Map Calculus in GIS: a proposal and demonstration

Mordechai (Muki) Haklay
Department of Geomatic Engineering
University College London (UCL)
Gower Street, London, WC1E 6BT
Phone: (+44) 20 7679 2745, Fax: (+44) 7380 0453, Email: m.haklay@ucl.ac.uk

Abstract

This paper provides a new representation for fields (continuous surfaces) in Geographical Information Systems (GIS), based on the notion of spatial functions and their combinations. Following Tomlin's (1990) Map Algebra, the term "Map Calculus" is used for this new representation. In Map Calculus, GIS layers are stored as functions, and new layers can be created by combinations of other functions. This paper explains the principles of Map Calculus and demonstrates the creation of function-based layers and their supporting management mechanism. The proposal is based on Church's (1941) Lambda Calculus and elements of functional computer languages (such as Lisp or Scheme).

Introduction

The use of computers for the analysis and representation of geographical information is now, arguably, approaching its fifth decade and the representation and storage of geographical information in computers seems to have matured and stabilised. Within the range of different geographical abstractions, Goodchild's overview (1993) identifies two major groups – field models, which represent the geographical space as a continuous "field" or "surface" (in other words, the analysis of phenomena as being continuous across space), and object models that represent discreet entities in space. A recent GIS textbook (Longley *et al.,* 2001, p. 145) lists six common field representations in a GIS: regularly-spaced sample points, irregularly-spaced sample points, rectangular grid cells (raster), irregularly-shaped polygons, triangular irregular network (TINs) and polylines which represent contours. Some other variations of the field model have been proposed over the years, like Tobler's (1976) vector fields, or combinations of models, such as Tomlin's (1990) storage of lineal data within a raster. As Unwin (1981) notes, the field model is based on the mathematical concept of a *scalar field*, which can be represented using the following formula:

$$z = f(x, y)$$

in which for every location, denoted by the co-ordinates x and y, there is a function f that yields the value z. The function f is a mathematical function which operates on different locations; it can also be termed a "spatial function". The extension of this form to a higher dimension is possible with the general form of:

$$z = f(location)$$

where location describes a set of co-ordinates.

There are two main classes of functional descriptions of geographical phenomena: piecewise and global. *Piecewise* functions are defined over sub-domains, which are constructed through a tessellation of the field. A good example for this is Triangular Irregular Network (TIN), where a set of linear functions describe the shape of the area. Piecewise functions are characterised by a set of constraints and limitations on the function's continuity at sub-domain boundaries. *Global* functions are defined in the same way across the domain. The Euclidean distance function from a point or the Inverse Distance Weighting (IDW) function are examples for global functions. The same function is used for every location in space.

In recent years, there has been new interest in radically different approaches to the representation of fields in GIS. While the foundations of these approaches can be traced to the 1970s and earlier, the capabilities of current computers and software enable their implementation. For example, Wood (1998) and Rana and Morley (2002) have made advances in the consideration of surface networks to represent topography based on concepts suggested by Pfaltz (1976); Cheng and Molenaar (1999) explored the representation of fuzzy objects in a GIS, linking their work to Burrough (1989); while Frank (1999) has suggested the development of algebra to analytically manipulate geographical objects, an approach that is somewhat similar to the one suggested in the ROSE algebra (Guting & Schneider, 1995). Such approaches and suggestions for geographical representations may deal with a specific domain of geographical objects, for example the relationship between surface networks and topography, or deal with the more general representational issues, as in the case of the ROSE algebra, and the representation of fuzzy objects. On a more theoretical level, Miller and Wentz (2003) made the explicit link between representational frameworks and their use in GIS and spatial analysis. In other disciplines that deal with solutions to partial differential equations (such as oceanography or atmospheric science), fields are represented using

piecewise polynomial functions over triangular and quadrilateral partitions while in computer graphic applications and, especially for applications of rendering algorithms, there is a wide use of splines to represent fields.

Many applications of GIS utilise field models, but it is in the area of spatial analysis that the similarity between the surface model and a well-defined mathematical function is the clearest. This is due to the nature of spatial analysis where the user explores the formal, quantitative structure of geographical problems (Longley & Batty, 1996). Furthermore, within socio-economic research, it is the need to translate data from one set of areal units to another that provides the motivation to explore different types of surfaces (Martin, 1996). As Thurstain-Goodwin (2003) demonstrated, data surfaces are valuable policy tools when they are used to depict socio-economic data sets.

