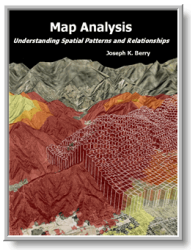
Beyond Mapping III

Topic 1: Object-Oriented Technology and Its GIS Expressions



***Map Analysis** book with companion CD-ROM for hands-on exercises and further reading*

What Is Object-Oriented Technology Anyway? *— establishes the basic concepts in object-oriented technology*
Spatial Objects—the Parse and Parcel of GIS? — discusses database objects and their map expressions
Does Anyone Object? *— discusses some concerns of object-oriented GIS*
Observe the Evolving GIS Mind Set *— illustrates the "map-ematical" approach to analyzing mapped data*

Note: The processing and figures discussed in this topic were derived using MapCalc™ software. See www.innovativegis.com to download a free MapCalc Learner version with tutorial materials for classroom and self-learning map analysis concepts and procedures.

<Click here> right-click to download a printer-friendly version of this topic (.pdf).

(Back to the Table of Contents)

What Is Object-Oriented Technology Anyway?

(GeoWorld, September 1996, pg. 26)

(return to top of Topic)

Object-oriented technology is revolutionizing GIS. At every turn, from researchers to salespersons, the buzz words "object-oriented" crop up. Mere mention of the concept greases the probability of funding and immediately places the competition on the defensive. But what does object-oriented really mean? Are there different types? Are there shades of object-oriented-ness? Can a system be "just a little" object-oriented? How can you tell?

Like most over-used and under-grasped terms, object-oriented (OO, pronounced like a cow's moo, but without the "m") has a variety of near religious interpretations. In reality, however, there are just three basic forms: 1) Object-Oriented User Interfaces (OOUI) , 2) Object-Oriented Programming Systems (OOPS) and 3) Object-Oriented Data Base Management (OODBM). Yep, you got it… *oo-eee, oo-la-la, oops 'n ooD'BUM, ra'ma, ra'ma ding-dong, whoopee…* sounds more lyrics from a 60's hit than a GIS revolution. *OOUI* uses "icons" in place of command lines and/or menu selections to launch repetitive

procedures. It's what used to make a Mac a Mac, and all the other PC's just techno-frustration.

With the advent of Windows '95, however, OOUI's have moved from "state-of-the-art" to commonplace. Its objects include all those things on the screen you that drag, drop, slide, toggle, push and click. On the "desktop" you have cute little sketches including file folders, program shortcuts, a recycle bin and task bar. In a "dialog box" the basic objects include the check box, command button, radio (or option) button, slider bar and spinner button, in addition to the drop-down, edit and list boxes. When activated, these objects allow you to communicate with computers by *point 'n click*, instead of *type 'n retype*. Keep in mind that OOUI defines "a friendly user interface (FUI) as a graphical user interface (GUI)"… yep you got it again, a *phooey-gooey*.

In addition to icons, OOUI's use "glyphs." These are graphical objects are controlled by programs to perform specific functions. The best examples of glyphs are in screen savers. The little dog, man or monster roaming across your screen which "destroys" it (sends it to black) piece by piece is actually a "screen object" expression of a computer program. In a GIS/GPS package, the blinking dot or arrow depicting a vehicle's movement utilize glyphs that roam across a registered GIS map, responding to realtime GPS positioning. In advanced systems, such as air traffic control, the glyphs are often intelligent. When two graphical objects (airplanes) come within a specified distance of each other, they can blink rapidly or change color. In short, OOUI's icons and glyphs encapsulate program parameters, control and linkages.

Just about every GIS out there uses a phooey-gooey interface, which must make them all object-oriented. Technically it does, so you had better ask a few more questions to differentiate the nature and degree of OO-ness. *OOPS* technology uses "widgets" in the development of computer code. Visual Basic and Visual C are examples of object-oriented programming systems. These packages allow a programmer to generate code by simply flowcharting a program's logic. At each "drafting" step, a widget (flowcharting object) is moved into place and a dialog box is popped-up. Completion of the dialog box writes the appropriately coded gibberish defining the step to a file.

