THE dbmap SYSTEM

Donald F. Cooke
Geographic Data Technology, Inc
13 Dartmouth College Highway
Lyme, New Hampshire 03768

Beman G. Dawes
RD #2, Box 35
Onancock, Virginia 23417

ABSTRACT

The dbmap system consists of a high-level language for specifying digital mapping and geographic information systems and a compiler/execution monitor to run an application programmed in dbmap. The system is readily expandable due to an open-architecture design, and produces applications with very modest execution overhead. dbmap currently runs on IBM PC-AT computers and can probably be ported to any system with an ANSI "C" compiler. Operational systems written in dbmap have proven their worth in over six months of testing in production environments.

BACKGROUND

After six years of writing monolithic computer programs to support map digitizing, updating and editing operations, we designed and implemented a language -- "dbmap" -- expressly intended for generating a wide variety of application packages. We had become frustrated with maintenance and modification of an increasing number of operational systems, each of which needed support from an experienced programmer. We were also faced with opportunities to market diverse map-related software systems applied to vehicle routing and dispatching, map updating, geocoding and spatial analysis.

We had rejected conventional approaches to "generalized" mapping such as systems supplied by Intergraph or E.S.R.I. for several reasons: first of all, in 1980, none were addressing topological data structures explicitly. Secondly, such systems sacrificed execution efficiency for generality and flexibility. Finally, they could not be used as the foundation for inexpensive added value products because of their high cost and the price of the hardware they required.

In January 1986, we started a design process which ran for three months. We rejected both previous monolithic programs and the ARC-INFO/Intergraph paradigm of a flexible, tailorable system. Instead we settled on a simple, powerful language capable of specifying the characteristics of a desired system. The language, called "dbmap", is simple to learn and extremely flexible.

A junior programmer can learn dbmap in a few days, especially if there are a variety of examples to follow. As with other languages, the dbmap language statements must be keyed into a command file which is read by the dbmap compiler.

The dbmap compiler generates "P-code" which is immediately executed.

The dbmap compiler is currently written in ANSI "C". We expect to run dbmap primarily in IBM PC-AT and '386 environments. Our experience with running the "db" subset (database management only, no graphics) on a Data General MV-4000 suggests that dbmap would perform well on Macintosh, VAX or other machines.

In general, we designed for the fast single-user systems we expect will prevail in the near future. A typical minimum configuration that we run in daily operation is a 640K PC-AT clone running at 8 Megahertz. We configure such a machine with a fast 40 Mbyte Winchester disk, a mouse and two monitors: EGA for map display and Hercules monochrome for text. Cost per workstation at "street price" is about $3500.

APPROACH

We developed dbmap using traditional software engineering methodology:

* Problem analysis
* Requirements definition
* Overall system design
* Component design, implementation, and testing
* System level testing
* Refinement and maintenance

As in other non-trivial systems, the actual process was often more interative than sequential. Two steps forward were often followed by one or two steps back.

INTEGRATED SYSTEM

dbmap is a single integrated software system rather than a series of separate application systems because of interaction between applications. For example, address matching (geocoding) is improved by the capability of displaying a map of matched addresses.

Consequently, we implemented dbmap as a single large program able to perform almost one hundred separate functions. Some of the functions are simple, such as changing the color of the cursor. Others are of medium complexity, such as performing an index file search. Very complex functions include creation of a complete database through a screen manager.

PROGRAMMING LANGUAGE

dbmap is designed to be programmable rather than fixed because the details of how an activity is carried out vary according to the context. For example, digitizing nodes or shape points in a Census Bureau TIGER database is a very different context from digitizing address matching rejects though the underlying digitizer functionality is the same.

dbmap therefore is a high-level programming language with facilities for data definition, execution sequence control and expression evaluation. These facilites can be used both

procedurally, like Pascal, FORTRAN or "C" programs, or declaratively to specify database and other parameters.

Execution of a dbmap application requires a dbmap language compiler/execution monitor which is described below.

OPEN ARCHITECTURE

We designed dbmap as an open system since we did not feel we could anticipate all required functions at design time. In the event that a new function (converting Arabic numbers to Roman numerals, or decimals to fractions) is needed, it is possible to define and write the function in such a way that the dbmap system makes it available to all dbmap applications.

dbmap's open software architecture presently requires that new functions be written in "C". A dbmap function begins with a control block that specifies dbmap language formal argument requirements to the dbmap compiler. This scheme allows the dbmap compiler to handle an ever-expanding library of functions as if all functions had been built into the language when the system was designed.

High level functions such as input/output, database indexing, graphic and geographic operations are not built into the dbmap language but have been implemented via the open architecture methodology.

New functionality can be added either through "value-producing" functions (similar to subroutines) or at a much higher (superstructure) level which allows creation of a complex environment like a sort/merge utility or report generator.

