EMAPS: AN EXTENDABLE, OBJECT-ORIENTED GIS

Stephen M. Ervin, Associate Professor
Department of Landscape Architecture
Harvard University Graduate School of Design
48 Quincy St. Cambridge MA 02138 USA
(617) 495 2682 FAX (617) 495 5015
email: servin@gsd.harvard.edu

ABSTRACT

EMAPS is an interactive interpreted object-oriented environment for manipulating and displaying geographic information in raster (grid-cell-based) form. A hierarchy of objects -- windows, maps, cells -- takes advantage of the inheritance of properties in the Common LISP Object System to allow easy prototyping and exploration of variations. In this system, cell values need not be integers or real numbers -- they may be arbitrary functions, allowing each cell to operate as a cellular automaton. Writing procedures using rows, columns, cell-values and neighborhoods is a powerful way of expressing cartographic modelling operations, including map overlay, image processing, and hill-climbing algorithms; an object-based windowing system makes the graphical interface friendly and flexible. EMAPS is particularly valuable for modelling and exploring dynamic interactions in landscape ecology, and should be useful for teaching basic concepts of programming geographic information sytems.

INTRODUCTION

Most geographic information systems (GIS's) provide tools for the storage, manipulation and diplay of geographic information, whether in raster (grid-cell) or vector/polygon form. The better of them come with "languages" or macro-interpreters enabling encapsulation of procedures and supporting the process of 'cartographic modelling' [Tomlin] for performing analyses. Most GIS languages, however, are not envisioned as programming environments with the usual associated documentation, editing and debugging tools, or other facilities. Their macro languages are typically not constructed with great attention to their denotational semantics -- they are typically *ad hoc*, extended and revised from version to version -- and sometimes lack such basic constructs as conditional control, iteration, or scoping of variables. Thus cartographic models are limited to those (mostly linear) sequences which are supported by the macro language, combined with any 'black-box' pre-packaged analyses provided by the GIS itself. These are often unsatisfactory.

On the other hand, environmental designers -- landscape architects, urban designers, regional planners -- and other professionals -- geographers, ecologists, et al -- who are interested in modelling are likely to resist the prospect of having to learn 'to program' in order to pursue their work. In order to support inventive, unrestricted use of modelling using GIS technology, a middle-ground is desirable, which both provides the rigorous and supportive environment enjoyed by software engineers,

and yet is built around the basic contructs of geographic analysis - maps, regions, values, cartographic modelling primitives.

EMAPS is just such a system: an interactive interpreted environment for manipulating and displaying geographic information in raster (grid-cell-based) form that is based on the principles of object-oriented design, and uses Common LISP as its underlying macro language. The object-oriented design provides a clear conceptual model and provides all the usual advantages of modularity and extendability; the use of LISP provides semantic rigor and a comfortable, productive programming environment for undertaking explorations in geographic analysis and design.

OBJECT-ORIENTED APPROACH

In object-oriented programming, such as the Common LISP Object System (CLOS) [Steele, Keene], or SMALLTALK [Goldberg], two principal ideas underlie all programs: methods for 'encapsulation' of both structure and behavior into 'objects', which maybe organized hierarchically such that some kinds of objects are generic (classes) while others instantiate or specialize the genus (instances); and 'message passing' as the means of communication between objects, such that all objects have an effective 'communications interface' specifying the messages to which they respond, and how they respond. Both of these characteristics may be found in traditional programming languages to greater or lesser degree, and may be thought of as simply implemented by procedure-calls in such languages.

The idea of object-oriented GIS is simple and appropriate: maps are objects with behavior, and a GIS or a cartographic model is simply a collection of specified maps with specified relationships between them. Object-oriented GIS may as well be vector-based as raster-based, or indeed, hybrid of the two. The system described here is primarily raster-based. A hierarchy of objects in EMAPS -- windows, maps, cells -- takes advantage of the properties of object-oriented Common LISP to allow easy prototyping and exploration of variations. The list of objects in EMAPS starts out small:

A WINDOW is an object capable of performing computer graphics for display, with characteristics such as size, colors, bit-depth, etc.

