Technical Report: alg03-001

February 11,2003

Peter Groves, pgroves@uiuc.edu
Sunayana Saha, sunayana_saha@yahoo.com
Peter Bajcsy, pbajcsy@ncsa.uiuc.edu

Automated Learning Group
National Center for Supercomputing Applications
605 East Springfield Avenue, Champaign, IL 61820

Boundary Information Storage, Retrieval, Georeferencing and Visualization

Abstract

We present an overview of a software system for storing, retrieving, georeferencing and visualizing boundary information obtained from Census 2000 Tiger/Line files, ESRI shape files or from image processing. Boundary information represents a vector data type as opposed to grid-based information denoted as a raster data type. In a Geospatial Information System (GIS) both data types are present. Vector data types are more memory efficient than raster data types for storing boundary information. However, internal organizations of vector data file formats pose a challenge to efficient information retrieval. In this work, we outline the efficiency issues related to boundary storage and boundary information retrieval.

Furthermore, we describe several ways for boundary visualization including (1) boundaries alone, (2) boundaries overlaid on a geo-referenced raster data and (3) boundaries' attributes. This paper also presents boundary processing extensions that are driven by GIS applications.

1. Introduction

There are two basic data structures used in Geo-spatial Information System (GIS). They are raster (or cellular) data and vector (or polygon) data structures [1, Chapter 15]. The need for memory efficient representations of a variety of types of GIS information drives the use of the two data structures in GIS applications. Examples of raster-based information would include terrain maps, land use maps, land cover maps and weather maps. Boundaries (or contours or outlines) of parcels, eco-regions, Census tracts or US postal zip codes would be examples of vector data. In this document we will focus on multiple representations of vector data and their file formats.

There already exist numerous file formats for storing vector data. Vector data contain points, lines, arcs, polygons or any combinations of these elements. Any vector data element can be represented in a reference domain defined by a latitude/longitude, UTM or pixel coordinate system. The challenge in storing vector data is to organize the data such that the positions and geographic meanings of vector data elements are efficiently stored and easily extracted.

Among all vector data representations in files, the following data structures have been used frequently: location list data structure (LLS), point dictionary structure (PDS), dual independent map encoding structure (DIME), chain file structure (CFS), digital line graphs (DLGs) and topologically integrated geographic encoding and referencing (TIGER) files. For detailed description of each data structure we refer a reader to [1].

In this paper, we overview boundary information extraction from (1) Census 2000 Tiger/Line files defined by the U.S. Census Bureau and saved in TIGER file data structures and (2) shape files defined by the Environmental Systems Research Institute (ESRI) and stored in a LLS data structure for efficient boundary information storage and retrieval. We also outline various tools that we have developed to derive useful information from boundary data. The optimizations done during the extraction and processing stages of the boundary data are also discussed.

2. Census 2000 Tiger/Line Files

The Census 2000 Tiger/Line Files provide geographical information on the boundaries of counties, zip codes, voting districts, and a geographic hierarchy of census relevant territories, e.g., census tracts that are composed of block groups, which are in turn composed of blocks. It also contains information on roads, rivers, landmarks, airports, etc, including both latitude/longitude coordinates and corresponding addresses [2]. A detailed digital map of the United States, including the ability to look up addresses, could therefore be created through processing of the Tiger/Line files.

2.1. File Format Description

Because the density of data in the Tiger/Line files comes at the price of a complex encoding, extracting all available information from Tiger/Line files is a major task. In this work, our focus is primarily on extracting boundary information of regions and hence other available information in Tiger/Line files is not described here.

Tiger/Line files are based on an elaboration of the chain file structure (CFS) [1], where the primary element of information is an edge. Each edge has a unique ID number (Tiger/Line ID or TLID) and is defined by two end points. In addition, each edge then has polygons associated with its left and right sides, which in turn are associated with a county, zip code, census tract, etc. The edge is also associated with a set of shape points, which provide the actual form an edge takes. The use of shape points allows for fewer polygons to be stored.

To illustrate the role of shape points, imagine a winding river that is crossed by two bridges a mile apart, and that the river is a county boundary and therefore of interest to the user. The erratic path of the river requires many points to define it, but the regions on either side of it do not change from one point to the next, only when the next bridge is reached. In this case, the two bridge/river intersections would be the end points of an edge and the exact path of the river would be represented as shape points. As a result, only one set of polygons (one on either side of the river) is necessary to represent the boundary information of many small, shape defining edges of a boundary.

This kind of vector representation has significant advantages over other methods in terms of storage space. To illustrate this point, consider that many boundaries will share the same border edges. These boundaries belong to not only neighboring regions of the same type, but also to different kinds of regions in the geographic hierarchy. As a result, storing the data contained in the Tiger/Line files in a basic location list data structure (LLS) such as ESRI shapefiles, where every boundary stores its own latitude/longitude points, would introduce a significant amount of redundancy to an already restrictively large data set.