Whereas an OOUI dialog box is independent, an OOPS dialog box is one in a linked set of commands depicted in the flowchart. Most modern GIS display at least a minimal level of OOPS-ness. By turning on the "command log" or "macro builder," a series of dialog box entries can be stored to a file, which in turn, can be rerun to repeat the processing. A robust OOPS, however, uses rigorous flowcharting structure, rules and mechanics to "graphically" assemble a computer application. From this perspective, OOPS is an effective programmer's tool, as it provides an easier way (thank heavens) to develop fully structured computer programs.

The OOPS flowchart captures a succinct diagram of an application's logic, as well as actual code. As GIS moves from descriptive geo-query applications to prescriptive modeling, the communication of logic becomes increasingly important. An insightful

end-user doesn't want to simply behold colorful map renderings of elk habitat or retail sales competition analysis—the gospel from mount GIS. Visualization of a mapped result quickly turns to discussion of the assumptions and expressions used in its derivation. The focus becomes one of cognitive space, as much as geographic space. The OOPS flowchart provides a mechanism for both communicating and interacting with model logic.

This topic was discussed in detail several issues ago (see Author's Note) within the contexts of a "humane GIS interface" and a "dynamic map pedigree." The discussion noted that the next generation of GIS will allow users to "interrogate and interact" with a model logic by simply mouse-clicking on the widgets (boxes and arrows) forming a model's flowchart, or more aptly termed, the modeled map's pedigree. If you take exception with the calibration of a GIS model (e.g., "Hey, slope isn't that important to elk habitat"), then click on those disagreeable portions of the flowchart, change them, and rerun the model. You can compare your "personalized" version with those of others to see the spatial impact of the different cognitive perspectives.

The full incorporation of a robust OOPS makes the transition of GIS from simply a "data-painter" to an interactive "spatial spreadsheet." Several GIS vendors are already well on their way. In fact, I'll bet you lunch that within a couple of years all GIS will have implemented OOPS—be careful, don't lose your lunch over the onset of *oo*.

***Author's Note**: GeoWorld November and December, 1993).*

Spatial Objects—the Parse and Parcel of GIS?

(GeoWorld, October 1996, pg. 28)

(return to top of Topic)

Last section suggested that there are three basic types of "object-oriented-ness." The distinctions between an OOUI (*object-oriented user interface*) and an OOPS (*object-oriented programming system*) were tackled. That leaves the third type of "OO-ness," object-oriented database management (*OODBM*, pronounced ooD'BUM) for scrutiny this time. The OODBM concept provides database procedures for establishing and relating spatial objects, more generally referred to as "data objects." Early DBM systems used "flat files," which were nothing more than long lists of data. The records, or data items, were fairly independent (usually just alpha-numerically sorted) and items in separate files were totally independent.

With the advent of "relational databases," procedures were developed for robust internal referencing of data items, and more importantly, indexing among separate data sets. OODBM, the next evolutionary (or quite possibly, revolutionary) step, extends direct data indexing by establishing procedural rules that relate data. The rules are used to develop a new database structure that interconnects data entries and greatly facilitates

data queries. They are a natural extension of our current concerns for data standards and contemporary concepts of data dictionaries and metadata (their discussion reserved for another article).

The OODBM rules fall into four, somewhat semantically-challenging categories: *encapsulation, polymorphism, structure,* and *inheritance*. Before we tackle the details, keep in mind that "object-oriented" is actually a set of ideas about data organization, not software. It involves coupling of data and the processing procedures that act on it into a single package, or *data object*. In an OODBM system, this *data/procedure* bundle is handled as a single unit; whereas the data/procedure link in a traditional database system is handled by separate programs.