A MAP is an object representing the spatial distribution of some attribute(s), that is capable of displaying itself in a WINDOW; each MAP has a list of characteristics such as name, thematic content, scope, scale and resolution, etc. Each MAP is also linked to a KEY.

A KEY is an object that basically implements a two-column table, providing a correlation between a value or range of values and a display attribute, such as a symbol, color, pattern, etc. The KEY is used when a MAP is displayed in a WINDOW.

A CELL is an object which represents a spatial extent on the ground and encodes an attribute or attrbutes for that extent. Each cell has a

'value', and one or more methods for displaying itself, and reporting, updating or changing its value.

In EMAPS, each MAP is composed of a number of CELLS, typically arranged in a rectangular two-dimensional array as in an ordinary raster GIS map. Each cell has one or more value(s), typically numeric, which are typically displayed by means of a color or pattern on CRT or hardcopy.

It's important to note, however, that in EMAPS, CELL values are in fact *procedures*, not simply values. In its general form, a CELL is an object which can respond to the request 'What is your value?' by invoking its associated procedure, perhaps with appropriate arguments (and depending on the time and form of the request the answer may even vary.) In this way, each cell is in fact a 'cellular automoton': an object with a state which may change as some function of a number of inputs, typically derived from other automota's states. Cellular automata provide a reasonably well-understood general model [Toffoli & Margoulis] with considerable potential for managing and exploring geographic information and environmental phenomena [Itami, Hogeweg].

EXTENDABLE CARTOGRAPHIC MODELLING

EMAPS is well suited to developing models involving geographic data, since the cellular automaton model embedded in object-oriented maps provides for all the inter-map (overlaying multiple layers) and intra-map (spatial, proximity, neighborhood and region) operators typically desired in a GIS, but the full expressive power of LISP is availabale as well. In the Map Algebra [Tomlin] and many other raster-modelling languages (and image-processing in general), there is an implicit 'For all Rows; For All Columns; Do...' nested iteration construct, which, for example, creates a new map as a sum of two input maps by simply adding each pair of cells in like locations in the input maps to produce the value in the output map. This is usually hard coded, in Pascal, Fortran, C, etc., and provides a useful and reliable base for forming cartographic models. In pseudo-code, each of the n-ary (inter-map, 'overlay or combination') operations may be characterized by the following algorithm:

```
FOR I := 1 TO NROWS
  FOR J := 1 TO NCOLS
    NEWMAP [I, J] := F (INPUTMAP1 [I, J], INPUTMAP2[I, J], ...)
```

where the function F maybe any of a dozen or so standard operations such as sum, product, min, max, etc.

Each of the unary (intra-map, 'spatial, neighborhood or proximity') operations may be characterized by the variant algorithm:

```
FOR I := 1 TO NROWS
  FOR J := 1 TO NCOLS
    NEWMAP [I,J] := F (NEIGHBORHOOD (INPUTMAP1 [I, J]))
```

where the function F is as before, and the function NEIGHBORHOOD returns a list of cell-values derived from a variety of standard options (circular or

square neighborhood of specified radius, all cells with similar values or values within a specified range, etc.)

In EMAPS, these two algorithms are built-in, available to the user through a top-level loop which requires only writing a LISP expression which serves as the function F, and which may use any of a number of pre-defined procedures for the NEIGHBORHOOD function. In these functions, each of the indices i,j and the cell-value at [i,j] is explicitly available as a special (global) variable. That is, in EMAPS the reserved words ROW, COL, and VAL may be used in the definition of both F and NEIGHBORHOOD, and will be evaluated by each cell appropriately.

