STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

SPATIAL DATA CAPTURE IN GIS ENVIRONMENT

A. M. IMBROANE

Abstract. Traditionally geographical data is presented on maps using symbols, lines and colours. A map is an effective medium for presentation and a database for storing geographical data. But herein lies some limitations. The stored information is processed and presented in a particular way and for a particular purpose. A map provides a fixed, static picture of geography that is almost always a compromise between many different users. Compared to maps GIS (Geographical Information System) has the inherent advantage that data storage and data presentation are separate and may be presented and viewed in various ways. The core of GIS environment is the matching of spatial data (digital maps) and attribute data (the meaningful of spatial data) together. The attribute data are in fact tables associated with geographical features stored in spatial database. The main problem is capturing and storing spatial data in digital form. In this paper we will approach just spatial data capture in vector format and not the database organization.

1. Introduction

The geographic entities or objects in GIS are based on two different types of data: spatial and attribute. Spatial data representation in GIS are classified into raster and vector, which are dual with regard to space bounding and space filling [6]. Parallel to representational duality, there is the duality of spatial concepts, namely entity-based and field-based concepts [4]. The fundamental difference between the representations as well as between spatial concepts causes problems in interoperability and multi-source fusion [8]. Attributes or descriptive, or aspatial data are alphanumeric data related to the graphic entities. They are also called thematic data because they contain themselves a theme of the graphic features. Attributes of vector units are stored in computer files as records or tuples that may be linked to them by pointers. Usually the attributes are stored in traditional relational database. This operation is called *geocoding* or *address matching*. The fact is that attributes data can be linked with other tabular data (external) and so the new data can be processed together with the others. This is one of the powerful specific operations in GIS. Spatial data and attribute data can be stored

2000 *Mathematics Subject Classification.* 91B99.

1998 *CR Categories and Descriptors.* J.2. [**Computer Applications**] : Physical Sciences and Engineering – *Earth and Atmospheric Sciences.*

separately or together depending of the database design performed by the GIS proprietary. A geographical database is a set of geometric entities (spatial features) and attributes. The major problems are to capture, store, manipulate, maintain and process the spatial data. In order to obtain a digital map which can be processed by GIS software it has to be depicted in separated thematic layers, like: topography, geology, hydrology, and so on. The next step is to separate points, lines and polygons associated with real entities from the layers above. And finally, each graphic feature, points, lines, polygons have to contain just a single theme, like towns, roads, administrative boundaries.

In this paper we will approach only vector format and related attributes. The vector format can be represented in two components: *geometric (graphic)* data and *topological* data. Geometric data have a quantitative nature and are used to represent coordinates, lines, areas etc. The two-dimensional vector format has three subtypes: *point*, *line* and *polygon*, named *features*. These features are called also 'graphics primitives' (definition borough from CAD/CAM software). Geometric, or non-topological data models are those in which any positional information are recorded. The recognition of individual spatial units as separate unconnected elements characterized the early data organization for digital cartography. Topological data describe the relationship between the geometric data. There are several types of topological relationship: *connectivity*, *adjacency*, and *inclusion*. Examples of queries concerning topological relationship are: which areas are neighbors of each other (adjacency), which lines are connected and form a network or a road (connectivity), and which lakes lie in a certain country (inclusion). Topological data are not always stored explicitly, because in principle they can be derived from geometric data. Note that this definition of topological data is slightly different from the stricter one used in mathematics.

To avoid the confusions and for simplicity it is better that every type feature represents only one type of an object in a layer. Example: in a point theme we have only towns and not other kind of point object. In a line theme we have only rivers and not roads.

2. Data capture programs in pseudo-code form for point feature

The core of the vector data storage is the capturing data from digitizer tablet. This procedure will be used for point feature storage, line feature storage, and polygon feature storage. We have different programs in pseudo-code for the three spatial features: point, line and polygon.

```
Program POINT                          {Captures and stores x,y coordinates}
open VECTOR_POINT
repeat
  read cod                               {cod =1 write in file, cod =-1 end}
  call DIGITIZE(x,y,id$)
  put id$,x,y
```

```
until cod =-1
close VECTOR_POINT
end
```

Subsequently we will illustrate the transfer of a string of data in ASCII format from a tablet and extract the (x,y) coordinate. The procedure DIGITIZE calls the port repeatedly looking for the data, assembles the (x,y) coordinates.

```
procedure DIGITIZE(x,y,id$)                                  {Get x, y and id$}
call GET_STRING(str$)
x=num(val(str$,1,4))
y=num(val(str$,5,4))
id$= num(val(str$,9,4)
endprocedure

procedure GET_STRING(str$)              {Extracts characters from the port}
str$="":port_signal=0
repeat
  call PORT(port_signal)
  call INKEY(key$)
  if key$ <> " " then
    char$=GET$
    str$=str$+char$
  endif
until (char$=chr$(10)) or (key = " ")
endprocedure
```

Val$ function extracts x, y values and the identifier from a specific part of string, **Num** function converts from string to numeric value. So the x coordinate is obtained at the numerical value of part of str$ string, starting at the character 1 and ending at character 4; y is starting at character 5 and ending at 8. The identifier feature (id$) is extracted from character position 9 and end at 12. The program may be used for control points and point theme as well.