It can be argued (easily) that traditional external programs are cumbersome, static and difficult to maintain— they incrementally satisfy the current needs of an organization in a patchwork fashion. We can all relate to horror stories about corporate databases that have as many versions as the organization has departments (with 10 programmers struggling to keep each version afloat). The very real needs to minimize data redundancy and establish data consistency are the driving forces behind OODBM. Its application to GIS is simply one potential expression of the looming radical changes in database technology.

Encapsulation is OODBM's cornerstone concept. It refers to the bundling of the data elements and their processing for specific functions. For example, a map feature has a set of data elements (coordinates, unique identifier, thematic attributes). Also it can have a set of basic procedures (formally termed "methods") it recognizes and responds to, such as "calculate your dimensions." Whenever the data/procedure bundle is accessed, *"its method is executed on its data elements.*" When the "calculate your dimensions" method is coupled with a polygonal feature, perimeter and area are returned; with a linear feature, length is returned; and, with a point feature, nothing is returned.

If the database were extended to include 3-dimensional features, the same data/procedure bundle would work, and surface area and volume would be returned… an overhaul of the entire application code isn't needed, just an update to the data object procedure. If the data object is used a hundred times in various database applications, one OODBM fix fixes them all.

The related concept of *polymorphism* refers to the fact that the same data/procedure bundle can have different results depending on the methods that are executed. In the "calculate your dimensions" example, different lines of code are activated depending on the type of map feature. The "hidden code" in the data/procedure bundle first checks the type of feature, then automatically calls the appropriate routine to calculate the results… the user simply applies the object to any feature and it sorts out the details.

The third category involves *class structure* as a means of organizing data objects. A "class" refers to a set of objects that are identical in form, but have different specific

expressions, formally termed "instances." For example, a linear feature class might include such diverse instances as roads, streams, and powerlines. Each instance in a class can occur several times in the database (viz. a set of roads) and a single database can contain several different instances of the same class. All of the instances, however, must share the common characteristics of the class and all must execute the set of procedures associated with the class.

Subclasses include all of the data characteristics and procedural methods of the parent class, augmented by some specifications of their own. For example, the linear feature class might include the subclass "vehicle transportation lines" with the extended data characteristic of "number of potholes." Whereas the general class procedure "calculate your dimensions" is valid for all linear feature instances, the subclass procedure of "calculate your potholes per mile" is invalid for streams and powerlines.

The hierarchical ordering embedded in class structure leads to the fourth rule of OODBM— *subclass inheritance*. It describes the condition that each subclass adheres to all the characteristics and procedures of the classes above them. Inheritance from parent classes can occur along a single path, resulting in an inheritance pattern similar to a tree branch. Or, it can occur along multiple paths involving any number of parent classes, resulting in complex pattern similar to a neural network.

In general, complex structure/inheritance patterns embed more information and commonality in data objects which translates into tremendous gains in database performance (happy computer). However, designing and implementing complex OODBM systems are extremely difficult and technical tasks (unhappy user). Next month we'll checkout the reality, potential and pitfalls of object-oriented databases as specifically applied to GIS… good or bad idea; inevitable step or technical fantasy?

Does Anyone Object?

(GeoWorld, November 1996, pg. 26)

(return to top of Topic)

The last couple of sections dealt with concepts and procedures surrounding the three forms of object-oriented (OO) technology—*OUI*, *OOPS* and *OODBM*. An object-oriented user interface (OOUI) uses point-and-click *icons* in place of command lines or menu selections to launch repetitive procedures. It is the basis of the modern graphical user interface. An object-oriented programming system (OOPS) uses flowcharting *widgets* to graphically portray programming steps which, in turn, generate structured computer code. It is revolutionizing computer programming—if you can flowchart, you can program. Both OOUI and OOPS are natural extensions to GIS which make it easier to use (a necessity if users are really going to use GIS).

Object-oriented database management (OODBM), on the other hand, represents a radical change in traditional GIS data structuring. Whereas OOUI and OOPS are actual or near

realities in most contemporary GISs, OODBM is in its infancy. In fact, some users might even question if it is a natural extension. Heck, some might even object.