In contrast to its apparent storage efficiency, the Tiger vector data representation is very inefficient for boundary information retrieval and requires extensive processing. From a retrieval standpoint, an efficient representation would enable direct recovery of the entire boundary of a region as a list of consecutive points. The conversion between the memory efficient (concise) and retrieval efficient forms of the data is quite laborious in terms of both software development and computation time.

Another advantage of the Tiger/Line file representation is that each type of GIS information is self-contained in a subset of files. As a result users can process only the desired information by loading a selected subset of relevant files. For example, each primary region (county) is fully represented by a maximum of 17 files. Therefore, the landmark information is separate from the county boundary definition information, which is separate from the street address information, etc. Those files that are relevant to the boundary point extraction, and the attributes of those files that are of interest, are the following:

- Record Type 1: Edge ID (TLID), Lat/Long of End Points
- Record Type 2: TLID, Shape Points
- Record Type I: TLID, Polygon ID Left, Polygon ID Right
- Record Type S: Polygon ID, Zip Code, County, Census Tract, Block Group, etc.
- Record Type P: Polygon ID, Internal Point (Lat/Long).

We denote this subset of files as "Census boundary records".

Deleted: ¶

As part of our description of the Census 2000 data, we would like to mention the difference between the U.S. postal service code areas (or zip codes) and the Census 2000 zip code analog. The zip code information provided in the Tiger/Line files do not give the actual boundaries of zip codes. They are instead Zip Code Tabulation Areas (ZCTA's), which are the Census Bureau's best attempt at representing zip code boundaries. This approximation is necessary because zip codes are not based on geographic regions, but rather on postal routes. Zip codes are therefore collections of points, not boundaries, and can be overlapping. For example, a large office building may have its own zip code while all the buildings around it share another zip code. The U.S. census bureau also made coverage of ZCTA boundaries contiguous, meaning that all of the United States is assigned to a ZCTA. Bodies of water therefore have their own ZCTA's that are designated by a five-digit code ending in 'HH'. Furthermore, some regions could not be appropriately defined as distinct ZCTA's, and are designated by the first three digits of the zip codes that the region's zip codes have in common and the suffix 'XX'.

In the current software implementation, all regions ending in 'HH' are removed with the option of also removing those with 'XX'. The default is to remove those with the 'XX'.

2.2 Software Functionality

The goal of processing a selected subset of Census boundary records from the Census 2000 Tiger/Line files, along with meta data files for each, is to generate a data structure that holds region names, lat/long points defining its boundary, neighboring regions to each boundary, and an internal point of the region. From an information retrieval standpoint, this is a much more efficient data organization of the Tiger/Line data, and such a data structure can be easily exported to shape files, which in turn can be used by many commercial GIS packages such as the free visualization tool ArcExplorer (available from the ESRI web site [3]).

2.2.1 Basic Functionality

The conversion process from Tiger/Line files to ESRI Shapefiles begins by loading the raw Tiger files into 2-D table-like data structures by making use of manually developed meta data files. As the Tiger files are fixed-width encoded flat files, meta data is necessary to define the indices of the first and last characters for each attribute in the lines of the flat file. This information, the attributes' names, and their type (integer, floating point number, string, etc) come from meta data files provided by the Census Bureau. The final piece of information contained in the meta data file is a "Remove Column" field, which dictates whether or not the attribute will be dropped from the table as it is read in. Attributes that are not used during the processing are removed early on for the sake of memory efficiency. The meta information for each Record Type is stored in a comma-separated-values (csv) file, which can easily be parsed into a table object, then accessed in that form by the routine that parses the main data file.

Once the Tiger data is in the form of tables, it is streamed through a complex system of procedures, including conversion to several intermediate data structures, before being inserted into Hierarchical Boundary Objects (HBoundarys). Each HBoundary represents one type of region (zip code, county, etc) for a single state. A high level view of this procedure, with the added step of conversion to ESRI shape files is given in Figure 1, and the D2K itinerary that achieves this, then writes the HBoundarys to an 'hba' file (a storage format designed for a set of HBoundary objects that cover the same area) is given in Figure 2.