In the object oriented approach, the algorithms become some variant of:

```
FOR I := 1 TO NROWS
  FOR J := 1 TO NCOLS
    (ASK NEWMAP
         (ASK (CELL  I J)
              (SET-VALUE  (F  (ASK  MAP1
                                    (ASK  (CELL I J)  (GET-VALUE))) ... )))
```

The last line, written now in pseudo-LISP syntax [1], indicates the hierarchical order of message passing, in which each map asks each cell to set- or get- its value. The function (CELL I J) returns the cell-object at location [i,j]; the function F is still any ordinary LISP expression. In fact, in EMAPS, a variety of macros provide access to a variety of different forms of the function F. For example, to set the contents of MAP3 to the sum of MAP1 and MAP2, the expression is:

```
(ASK MAP3  (UPDATE  (ARRAY-SWEEP  (MAP+  MAP1 MAP2))))
```

MAP+ is just a macro which expands into the longer expression

```
(+ (ASK MAP1 (ASK (CELL ROW COL) (CELL-VALUE)))
   (ASK MAP2 (ASK (CELL ROW COL) (CELL-VALUE))))
```

+ is just the ordinary addition function, except that it has been modified to not produce errors when it encounters a non-numeric value, but simply to return the special value NA, or 'No Value'. (All the mathematical functions have been so modified, so that no errors occur from mixing numeric- and non-numeric cell values; this has the desirable side-effect of propagating 'NoValue' in all mathematical expressions, a useful facility in many models. Having such a special value is a great benefit in many data analysis operations; the ability to simply incorporate one is due to the flexibility of the underlying LISP language.)

UPDATE is a message defined for all maps, which says in effect, "set all your cells to the values indicated."

---

[1] In these examples I have used a simplified 'Object Lisp' syntax, of the form (ask object (expr)) which simply means evaluate 'expr' in the environment of 'object'. If expr is of the form (proc args), then this is approximately equivalent to the Common Lisp Object System (CLOS) syntax: (proc object args), assuming proc has been duly defined as a method specialized for object [Keene].

ARRAY-SWEEP is the term in EMAPS for the iterative construct in the algorithms above. (While it is the most commonly used general case, the alternative SPREADING-ACTIVATION function, discussed below, is also available.)

The various standard map operations are all available or easily written as LISP functions: for example, the operation COVER has the following form:

```
(DEFINE-MAP-FUNCTION COVER (MAP1 MAP2)
      (ARRAY-SWEEP
            (LET ((VAL1 (ASK MAP1 (ASK (CELL ROW COL) (GET-VALUE))))
                  (VAL2 (ASK MAP2 (ASK (CELL ROW COL) (GET-VALUE)))))
             (IF (= VAL1 0) VAL2
                            VAL1))
```

Without either assuming or explaining all of the niceties of LISP syntax, suffice it to say that the above procedure defines a map operation which takes two maps as input, and generates a new map such that, for each cell: if the value in the first input map is non-zero, that value is put in the new map, and if the first map value is zero, then the value of the second map is put into the output map (literally 'covering' the second map with the first, where zero is 'transparent')

In the above examples, we have seen ROW and COL used as special variables, successivly bound to all combinations of I,J in the iterative construct, and used only as arguments to the (CELL) function to specify a particular cell. But they can be used in any legal LISP expression, including simply standing alone:

```
(ASK MAP1 (UPDATE (ARRAY-SWEEP ROW)))
```

will generate a map which has all 1's in the first row, 2's in the second row, and so on (numerically, this might represent a tilted plane, tilted down toward the user, ranging in elevation from to nrows.)

Similarly, the special variable VAL can be used, as shorthand for

```
(ASK (CELL I J) (GET-VALUE));
```

the expression

```
(ASK MAP1 (UPDATE (ARRAY-SWEEP (SQRT VAL))))
```

will take the square root of MAP1, i.e. replace each cell value with its square root. Finally, we may even use a constant instead of any variables:

```
(ASK MAP1 (UPDATE (ARRAY-SWEEP 7.5 )))
```

will produce a 'constant' map, all of whose cells have the value 7.5.