The OOBDM concept provides database procedures for establishing and relating data *objects*. Last issue discussed the four important elements of this new approach (*encapsulation*, *polymorphism*, *class structure*, and *inheritance*). As a quick overview, consider the classical OODBM example. When one thinks of a house, a discrete entity comes to mind. It's an object placed in space, whose color, design and materials might differ, but it's there and you can touch it (spatial object). It has some walls and a roof that separates it from the air and dirt surrounding it. Also, it shares some characteristics with all other houses—rooms with common purposes. It likely has a kitchen, which in turn has a sink, which in turn has a facet.

Each of these items are discrete objects which share both spatial and functional relationships. For example, if you encounter a facet in space you are likely going to find it connected to a sink. These relational objects can be found not only in a kitchen, but a bathroom as well. In an OODBM, the data (e.g., facet) and relationships (e.g., rooms with sinks therefore facets) are handled as a single unit, or *data/procedure bundle*. Within a GIS implementation, one "hit to disk" you know where and what an object is, as well as its context and linkages.

Such a system (if and when it is implemented) is well suited for organizing the spatial objects comprising a GIS. For example, we know that all airports must have a road connecting it to the highway network. In an OODBM, this connection is part of the data/procedure bundle. In a traditional "layered" GIS, you must overlay a map of all roads with a map of all municipal facilities, then select the airport/road coincidence to identify the connection.

For years GIS users have preempted this process by constructing a single, huge "vector composite" which overlays all of the maps within a data base. Once constructed, a simple database command selects the "regions" (i.e., the "wee-little" polygons formed by the intersection of the multitude of lines) which meets any conceivable query. In a sense, the vector composite approach results in a OODBM-like system based on spatial coincidence. It generates a complete set of spatial objects with a direct data base link to their characteristics. The objects might not be readily recognizable (e.g., a timber stand will be "broken" into pieces representing different soil and slope conditions you can't see), but the GIS can easily respond to your queries involving the coincidence of any map layers.

The main problem of both OOBDM and its "vector composite" cousin comes from their static and discrete nature. With OODBM, the specification of "all encompassing rules" is monumental. If only a few data items are used, this task is doable. However, within the sphere of a "corporate" GIS, the rule set geometrically expands and soon becomes unmanageable. The "vector composite" technique provides a mechanism to automatically generate at least one dimension for defining objects (spatial coincidence).

However, each time a layer changes, a new composite must be generated. This can be a real hassle if you have a lot of layers and they change a lot. New data base "patching" procedures (map feature-based) hold promise for generating the updates for just the areas affected, rather than overlaying the entire set of layers.

Even if the "mechanics" of applying OODBM to GIS can be streamlined, problems still exist. First, the approach assumes all spatial objects are discrete—bounded by well-defined edges. While this might be the case for a house or a road, it's a bit more "fuzzy" for soils (uncertainty) and barometric pressure (continuous surface). In fact, a "fuzzy object" itself seems a contradiction. Secondly, many GIS applications involve cognitive expressions as much as they involve spatial objects. For example, areas of good elk habitat or high probability of sales aren't things you bump into on a walk, but interpretations of spatial relationships.

Most often, these applications involve interactive processing as users run several "what if…" scenarios. From this perspective, habitat and sales maps aren't composed of spatial objects as much they are iterative expressions of differing opinions and experience (spatial spreadsheet). For OODBM to work and garnish data accessing efficiencies there must be universal agreement on the elements and paradigms used in establishing fixed data objects. In one sense, the logic of a GIS model for habitat or sales can be equated to an OODBM rule as it carries both the data (base map ingredients) and procedure (command macro recipe) defining something.