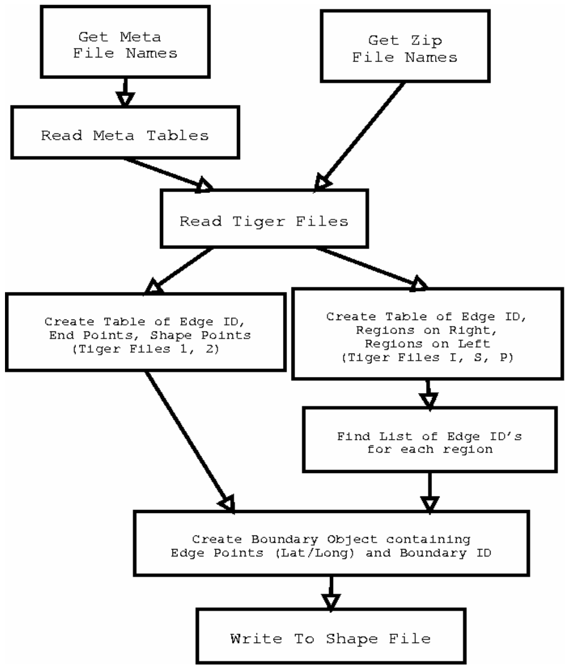
Get Meta File Names
Get Zip File Names
Read Meta Tables
Read Tiger Files
Create Table of Edge ID, End Points, Shape Points (Tiger Files 1, 2)
Create Table of Edge ID, Regions on Right, Regions on Left (Tiger Files I, S, P)
Find List of Edge ID's for each region
Create Boundary Object containing Edge Points (Lat/Long) and Boundary ID
Write To Shape File

Figure 1: A flow chart of boundary information extraction from Census 2000 Tiger/Line files (memory efficient format) into the ESRI shape file format with location list data structures (retrieval efficient format).

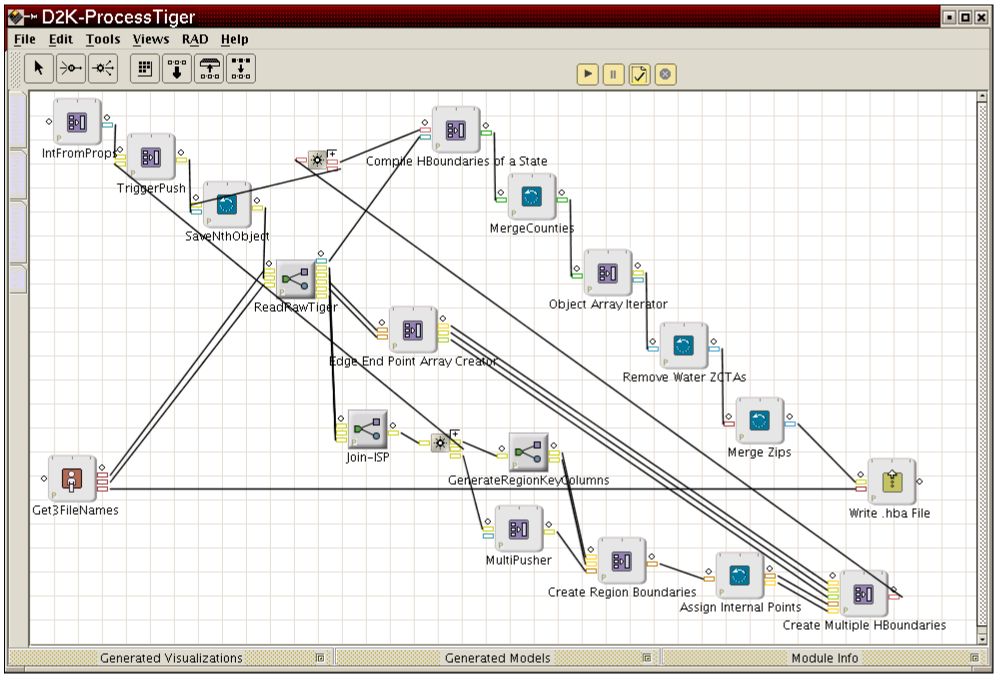

Figure 2. Raw Tiger/Line Files to HBoundary Conversion in D2K

2.2.2 HBoundary Object

HBoundarys are vector data storage and access objects that are designed to provide an efficient tradeoff between access speed and memory usage when a set of polygons/boundaries provide complete, non-overlapping coverage of a region (such as counties covering a state) and/or there are multiple types of boundaries that fall into a hierarchy (such as all state boundaries are also county boundaries, and all county boundaries are also census tract boundaries). This is done by having one master list of boundary points that all boundaries reference by pointers. Given that in the Java programming language, integers, which are used as the pointers, are 32 bits, and double precision floating point numbers, two of which are used for each geographical point (latitude and longitude), are 64 bits, the memory savings for each point that is shared by two county, two census tract, and two block group boundaries is 30 bytes. For the state of Illinois, this optimization translated into a 38% reduction in memory usage, with HBoundary objects requiring 16.45 MB of memory to store their geographic point information while a representation that stores a copy of every point for every boundary would use 26.64 MB of memory.