DATABASE MODELS

dbmap supports a variety of database structures because no single database design can serve all applications. For example, (1) the network database model (2D) works well for DIME or TIGER file creation and maintenance, (2) an unnormalized relational model (the traditional 300-byte DIME file) is better as a distribution format and (3) a point database is adequate for supporting centroid-based retrieval systems like "On-Site" and "Area Profile Reports". Route optimizing or choropleth mapping may require still other structures.

In internal memory, dbmap often uses a network model, but external disk files may be any structure. In general, the needs of the application determine the file structure.

OPERATING ENVIRONMENT

We required flexibility in hardware and operating system environments for dbmap applications since no single environment can serve all potential uses. For example, spatial spreadsheet (desktop GIS) manipulation calls for a personal computer, map digitizing is a workstation application and large-scale address matching is still a mainframe operation. Applications on the new Apple, Amiga and Atari machines (not

to mention the forthcoming CD/I systems) look increasingly attractive.

The only common thread in this mix of environments is the likelihood of availability of a "C" compiler. dbmap was implemented using the draft proposed ANSI "C" language. Considerable care was taken to insure portable code, with initial development done on AT-Clone microcomputers. Subsequently the non-graphics portions of dbmap have been moved to a Data General MV-4000 minicomputer; all code compiled and ran correctly on the first attempt and has been used in that environment for several months.

TRADEOFFS

Operating speed is of considerable importance to dbmap users for economic reasons and ease-of-use. Many users will spend much of their workday using dbmap, so slow response or error prone applications are not acceptable.

Ease of programming is not as important as flexibility and ease of use. For every person hour spent programming applications in dbmap, hundreds of person hours will be spent using the applications. On the other hand, programmers must be much more productive in dbmap than in languages like C or Pascal to avoid traditional software bottlenecks - and these same dbmap programmers will likely be less experienced than typical C or Pascal programmers!

The dbmap compiler uses a compile and go approach similar to commercial "turbo" compilers. Thus a dbmap program is compiled into memory each time it is used. Compile time is less than half a second for simple dbmap programs, and less than ten seconds for very large programs.

The fast combined compile/test cycle aids dbmap programmer productivity.

Compiling each use implies that the latest version of dbmap functions are always invoked. Thus if a bug is fixed or an algorithm is improved, the benefits apply right away to all applications which use the function.
Compiling rather than interpreting results in acceptable operating speed. For example, complex data processing jobs commonly run at 100,000 to 200,000 records an hour on an AT-clone while simple sequential file searches run at over 1,000,000 records per hour. File operations actually run acceptably fast on a PC-clone but graphics operations need the speed of an AT at a minimum.

"ZONE RANGER" -- A dbmap APPLICATION

The first system written in dbmap was one we call "Zone Ranger". We use Zone Ranger to create custom address coding guides for computerized dispatching services. Customers indicate their delivery territories (zones) on standard maps, usually USGS 7.5 minute quads, by outlining the zone boundaries with a marker. We display corresponding images of our digital maps on Zone Ranger's EGA monitor and use the mouse to "lasso" each zone in turn. When all zones have

been lassoed (and line-segments in our data base tagged with zone numbers) a utility program creates the customer's address coding guide in deliverable format.

Figure 1 illustrates the Zone Ranger CRT display in a monochrome implementation used for these figures. The main window shows a zoomed image of the New York City financial district selected from the Manhattan's 8753 line segments previously loaded into dbmap's working RAM. The operator has used the "Name" function to label ten streets.

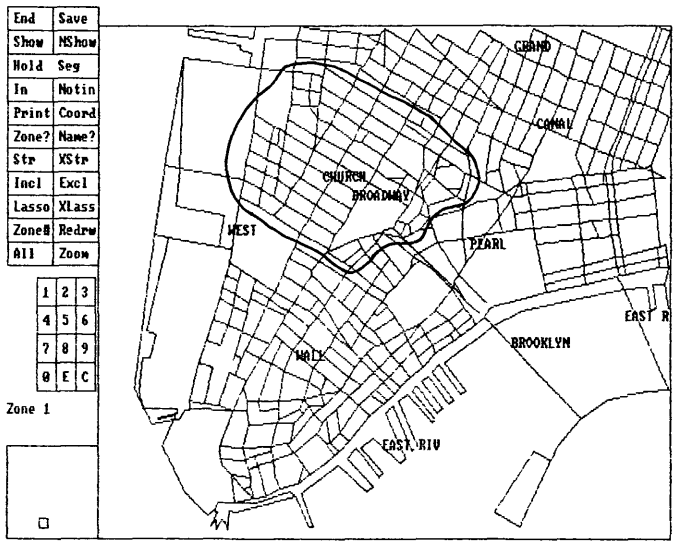

Figure 1 Zone Ranger CRT display

On the left of the screen from top to bottom are the 22-item main function menu, a "soft" keypad for entry of zone numbers, the current zone identifier and a small orientation map that confirms that we've zoomed in on a small area in the south of the submodel.