In fact, GIS applications are like banana-bread. If we all agree on the recipe, then it can be stored as an object; however, if everyone insists on adding varying amounts of ingredients and/or entirely new ingredients, then banana-bread is not a "fully structured object." In these instances you need a cupboard full of directly accessible basic ingredients (i.e., base map layers). In short, object-oriented data management holds promise for increased efficiencies the handling of spatially discrete data (for descriptive inventory and mapping), while *process-oriented* GIS modeling systems (for prescriptive analysis and spatial reasoning) gives you "*a license to color outside the lines.*"

But another less obvious impediment hindered progress. As the comic strip character Pogo might say, "we have found the enemy and it's us." By their very nature, the large GIS shops established to collect, nurture and process spatial data intimidated their potential customers. The required professional sacrifice at the GIS altar "down the hall and to the right" kept the herds of dormant users away. GIS was more often seen within an organization as an adversary for corporate support (a.k.a., money pit) than as a new and powerful capability one could use to improve workflow and address complex issues in entirely new ways.

The 90's saw both the data logjam and GIS mystique erode. As Windows-based mapping packages appeared on individuals' desks, awareness of the importance of spatial data and its potential applications flourished. Direct electronic access enabled users to

visualize their data without a GIS expert as co-pilot. For many the thrill of "visualizing mapped data" rivaled that of their first weekend with the car after the learner's permit.

So where are we now? Has the role of GIS developers been extinguished, or merely evolved once again? Like a Power Rangers transformer, software development has taken two forms that blend the 70's and 80's roles. These states are the direct result of changes in software programming approaches in general and "object-oriented" programming in particular.

MapInfo's MapX and ESRI's MapObjects are tangible GIS examples of this new era. These packages are functional libraries that contain individual map processing operations. In many ways they are similar to their GIS toolbox predecessors, except they conform to general programming standards of interoperability, thereby enabling them to be linked easily to the wealth of non-GIS programs.

Like using a Lego set, application developers can apply the "building blocks" to construct specific solutions, such as a real estate application that integrates a multiple listing geo-query with a pinch of spatial analysis, a dab of spreadsheet simulation, a splash of chart plotting and a sprinkle of report generation. In this instance, GIS functionality simply becomes one of the ingredients of a solution not the entire recipe.

In its early stages, GIS required "bootstrap" programming of each operation and was the domain of the computer specialist. The arrival of the GIS toolbox and macro languages allowed an application specialist to develop software that tracked the spatial context of a problem. Today we have *computer specialists* generating functional libraries and *application specialists* assembling the bits of software from a variety of sources to tailor comprehensive solutions.

The distinction between computer and application specialist isn't so much their roles, as it is characteristics of the combined product. From a user's perspective the entire character of a GIS dramatically changes. The look-and-feel evolves from a generic map-centric view to one of a few tailored buttons that walk users through analysis steps that are germane to an application. Instead of presenting users with a generalized set of map processing operations as a maze of buttons, toggles and pull-down menus, only the relevant ones are integrated into the software solution. Seamless links to non-spatial programming "objects," such as pre-processing and post-processing functions, are automatically made.

As the future of GIS unfolds, it will be viewed less as a distinct activity and more as a key element in a thought process. No longer will users "break shrink-wrap" on stand-alone GIS systems. They simply will use GIS capabilities within an application and likely unaware of the underlying functional libraries. GIS technology will finally come into its own by becoming simply part of the fabric of software solutions.

Observe the Evolving GIS Mind Set

(GeoWorld, March 1999, pg. 24-25)

(return to top of Topic)

A couple of seemingly ordinary events got me thinking about the evolution of GIS. It's obvious that technical advances, particularly in GPS and desktop mapping, have profoundly changed how we collect, process and interact with spatial data. However, changes in the community and mindset of GIS'ers are less obvious, yet might prove to be even more dramatic.