Deleted: ,

To access the data encapsulated by the HBoundary object(s), the Java interface ShapeSet is used. ShapeSet is a generic set of access methods (or "getters") for vector data that is part of the I2K image processing package. It was developed so that different visualization, computation and certain I/O functions could be developed for vector data without dependence on its underlying data structure.

The prime example of the use of the ShapeSet interface thus far is in writing vector data into ESRI shape files. Figure 3 shows the D2K itinerary that writes an array of HBoundary objects (read in as an hba file that was written out by the itinerary in Figure 2) into a set of shape files. The module that actually creates the shape files takes not an HBoundary specifically but rather any object that implements the ShapeSet interface. In this way, the conversion of any object that contains vector data into a set of shape files can be done without any additional coding.

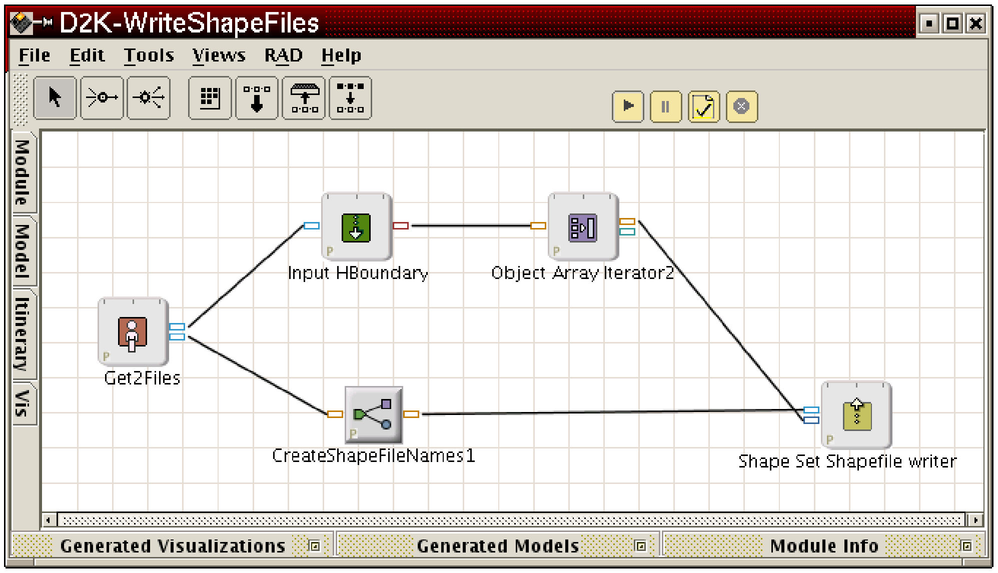

Figure 3: Writing Shapefiles. The user is prompted for the name of the hba file to read in, and a filename prefix that all written shapefiles will share.

3. ESRI Shapefiles

A shapefile is a special data file format that stores non-topological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates in a location list data structure (LLS). Shapefiles can support point, line, and area features. Area features are represented as closed loop polygons.

3.1. File Format Description

A shapefile must strictly conform to the ESRI (Environmental Systems Research Institute) specifications [4]. It consists of a main file, an index file, and a dBASE table.

The **main file** is a direct access, variable-record-length file in which each record describes a shape with a list of its vertices. In the **index file**, each record contains the offset of the corresponding main file record from the beginning of the main file. The **dBASE table** contains feature attributes with one record per feature. The one-to-one relationship between geometry and attributes is based on record number. Attribute records in the dBASE file must be in the same order as records in the main file.

All file names adhere to the ESRI Shapefile 8.3 naming convention. The 8.3 naming convention restricts the name of a file to a maximum of 8 characters, followed by a 3 letter file extension. The main file, the index file, and the dBASE file have the same prefix. The suffix for the main file is ".shp". The suffix for the index file is ".shx". The suffix for the dBASE table is ".dbf".

Examples:

1. main file: counties.shp

2. index file: counties.shx

3.DBASE table: counties.dbf

There are numerous reasons for using shapefiles. Shapefiles do not have the processing overhead of a topological data structure such as a Tiger file. They have advantages over other data sources, such as faster drawing speed and edit ability. Shapefiles handle single features that overlap or that are noncontiguous. They also typically require less disk space and are easier to read and write.

3.2. Software Functionality

As mentioned above, a shapefile is a very access efficient file format. Tiger files are also converted to shapefiles for this reason. Thus, it becomes important to have extensive support for shapefiles. In this section, we will describe the i2k tools developed for shapefiles. These include loading and saving out shapefiles, storing shapefile data in a special data structure (ShapeObject), and processing shapefile data for visualization and statistical calculations.