The next step is to create attribute table linked together with VECTOR_POINT which contain descriptive data on point features. This mean to open VECTOR_POINT and read every identifier code and create for it a field (character or numeric). In subsequent we present the program in general form.

```
Program ATTRIBUTE_POINT                          {Creates the attribute table}
open VECTOR_POINT                       {assigned to point feature table}
open ATTRIB_POINT
repeat
  get id$,x,y
  repeat
    put id$
    read file_type                  {1 for char 0 for numeric -1 for finish}
    if file_type=1 then read chr_field$
    put chr_field$
```

```
    if file_type=0 then read num_field
    put num_field
  until file_type=-1
until endfile
close VECTOR_POINT
close ATTRIB_POINT
end
```

Usually just a few fields are generated in the attribute data file. More data can be joined from an external relational database. The join condition is that every file has a common field.

Examples. Assume that we must create a point theme, which is representing towns at sufficient small scale that the shape of the town is not important. In the VECTOR_POINT file we have the coordinates of every town and in ATTRIBUTE_POINT we can have the subsequent fields: town code (which must be unique), name, population and so on.

3. Data capture programs for line feature

For the line feature storage it is convenient to use multi-button cursor. The functionality of the buttons is pre-specified. Here we define one button (say button 1) to denote a digitized point, another (button 2) to mean 'start node – start of arc', and the other (button 3) 'end node end of arc'. The following code shows how a vector of points can be build up by correctly interpreting incoming data from the tablet using this principle. Such code normally is treated as a procedure or a sub-routine within the program controlling the data generation process.

```
Program LINE                                        {Captures line feature}
open VECTOR_LINE
repeat
  read cod                              {cod=1 write in file; cod=-1 for end}
  k=0
  repeat
    call GET_STRING (str$)
    k=k+1
    con$=val(str$,9,1)
    x=num(val(str$,1,4))
    y=num(val(str$,5,4))
    if con$="button2" then     begin
      line(k,1)=x: line(k,2)=y end
    if con$="button3" then     begin
      line(k,3)=x: line(k,4)=y end
    if con$="button1" then begin
      line(k,3)=x: line(k,4)=y
      line(k+1,1)=x: line(k+1,2)=y
    endif
```

```
  until con$="button3"
  read id$
  put id$
  kk=k-1                                    {kk = total number of segments}
  put line(1,1),line(1,2),line(1,3),line(1,4),kk
  for k=1 to kk
    put line(k+1,1), line(k+1,2)
  endfor
until cod=-1
close VECTOR_LINE
end
```

The attribute file contains description data on line feature. Usually line attribute contains the arc length, beside others. This characteristic is required in almost all spatial procedures analysis. The algorithms that follow use the next functions two compute the length of a segment and area of a triangle:

```
length(x1,y1,x2,y2) = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))
area(x1,y1,x2,y2,x3,y3) = (x1*y2+x2*y3+x3*y1-y1*x2-y2*x3-y3*x1)/2
```

An algorithm for the attribute file is given below.

```
Program  ATTRIBUTE_LINE          {Creates attribute table for line feature}
open VECTOR_LINE
open ATTRIB_ LINE
repeat
  get id$,xs,ys,xe,ye,n
  len =length(xs,ys,x(1),y(1))
  for i=1 to n-1
    len = len + length(x(i),y(i),x(i+1),y(i+1))
  endfor
  len = len + length(x(n),y(n),xe,ye)
  put id$
  put len
  repeat
    read file_type               {1 for char, 0 for numeric, -1 for finish}
    if file_type=1 then read chr_field$
    put chr_field$
    if file_type=0 then read num_field
    put num_field
  until file_type=-1
until endfile
close VECTOR_LINE
close ATTRIB_ LINE
end
```

Examples. If LINE_VECTOR refers to roads, the attribute data must be code of the road, name, length, quality and so on. If LINE_VECTOR refers to rivers,

attribute data must be code of the river, name, length, quality, mean debit and so on.

4. Data capture programs for polygon feature

The polygon theme is in a way synonym with the line theme. In fact a polygon is a closed line (start point and end point are the same). When the line is finished placing the cursor in the right location (exactly in the same location where the start point is) might be problematic. This is a real problem because we are not able to mark exactly the same point at different times (if we move a little bit the cursor). It is an "error" problem. So it is necessary to establish an error inside which the two points represent the same point. It is often named *snap node error* (or *snap node tolerance*). This is linked with the digitizer precision. Anyway the error is establish sufficiently small for the purpose of using the map. Operator specifies this error before the digitizing process is running. If the two nodes are inside error tolerance, the start node 'snap' the end node and therefore we have just one node (xe=xs, ye=ys).