The operator can toggle line segments in or out of the current zone one at a time (Incl & Excl), a street at a time (Str & XStr) or by using the mouse to lasso a group of segments (Lasso & XLass). Figure 1 shows the lasso trace of a delivery zone northeast of the World Trade Center.

In this example the operator activated the lasso function by clicking the mouse in the "Lasso" menu box, the 17th pane of the main function window. This action activates case "win.p=17" of the dbmap language code for the Zone Ranger application (Figure 2).

Figure 2 shows all of the dbmap code needed to specify the lasso capability. Line 1 identifies the code for clicking on the 17th pane of the main menu; an exclamation point delimits a comment field. "setwidth(2)" invokes a dbmap function that makes subsequent calls to "draw()" plot

```
case win.p=17      ! Lasso segments into current zone
setgwidth(2)       ! Make "draw" default double-width line
while lasso(SEG,segdata,    ! Test: segments in lasso?
  {segflag <- 1             ! Set flag if segment is in
   draw(SEG id())},         ! Redraw included segments
  "Hold button down while lassoing segments to include")
endwhile           ! End of case win.p=17
```

Figure 2 dbmap Code for Zone Ranger Lasso Function

double-width lines. (In the EGA color version of Zone Ranger we choose a line color instead of width.)

Lasso needs four parameters:

> **lasso(<cell type>, <data>, <action>, <help>)**
>
> **lasso** displays the **<help>** message, which must be a string constant,
> and interacts via **mouse/digitizer** and graphics display to identify **zero** or more internal
> database entries for **<celltype>**.
> Then, for each of these internal database entries in turn, <data> is retrieved,
> **<action>** is performed, and **<data>** then replaces the original database entry
>
> For practical purposes, **<action>** will usually be a procedure which modifies
> **<data>** to effect the purpose of the **lasso**
>
> **lasso** returns **1** if successful, **0** if failed Failure would imply out of current window,
> operator decided to terminate, or similar non-completion

Figure 3 "Lasso" Function Definition from dbmap Manual

The first parameter says we're interested in lassoing line segments in the database, not nodes, points or other objects. "segdata" is the name of a data control block for line segment data, one element of which is "segflag" which if set indicates inclusion in the current zone.

"<action>" happens to each segment determined to be inside the lasso: "segflag" gets set to 1 and the segment is redrawn, now with double line width. The help message appears on top of the main map screen as the lasso is being drawn. (The help message doesn't show in Figure 1 which really shows the "Print" function.)

Figure 4 shows the Zone Range screen after the lasso operation.

CONCLUSION

It's probably clear from the example that dbmap is more difficult to program than a general database system like dBase III or RBase 5000. We feel that dbmap's ability to handle the more complex world of computer cartography compensates for and explains its demands on the programmer. We have found that junior programmers can become conversant with dbmap in a week or two and can master implementation of a new system after a month's experience.

A year ago we had not begun implementation of dbmap; at present we have several production systems in operation in-

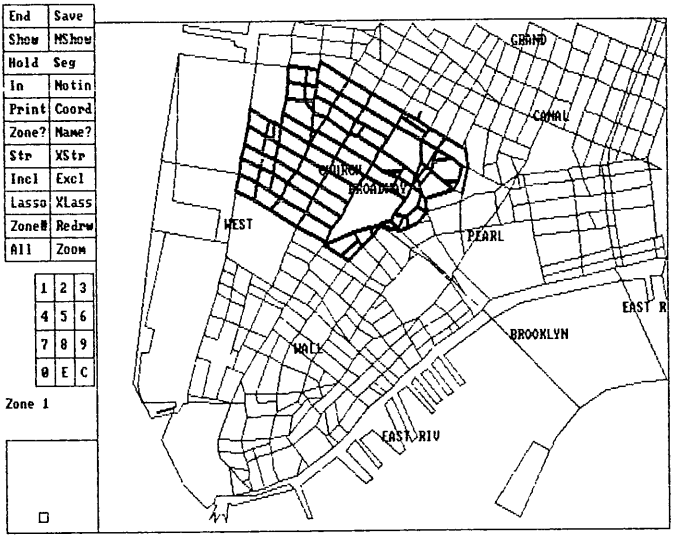

Figure 4 Result of Lasso Operation on Figure 1

house and at customers' sites. We are unequivocally pleased with both the dbmap language and the applications developed so far. We expect dbmap applications to supply all our internal mapping needs by mid-summer, 1987 and to provide the foundation of a family of integrated mapping products.

ACKNOWLEDGMENTS

The authors wish to thank Robert Reeder, Ted Jerome and Patrick Ferraris for their valuable suggestions and contributions.