3.2.1 Loading A Shapefile

Given the shapefile name, the Shapefile readers read the shapefile. The implementation defines a base class called ShapefileShape and all possible boundaries types (ShapeArc, ShapePoint, etc) are subclasses of this base class. Since the number and type of boundaries in a shapefile cannot be known until the whole file has been read, the boundaries are first stored in a linked list. Depending on the type of the boundary read from the file, an object of that shape type is instantiated and inserted into the linked list. The java 'Vector' class is used to store all the boundaries. Although the Shapefile implementation users can get this Vector from the Shapefile reader class, we would advice users not to get into the intricacies of the shape records and work with ShapeObject instead. A ShapeObject (described in detail later) is an array-based data structure especially designed for efficient storage and easy retrieval of boundary data. Our implementation provides methods that convert the linked list (Vector), into which the boundaries were initially read in, to a ShapeObject.

3.2.2 Loading Multiple Shapefiles

Our tools allow users to specify multiple shapefiles that can then be loaded and viewed together. This is particularly useful when we want to view shapefiles that describe related information. For instance, individual shapefiles may describe adjoining states of United States. To load all the shapefiles together, we basically insert the shape records of all the shapefiles into a single linked list structure (similar to opening a single shapefile). We then convert this records list into a ShapeObject that the users can use like any other ShapeObject.

3.2.3 Saving a Shapefile

ShapeObject is the link between the Shapefile readers and writers and different I2K tools. Just as we can read a shapefile and get the data in a ShapeObject, similarly, we can write out a ShapeObject to a Shapefile. The Shapefile writer takes a ShapeObject and first converts the boundaries in a ShapeObject into Shapefile records' format. There is another way to save out shapes to a shapefile. A user who is very familiar with Shapefile formats may form a linked list of Shapefile records (java Vectors) in his own code, and set the ‘records’ member variable in his instance of the Shapefile class. The user can then call a different Shapefile writer that writes out the current ‘records’ to an output shapefile. However, we would not recommend using this approach unless one is extremely familiar with the Shapefile code.

If called with ShapeObject, the shapefile writer prepares the Shapefile header, which contains information like the file length, the number of records, the type of the records and the global bounding box that encloses all the records. The writer also prepares the information required for the index file. A shapefile must have a corresponding DBF file. The DBF file contains per shape-record attributes and must have as many rows as there are records in the corresponding main Shapefile. ShapeObject does not store any record attributes. However, if the DBF file is missing, commercial GIS tools refuse to open the corresponding shapefile’s main file. We wanted our shapefiles to be compliant with commercial GIS tools. Thus, when we do not have information for the DBF file, we create a DBF file with dummy fields. A scenario where we had to create a dummy DBF file was when we wanted to store the contours extracted by I2K’s iso-contour extraction tool. The iso-contour extraction tool (please refer to [5] for more detail on iso-contour tool) extracts contours from historical maps and saves them out as a shapefile. We create dummy fields in the DBF file since there are no inherent attributes with the generated contours.

Another interesting feature worth mentioning, while saving shapefiles, is that we can save the shape points in latitude/longitude instead of storing them in pixels. For instance, when we extract contours from historical maps, we have sufficient geo-specific information to convert the pixel values of the contours to latitude/longitude. Storing the contour points, in latitude/longitude in the output shapefile, enables geo-referencing later. These contour points can be geo-referenced and overlaid on a different geographical image if the points and the image lie in the same geographical location. Section 3.2.6 on talks more about geo-referencing shapefiles.

3.2.4 ShapeObject

The ShapeObject is the data structure especially designed to efficiently store and retrieve shape points loaded from a shapefile. To prevent oneself from getting caught into the complexities of Shapefiles and making common mistakes, users of Shapefile class should get a ShapeObject containing the data loaded from the shapefile. ShapeObject should be used for all internal processing purposes as it provides a common and generic implementation that has already been optimized and well tested to store and manipulate shape data.

A ‘shape’ is referred to as a ‘boundary’ in the ShapeObject code. When a ShapeObject is constructed, the number of boundaries it will hold must be specified. To allow maximum flexibility and prevent space wastage, the actual number of points within each boundary can be fixed later when the points are being actually inserted into the ShapeObject. The ShapeObject consists of the number of boundaries it stores and various arrays storing different information about boundaries. The array structures can be classified into two classes: data-holding arrays and index arrays. There are separate data-holding arrays to store the number of parts in each boundary, the number of points in each boundary, the number of points in each part of a boundary and the bounding boxes and types of each boundary. There is one single array to store the actual points of all the boundaries. Some data-holding arrays have a corresponding index array. This index array stores the starting index of each boundary's information in the corresponding data-holding array. This allows the data-holding arrays to be accessed directly according to the boundary we are interested in.