Every polygon must contain an isolated point inside the polygon (not necessarily in centre) named centroid. Like for the line feature storage we use a multi-button cursor.

```
Program POLYGON                        {Captures and stores polygon features}
open VECTOR_POLY
read snap_point
repeat
  read cod              {if cod=1 then capture polygon; if cod=-1 then end}
10 k=0
  repeat
    call GET_STRING (str$)
    k=k+1
    con$=val(str$,9,1)
    x=num(val(str$,1,4))
    y=num(val(str$,5,4))
    if con$="button2" then begin
      line(k,1)=x: line(k,2)=y
    end else if con$="button3" then begin
      line(k,3)=x: line(k,4)=y
    end else if con$="button1" then begin
      line(k,3)=x: line(k,y)=y
      line(k+1,1)=x: line(k+1,2)=y
    end
  until con$="button3"
  xs=line(k,1): ys=line(k,2): xe=line(k,3): ye=line(k,4)
  dist = length(xs,ys,xe,ye)
  if dist>snap_point then goto 10
```

```
  kk=k-1                                        {kk = total number of segments}
  read id$
  put  id$
  put xs,ys, kk
  call DIGITIZE (x,y,id$)
  put x,y
  for k=1 to kk
    put line(k+1,1), line(k+1,2)
  endfor
until cod=-1
close VECTOR_POLY
end
```

A multi-polygon map often requires a hierarchical data structure because polygons will share common boundary and line segments will terminate at the common nodes. Moreover, to produce cartographic output, any combination of polygons may be required.

Usually the attribute table contains beside identifier, both perimeter and area of the polygon, followed by an arbitrary number of fields. For the perimeter we can use the same procedure as for arc length. The measurement of an irregular feature such a polygon can be done by calculating the areas of the trapezoids under the successive line segments which make up the polygon [1]. Another method for finding area of a polygon is to decompose the polygon in triangles using the centroid coordinate, and finally calculate area of each triangle.

For an effective evaluation of the area we depict the polygon in three parts: first triangle, intermediate triangles and last triangle. A pseudo-code program for attribute file is:

```
Program  ATTRIBUTE_POLY                        {Creates the attribute table}
open VECTOR_ POLY                            {associated with polygon feature}
open ATTRIB_ POLY
repeat
  get id$,xs,ys,n
  get xc,yc
  len = length(xs,ys,x(1),y(1))
  a = area(xs,ys,x(1),y(1),xc,yc)
  for i=1 to n-1
    len = len + length(x(i),y(i),x(i+1),y(i+1))
    a = a + area(x(i),y(i),x(i+1),y(i+1),xc,yc)
  endfor
  len = len + length(x(n),y(n),xs,ys)
  a = a + area(xs,ys,x(n),y(n),xc,yc)
  put id$,len,area
  repeat
    read file_type               {1 for char, 0 for numeric, -1 for finish}
    if file_type=1 then read chr_field$
```

```
    put chr_field $
    if file_type=0 then read num_field
    put num_field
  until file_type=-1
until endfile
close VECTOR_ POLY
close ATTRIB_ POLY
end
```

Data entry process, such as retrieval of thematic data from secondary sources, or topographic data capture from digitizing operations, are typically managed within a GIS. The task is to provide the user with techniques for interfacing with input device and file handling procedures.

References

[1] Bracken I., Webster C., *Information Technology in Geography and Planning*, Routledge, London, 1990.

[2] Buttenfield B.P., *Digital Definitions of Scale Dependent Line Structures*, in Proceeding Auto Carto M.J. Blakemore (ed), vol. I, 1986, 497–506.

[3] Foley J. D. and Van Dam A., *Fundamentals of Computer Graphics*, Addison-Wesley, Reading, Mass, 1982.

[4] Frank A.U., *Spatial Concepts, Geometric Data Models and Geometric Data Structures*, Computers & Geoscience, 18, 1992, 409-417.

[5] Imbroane, A.M., Moore D., *Iniţiere în GIS şi Teledetecţie*, Presa Univ. Clujeană, Cluj-Napoca, 1999.

[6] Peuquet D.J., *A Conceptual Framework and Comparison of Spatial Data Models*, Cartographica, 66, 1984, 113–121.

[7] Wallace V.L., *The Semantics of Graphics Input Devices*, Computer Graphics, 10, 1976, 1, 61–65.

[8] Winter S., *Bridging Vector and Raster Representation in GIS*, ACM, 11, 1998, 57–63.

"Babeş-Bolyai" University, Faculty of Geography
E-mail address: alex@geografie.ubbcluj.ro