All of the invariant part of the expression (ASK MAP1 (UPDATE (ARRAY-SWEEP ... can be wrapped up inside 'syntactic sugar' [Abelson & Sussman], and replaced by the expression (SWEEP ...) (or even placed on an interactive menu system, so all the user has to type is the essential part of the expression.) Thus, the command (SWEEP (+ ROW COL)) will produce another 'tilted plane' map, this one tilted from northwest to southeast, ranging in elevation from 2 to (+ nrows ncols).

The neighborhood functions are also easily defined: the square neighborhood of eight cells immediately adjacent to (CELL I J) is represented by the list

```
( (CELL (1- I) (1- J)) (CELL (1- I) J)  (CELL (1- I) (1+ J))
  (CELL I (1- J))                       (CELL I (1+ J))
  (CELL (1+ I) (1- J)) (CELL (1+ I) J) (CELL (1+ I) (1+ J)))
```

(This is true for an interior cell 1< I < NROWS, 1< J< NCOLS; some proper strategy must be defined for handling of edge conditions.) If the function (SQUARE-NBRS I J R) returns the list of values in the square neigborhood of diameter R (R ODD, R>2), then the expression
(SWEEP (/ (SUM (SQUARE-NBRS ROW COL 3)) 8)) will result in 'smoothing' the map, replacing each cell with the average of its eight nearest neighbors (summing their values, and dividing the sum by eight). A variety of other smoothing and 'image processing' operations such as convolutions can be simply defined in the same way.

Circular or square neighborhoods are the commonest and most easily implemented, but the cellular-automaton model suggest that we might want to be able to specify variants. Indeed, for some purposes it may be desirable for cells to be 'wired' in completely arbitrary connections. In the object-oriented approach, this is easily accomplished: each may mat optioonally specialize the generic 'neighbors' function(s), to return a specialized list of neighbors in each case. The highlevel (SWEEP ...) command remains the same, but the result will depend on the details of the neighborhood interconnections. For example, in some smoothing operations, certain cell values should be considered fixed and not be smoothed; these should simply have their neighbor function redefined to consider only themselves (or, more preciseley, to return their value multiplied by 8, since the smoothing function above divides by 8). More speculatively, we might imagine different kinds of cells, some of whom ignore their neighbors altogether, some of whom are influnced by a relatively small radius, others by a larger radius. A community of such cells could provide a model of certain kinds of peer-pressure in neighborhoods, or variations in species sensitivity to pollutants.

We may also wish to model cells as 'emitters', rather than 'receivers', where the diameter of the neighborhood of influence is not determined by the receiving cell, but rather by the emitting cell. In these cases, and in other scenarios, it is useful to have an alternative to the ARRAY-SWEEP approach which must consider all NROWS X NCOLS cells. If, for example, some cell represents an object which has just emitted a cloud of radioactive smoke, an iterative process which starts there and spreads according to some algorithm away from the source might be more efficient than an array sweep (so long as it is known that distant effects are nil).

For these kinds of models, EMAPS provides the (SPREADING-ACTIVATION ...) macro, which takes as its argument a single cell-position or list of positions, and runs the 'inverse neighborhood' function for each source cell ("In whose neighborhood is this cell included?") iteratively (or recursively). Thus, if there is a global vector (such as a wind speed and

direction) that determines neighborhoods, a plume effect can be modelled; if there is a concentric decay function, that too can be modelled. In this approach, EMAPS continues to cycle until all neighbor lists come up empty, or off the edge of the map, effectively providing an animation of the model.

Hill climbing (or descending) algorithms are also easily implemented in this same style. In fact, in the object-oriented approach, we simply introduce a new kind of object - a CLIMBER. This object interacts with maps and cells, querying surrounding cells for elevation values, choosing a path, and leaving a trail. Whether the climber is responsible for any graphics to be displayed in the window, or cells are responsible for marking the passage of the climber, is a responsibility-allocation (design) question that has no single correct answer. One benefit of the modularity of the object-oriented design approach is that neither maps nor cells must change with the introduction of new types of objects; on the other hand, once those objects are recognized as useful parts of a system, some system redesign may be called for.