For example, there is an array that stores the starting index of each boundary's points in the common array that stores points for all the boundaries. If we want to find all the points of the second boundary, we can get the starting index of that boundary's points from its corresponding index array. Let this index be '*i*'. We can also get the number of points, *n*, in the second boundary, from the array storing the number of points in each boundary, by directly accessing the number stored in the second index of the array. To get the points of the second boundary, we can now traverse the common array storing points for all boundaries, starting from the '*i*'th point and accessing '*n*' subsequent points. Note, this explanation about the ShapeObject is simply to give an insight into the ideas we had in mind while designing the ShapeObject. There are getters and setters for all the information stored the ShapeObject, and as such, a user will not need to directly manipulate these arrays.

Now that we have described the ShapeObject, we can give an overview of the tools developed for processing Shapefile data. All the tools explained below start by reading a Shapefile, getting the data in a ShapeObject and then working on that ShapeObject.

3.2.5 Viewing a Shapefile

There are tools in I2K that can display images of different types. So that we don't have to re-invent the wheel, we decided to convert the ShapeObject to into a geo-image. This is done by setting the pixels where the shape points fall to some RGB value. The problem here is that sometimes the shape points may be latitude/longitude coordinates. This is especially true for state boundaries we extracted from the Tiger files. To display them, we have to convert the latitude/longitude points to pixels. This conversion requires geo-specific information like the Geo-center latitude of projection, the model-type etc [6]. Shapefiles and thus ShapeObject(s) don't contain geo-specific information. Therefore, to convert the latitude/longitude points from a ShapeObject to a pixel image, we have to first fill in the necessary geo-specific information on our own. Once we have the geo-information, we convert the points to pixels and store them as image points. This image can be displayed using I2K's raster image display tools [5]. Figure 4 shows the shapefile containing the state of Illinois. Figure 5 shows two shapefiles that were opened together using the mechanism described in 3.2.2 and viewed showing the I2K's raster image display tools.

Deleted: ,

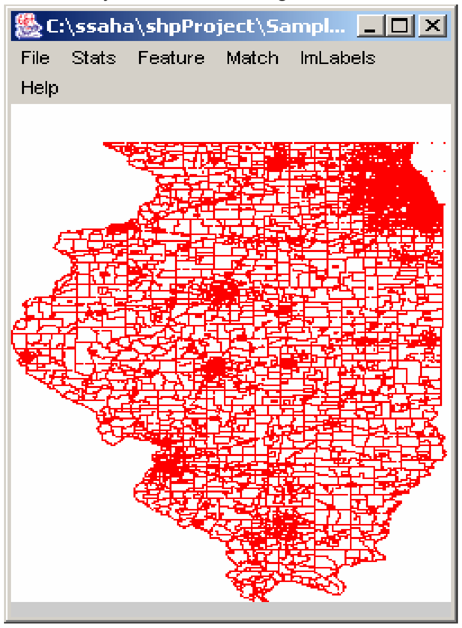

Figure 4. A screenshot of a loaded shapefile containing the state of Illinois' data.

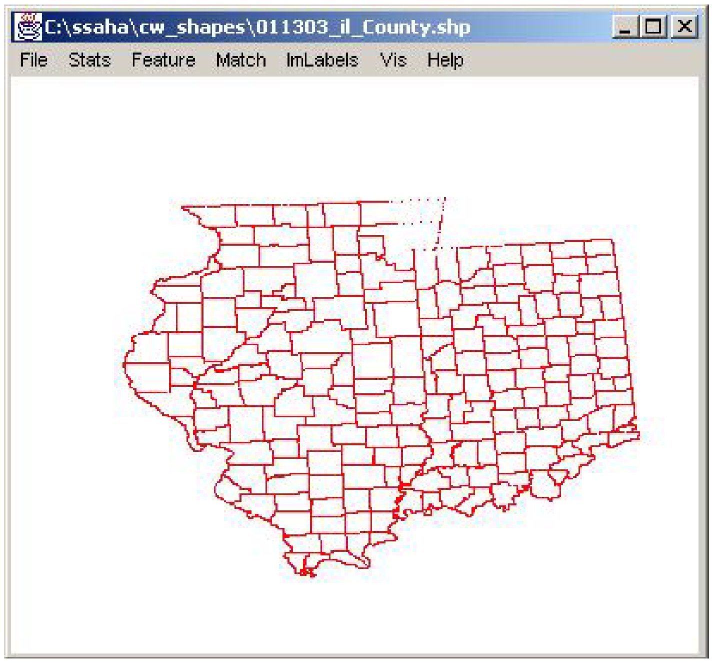

Figure 5. A screenshot of two shapefiles loaded together. One of the shapefiles had Illinois counties while the second one had the counties of the state of Indiana.

