by indexing geometry and domain-specific information (length in the example but in general it may be any kind of domain-specific information) *together* in the same index structure. Other applications for this method include determining all cities in a given country that have between 10000 and 30000 inhabitants or retrieving all fields in a given area that have a pH-value higher than a particular value.

Closely related to spatial information is temporal information. Consider new streets being built that replace other streets, or country boundaries being changed. This kind of information includes time-varying spatial information. Therefore efficient management of spatio-temporal data is almost as important as the problems described above. This work will also show how this problem may be solved in the case where geometries change discretely (as opposed to continuously). The basic idea is to extend the

[1]The server is a recent dual-processor database server with 1 GB main memory running Oracle 8*i*/9*i* under Linux.

[2]If no additional query facilities are required the time can even be reduced to about 2 seconds. Details can be found in chapter 8.3.2.

[3]Experiments were carried out on data in the temporal domain with additional domain-specific information, see chapter 8.3.1.

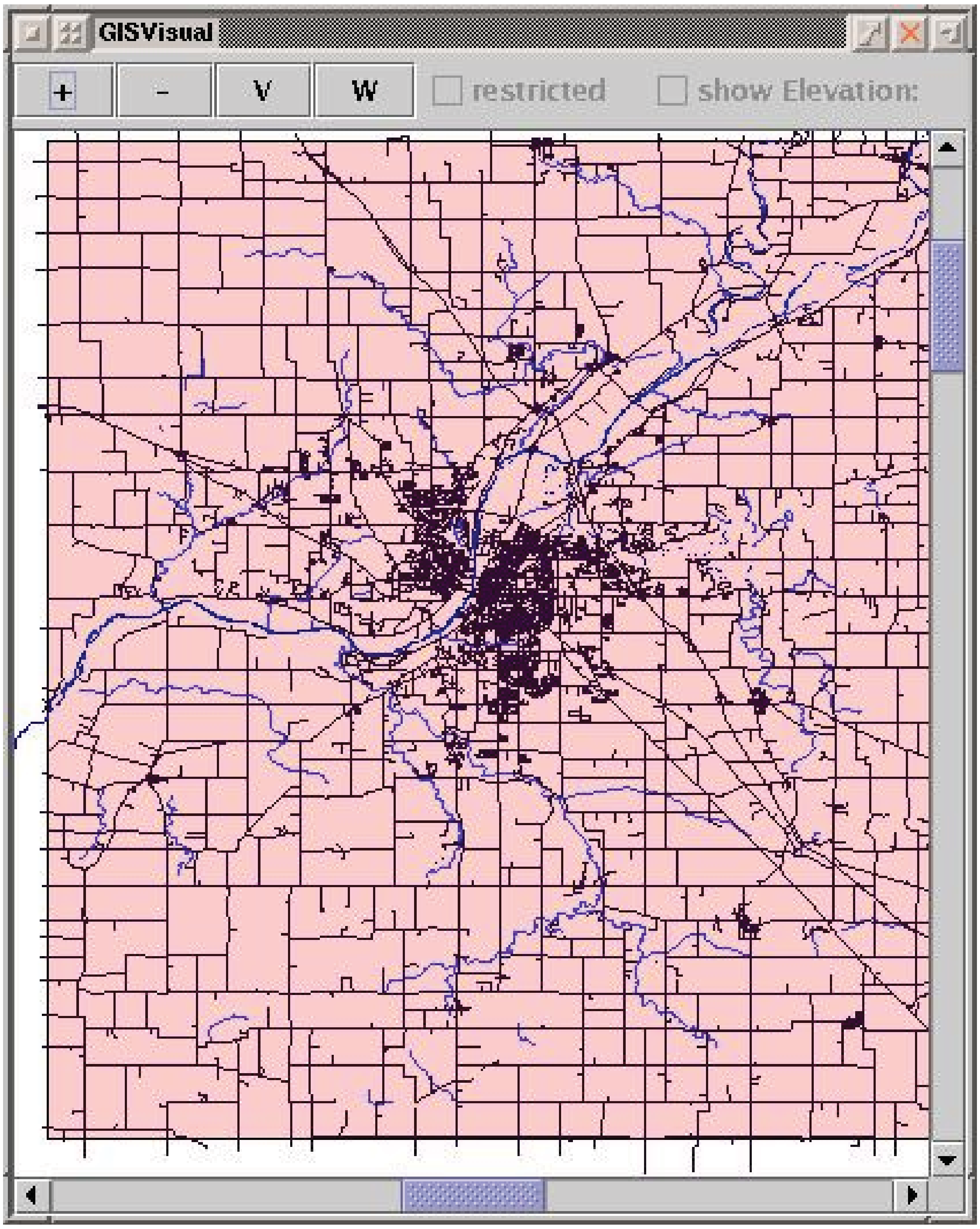

Figure 1.2: *Map of rivers and roads in Tippecanoe county*