CONCLUSIONS

Environmental design and planning (the set of disciplines including architecture, landscape architecture, urban design and planning among others) demands the construction and testing of explicit models of complex and dynamic systems. Traditionally, most often, those models have been constructed in the form of drawings or physical scale models, and tested by visual inspection. These models are eprforce static and over-simplified. Computer-assisted techniques for design and planning models at the larger-than-site scale, epitomized by 'GIS' systems, suffer from much the same limitations. These systems for managing and manipulating spatial information, whether in vector- or cell-based format, are designed primarily to manage multiple overlays of descriptive data and produce analyses of a statistical nature; the integration of these analyses into a synthetic design or planning process is left outside of the system.

Some models are now routinely computed in the course of environmental design -- stuctural and thermal, financial, economic and other calculations performed in stand-alone applications or general-purpose spreadsheets are common. The integration of these models with drawing (CAD) or mapping (GIS) systems, especially across different scales of resolution and abstraction, is ad-hoc at best, impossible or incomplete much of the time, and fails dramatically at worst. Some solutions have been proposed and attempted: inter-application data communication is an area of intense research and development, based largely on the definition of standards.

The approach advocated here is not meant to deny the benefits of standards and inter-process communication, but describes an alternative in the form of an integrated environment for design and planning that extends the domain of grid-cell based data. The declaration of object boundaries, names and class-instance relationships is a major part of design knowledge, especially in schematic and preliminary design stages.

Object-oriented programming systems that provide static object descriptions and hierarchies, and that restrict or forbid dynamic restructuring of links, will not serve in a design environment, in which the ability to add and delete parent- and child-links at any time is an essential part of design development and model exploration.

The structure and behavior of EMAPS is also particularly conducive, I believe, to the development of an approach to "Designing with Maps". Designing with Maps requires a different approach than either traditional GIS or CAD systems: more flexible, speculative and prescriptive than GIS; more abstract and geographic than CAD. A traditional description of designing has it composed of analysis, synthesis and evaluation; EMAPS supports the synthesis stage better than traditional GIS modelling packages which were designed primarily for analysis and evaluation. As part of an ongoing inquiry into models of computer aided design, including maps and other modes of representation, EMAPS is being interfaced with other sofware in a heterogeneous network. LISP is especially conducive to this sort of integration, as it enables the control of other applications and operating systems from within a single, flexible programming environment.

EMAPS is particularly valuable for modelling and exploring dynamic interactions in landscape ecology, and it should be useful for teaching basic concepts of programming geographic information sytems. In conjunction with other software and systems, such as for three-dimensional renderings and animations, EMAPS provides the kind of flexible, extensible, systematic modelling 'testbed' required for the next generation of computer-aided environmental design.

REFERENCES

Goldberg, Adele & David Robson, 1989. *SmallTalk-80 The Language,* Reading, MA: Addison Wesley Publishing

Hogeweg, P. 1988. "Cellular Automata as a Paradigm for Ecological Modeling", *Applied Mathematics and Computation* 27:81-100 (1988)

Itami, R.M. 1988. "Cellular Automatons as a Framework for Dynamic Simulations in Geographic Information Systems", in *Proceedings, GIS/LIS,* American Society for Photogrammetry and Remote Sensing, Falls Church VA 1988, pp. 590-597.

Keene, Sonya. 1989 *Object-Oriented Programming in COMMON LISP: A Programmers Guide to CLOS.* Reading, MA: Addison Wesley Publishing

Steele, G.L. Jr. 1990. *Common Lisp The Language,Second Edition,* Bedford, MA: Digital Press, 1990

Toffoli, T. and N Margolus, 1987 *Cellular automata machines : a new environment for modeling.* Cambridge, Mass. : MIT Press, 1987.

Tomlin, D. 1990. *Geographic Information Systems and Cartographic Modelling,* Prentice-Hall, 1990.

ACKNOWLEDGMENTS:

The development of EMAPS has been supported by an equipment grant from the Milton Fund of Harvard University and by the excellent software development environment offered by Macintosh Common Lisp.