3.2.6 Georeferencing Shapefiles

Given a raster data set and a vector data set (ShapeObject) defining boundaries, we address the problem of extracting information from raster data over a set of boundaries. The assumption of this problem is that the raster and vector data sets overlap geographically and both data are described by sufficient georeferencing information for data fusion. Georeferencing, or geographical referencing, is the name given to the process of assigning values of latitude and longitude to features on a map. The tool fuses raster (e.g., elevation) and vector (e.g., county boundaries contained in shapefiles) data using geo-referencing information and displays overlaid data.

The tools we have right now have been tested for overlaying shapefiles on raster files like forest files and digital elevation maps. Overlaying shapefiles on geographical raster images is an intermediate step in collecting statistics of the information represented by pixel values in an image within boundaries defined by shapefiles.

When we overlay shapefiles (via ShapeObjects) on an image, there are many issues that must be considered. For example, a shapefile may describe a large number of points, many of which may not even fall on the underlying image. Since shape points are scanned repeatedly for different purposes, it becomes absolutely necessary to discard irrelevant points as early as possible, thus reducing the space and time used during processing. Each boundary in the ShapeObject has a bounding box defining the maximum extent of that boundary. While overlaying a ShapeObject's boundaries on an image, we first compute the bounding box of the underlying image. We then check if the bounding box of each individual boundary in the ShapeObject lies within or overlaps the image's bounding box. Those boundaries whose bounding boxes are totally outside the image's bounding box are discarded. Thus, we don't even go to the granularity of a boundary's points if its bounding box is outside the image. If a boundary's bounding box lies within, or overlaps the image's boundary, we iterate through all its points to keep only those points that fall on the image. We process the points further to make it more space and time efficient. The boundary points found relevant in the previous step may be in latitude/longitude. All visual tools in I2K, however deal with pixels. The conversion from latitude/longitude to pixels requires many complex calculations and is rather time-consuming. To eliminate the need for conversion from latitude/longitude to pixels every time we work with the shape points, the reduced set of relevant points (in latitude/longitude), are converted, exactly once during processing, into pixels. This greatly enhances the speed of calculations performed on these points later. Another issue we handled was the fact that many points in latitude/longitude, after conversion to pixel, give the same pixel value. Saving all such points would lead to unnecessary space usage. Thus, we only keep one such point, discarding all duplicate points. The reduced set of points, expressed in pixels, is stored in a new ShapeObject that is used for all future computations. The new ShapeObject is then displayed over the image using I2K's raster image display tools. Figure 6 presents a snapshot of Illinois' Counties shapefile overlaid on the digital elevation map of Illinois.

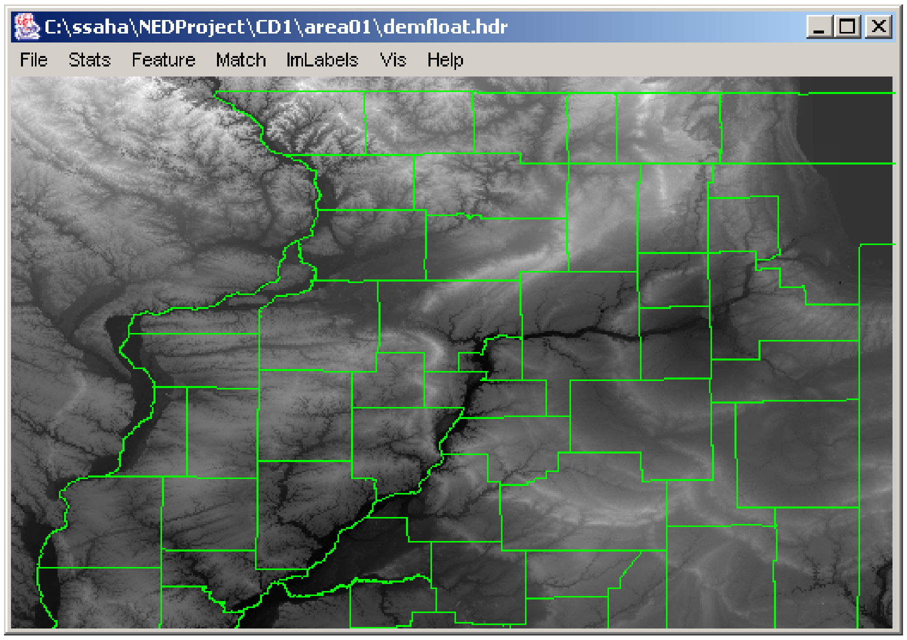
C:\ssaha\NEDProject\CD1\area01\demfloat.hdr
File Stats Feature Match ImLabels Vis Help

Figure 6: The shapefile describing the Illinois' Counties overlaid on a digital elevation map

3.2.7 GeoFeature and GeoFeatureHist Tool

Along with the tool for fusing raster and vector data described above, we also have tools called the GeoFeature tool and GeoFeatureHist tool that provides a mechanism for computing statistical descriptors of geographical regions computed from raster-based information about a geographical area, for example, raster-based information about elevation. The tool uses georeferencing and data fusion and computes statistics over each boundary. In the case of continuous raster data, such as elevation maps, we compute the sample mean, standard deviation, skew and kurtosis per boundary using the GeoFeature tool. In the case of discrete raster data such as forest labels file, we compute histograms of occurrence of different forest labels in each boundary, using the GeoFeatureHist tool. The results of the computation can be reported in a DBF file, as well as in a visual form like tables and color-coded images. Figure 7 shows the mean elevation per county of

Illinois calculated over the digital elevation map of Illinois. In the gray scaled image, the darker the color in a region, the higher the region's mean elevation.