I recently attended an open house for a local GIS company that had outgrown its start-up digs. Two things had a significant impact on me— I hardly knew anyone and the conversations almost exclusively focused on data collection, storage and display (aside from the obligatory weather, sports and Bill/Monica discussions). The GIS community has grown (that's for certain) and the diversity of participants is a major factor accompanying its evolution. Whereas a few years ago, I would have known everyone at a GIS function (Fort Collins is still a pretty small town), the select set of "insiders" has been augmented (replaced?) by hordes of GIS users.

The expanded community has brought a refreshing sense of practicality and realism. The days of "GIS-ing for GIS sake," basic research and focus on general tools have given way to application-specific needs and constraints. A decade ago, "proof-of-concept" projects ruled and given a year and a topographic sheet-sized area, anything was possible. Today, operational systems are the focus and trickle-sized data sets have grown into an insatiable torrent of data flows. With the conversion of paper maps, resurrection of remote sensing data sources, GPS-linked data collection, geo-coded addressing of traditionally non-spatial data, and the accessibility of all these data over the Internet, GIS has more of a database technology character than that of a mapping science. The old adage that "a picture is worth a thousand words" has quickly become "an image is composed of millions of records." However, the full worth of a GIS image, in many cases, has yet to be determined.

That brings up the other event that got me thinking. It was an email that posed an interesting question…

> *"We are trying to solve a problem in land use design using a raster-based GIS (ESRI ArcView Spatial Analyst) to no avail. It has to do with the conflict resolution step of the problem. In this step, multiple raster layers are overlaid to produce an output grid depicting the most suitable land use based on where the maximum value was found. Hopefully the attached (see figure 1) should render our problem transparently clear."*

The problem involves "map-*ematical* reasoning" since there isn't a button called "identify the most suitable land use" in any of the GIS systems I know. Note that each of the grid cells in the maps (Residential, Golf Course and Conservation) contains a value identifying its relative suitability (higher value indicates more suitable). A human would

simply determine the highest value for each grid location, note its data layer and color the cell appropriately (e.g., top-left cell would be 76, Residential, red; bottom-right cell would be 87, Conservation, green).

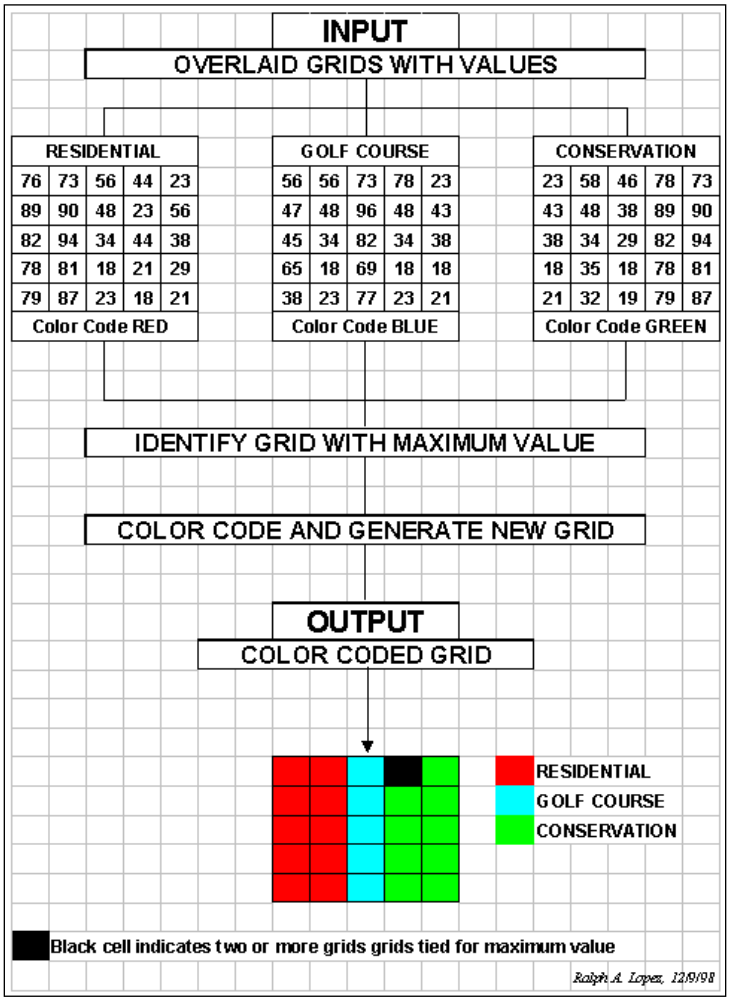


Figure 1. Schematic of the problem to identify the most suitable land use for each location from a set of grid layers.