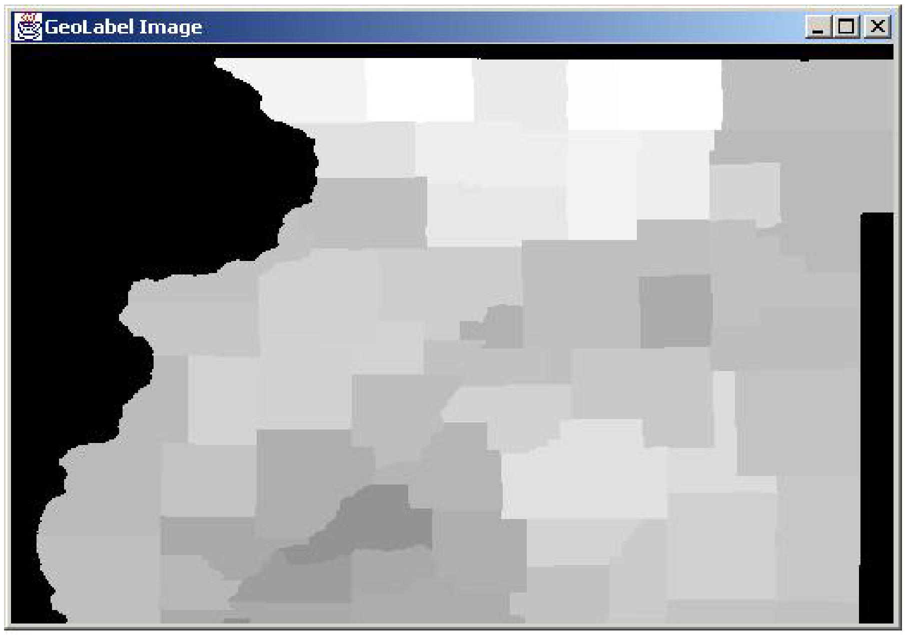

Figure 7. Grayscale coded image showing the mean elevation per county of Illinois

3.2.8 Extensions to DBASE file support

The DBF files used to store attributes of the regions represented by shapefiles is, in essence, a simple two-dimensional table that allows different types of data in each column. As such, our system reads the DBF files associated with shapefiles into a data structure called Table that is part of the D2K data mining core libraries. The Table class, in its simplest form, is a collection of columns, represented by arrays of various types, that contain meta-data such as attribute names and types. Importing DBF files into this structure allows the information in the DBF files to be modified using an extensive library of functionality developed for use with the Table interface. This includes functions for sorting, merging multiple Tables, and scaling. What's more, the Table interface is the primary input to advanced statistical modeling procedures developed for generic data mining applications in D2K.

4. Summary

In this paper we have presented an overview of a software system for storing, retrieving, georeferencing and visualizing boundary information obtained from Census 2000 Tiger/Line files, ESRI shape files or from image processing. Various tools for processing shapefile data are also discussed. The paper talks about some problems encountered while working on the tools and the solutions we have devised to overcome the problems. We have also given an overview of the storage and retrieval optimizations we have considered. Future works involves developing more tools and extending the current tools for processing and extracting different kinds of information from different file formats.

References

[1] Campbell, James B. Introduction to Remote Sensing, Second Edition. The Guilford Press, New York. 1996.

[2] Miller, Catherine L. Tiger/Line Files Technical Documentation. UA 2000.
US Department of Commerce, Geography Division, U.S. Census Bureau.
http://www.census.gov/geo/www/tiger/tigerua/ua2ktgr.pdf

[3] ArcExplorer, ESRI web site:
http://www.esri.com

[4] ESRI Shape file, File Format Specification,
http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[5] Bajcsy P., "Image To Knowledge", documentation at web site:
http://www.ncsa.uiuc.edu/Division/DMV/ALG/activities/projects/i2k/documentation/index.html

[6] Alumbaugh T.J. and Bajcsy P., "Georeferencing Maps with Contours in i2k", ALG NCSA technical report, alg02-001, October 11 2002.

Deleted: ¶
¶
¶