That's simple for you, but computers and GIS systems don't have the same logical reasoning skills and have to slog-around, ankle-deep in the numbers. For example, one solution (there's others) might be expressed as…

Step 1. Find the maximum value at each grid location on the set of input maps—

CMD: compute Residential_Map maximum Golf_Map maximum Conservation_Map for Max_Value_Map

Step 2. Compute the difference between an input map and the Max_Value_Map—

CMD: Compute Residential_Map minus Max_Value_Map for Residential_Difference_Map

Step 3. Reclassify the difference map to isolate locations where the input map value is equal to the maximum value of the map set (renumber maps using a binary progression; 1, 2, 4, 8, 16, etc.)—

CMD: Renumber Residential_Difference_Map for Residential_Max1_Map
assigning 0 to -10000 thru -1 *(any negative number; residential less than max_value)*
assigning 1 to 0 *(residential value equals max_value)*

Step 4. Repeat steps 2-3 for the other input maps—

(Golf_Max2_Map using 2 and Conservation_Max4_Map using 4 to identify areas of maximum suitability for each grid layer)

Step 5. Combine individual "maximum" maps and label the "solution" map—

CMD: compute Residential_Max1_Map plus Golf_max2_Map plus Conservation_Max4_Map
for Suitable_Landuse_Map

CMD: label Suitable_Landuse_Map

1 Residential	(1 + 0 + 0)
2 Golf Course	(0 + 2 + 0)
4 Conservation	(0 + 0 + 4)
3 Residential and Golf Course Tie	(1 + 2 + 0)
5 Residential and Conservation Tie	(1 + 0 + 4)
6 Golf Course and Conservation Tie	(0 + 2 + 4)
7 Residential, Golf Course and Conservation Tie	(1 + 2 + 4)

(Note: the sum of a binary progression of numbers assigns a unique value to all possible combinations)

OK, how many of you map-*ematically* reasoned the above solution, or something like it? Or thought of extensions, like a procedure that would identify exactly "how suitable" the most suitable land use is (info is locked in the Max_Value_Map; 76 for the top-left cell and 87 for the bottom right cell). Or generating a map that indicates how much more suitable the maximum land use is for each cell (the info is locked in the individual Difference_Maps; 56-76= -20 for Golf as the runner up in the top-left cell). Or thought of how you might derive a map that indicates how variable the land use suitabilities are for each location (info is locked in the input maps; calculate the coefficient of variation [[stdev/mean]*100] for each grid cell).

This brings me back to the original discussion. It's true that the rapid growth of GIS has greatly extended the community of users and certainly made terra-bytes of spatial data a mouse-click away. It has democratized the technology and brought practicality and realism into the equation. In effect, the evolution has made the spatial technologies (GIS, GPS and remote sensing) household terms and a near necessity in the modern workplace.

However, in many instances the focus has shifted from the analysis-centric perspective of the original "insiders" to a data-centric one shared by a diverse set of users. As a

result, the bulk of current applications involve spatially-aggregated thematic mapping and geo-query verses the site-specific models of the previous era. This is good, as finally, the stage is set for a quantum leap in the application of GIS. We have data, we have users, and we have tools. What remains is a pervasive awareness of spatial reasoning— a new way of thinking with maps. The next epoch of the GIS evolution will change the GIS mindset as much as the previous ones have changed our tools and data sets.

(return to top of Topic)

(Back to the Table of Contents)