This paper describes a new way to store surface information, by using function-based layers in a GIS. A function-based layer is defined by its mathematical and spatial function and a number of bounded variables, while the system takes care of representing and manipulating it in a way that is transparent to the user. Unlike current representations, the GIS stores the function itself and not spatial objects which contain the z value in the formula above. Such an approach existed in computer models in meteorology for many years (Goodman, 1985) and is being used in some global climate models, but was not adopted in GIS and spatial analysis. The main strength of the new representation is the ability to treat each analytical layer in its symbolic form, thus making it possible to manipulate layers (maps) through calculations of functions, and manipulates GIS layers in a formulaic form. This can increase the range of the GIS analytical toolbox and open up new directions in spatial analysis research. Following Tomlin's (1990) Map Algebra, the term Map Calculus is used here to describe the application of function-based layers in a GIS. In essence, Map Calculus is an alternative to the current practice of using raster layers to represent surfaces. By using Map Calculus and taking advantage of the capabilities of existing and emerging computing environments, some of the problems that are associated with the raster-based approach are eliminated: there is no need to select a specific cell resolution or to consider the storage of large raster layers.

The paper is divided into four parts. First, Map Calculus is explained in detail. Second, the computational considerations and theoretical concepts that are material to this approach are described. Third, a simple implementation of Map Calculus in a modern desktop GIS is presented. Fourth, the proposed approach is compared to the current

practice of using raster layers to simulate surfaces. The paper is completed by drawing conclusions and pointing to future research directions.

Throughout this paper, the term "Map Calculus-enabled GIS" is used to describe a GIS that has been enhanced with the required procedures and data structures to handle function-based layers. This paper envisages Map Calculus capabilities as being an extension to raster and vector-based representations and, while a GIS that is based on Map Calculus is theoretically possible, the discussion in this paper will focus on a hybrid implementation where function-based layers extend the vector and raster capabilities of a common GIS.

Map Calculus in a GIS

A GIS that can handle Map Calculus is somewhat equivalent to mathematical or statistical software suites (such as Matlab™ or SAS™). Such a system should have the capabilities to store and calculate functions in real time, while storing information about these functions (such as variable values) in a form that facilitates fast and efficient computation. As previously mentioned, function-based layers are especially suited as an alternative to the representation of continuous surfaces, which are based on global functions, as raster layers. To understand how function-based layers works, it is best to look at three examples – first, a simple distance function; second, a spatial interpolation based on IDW, and finally a Digital Elevation Model (DEM).

A distance function from a given location (A) with the co-ordinates (x_A, y_A) is currently represented in GIS packages as a raster layer with an arbitrary resolution, where each cell is assigned a value that represents the distance from the centre of the pixel to A. In a function-based layer, this can be represented by a single formula – naturally, the formula for the Euclidian distance for any given location of B from A[1]:

$$ThisFarA = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

---

[1] ThisFar is taken directly from Tomlin (1990) and while Tomlin discusses the layer as the product of ThisFar operation, here the function itself is stored.

It is important to note that in this function, the values of x_A and y_A are assigned to the co-ordinates of A, while x_B and y_B are left unassigned until we choose the location B. For example, if A' is at the location (124,64) then *ThisFarA'* will be stored as the function:

$$ThisFarA' = \sqrt{(x_B - 124)^2 + (y_B - 64)^2}$$

Of course, Euclidean distance is one of the simplest spatial functions. Spatial interpolation, on the other hand, is more complex process and more interesting aspects of Map Calculus-based GIS can be clarified by examining its implementation. In IDW, the function operates on a set of sampled points $(L_1, L_2, \ldots L_n)$ and calculate the value for a new location L' by calculating:

$$L' = \frac{\sum_{i=1}^{n} \frac{1}{d_i^{\ p}} L_i}{\sum_{i=1}^{n} \frac{1}{d_i^{\ p}}}$$

Where d_i is the distance from L' to the location L_i, and p is a power of the distance. Usually, the search radius is taken as a parameter of the function. In a function-based layer implementation, the GIS will store the template for IDW, the defined radius distance and a linkage to the set $(L_1, L_2, \ldots L_n)$. Upon request to calculate the IDW value for a location L', buffer operation on the basis of the search radius extracts the points that should be included in the calculation and constructs the sub-set. This is then followed by the computation of the equation for L' and the sub-set which yields the requested output. The process can be repeated for any set of points. For example, it is possible to analyse IDW values from the set $(L_1, L_2, \ldots L_n)$, but to calculate them only for location $(K_1, K_2, \ldots K_n)$. This will enable the linkage of two sample sets, by calculating a precise estimate of the field that the data set L represent for the locations of the set K. Notably, the user interface of the GIS for entering the parameters can stay the same.

The two examples above are of global functions where the implementation of Map Calculus-enabled GIS is rather straight forward. Piecewise functions, however, represent a different challenge. This is the case with DEMs where it is impossible to devise a global function and the description of the field is done through tessellation and a use of a family of functions that are fitted to each sub-domain, as is the case with finite elements methods (Lancaster & Salkauskas, 1986). Therefore, piecewise representations can be

stored as an array of functions, where each function is stored with a direct reference to the domain over which it is defined. This will require the GIS to store the tessellation as part of the function, with a template for the function family. As noted, TIN representations of fields are implementing such an approach: a set of tessellated triangles, with linear functions that are defined across them. How much a function-based representation improves the current methods of DEM representation is a matter for research.

In general, a Map Calculus-enabled GIS will hold the templates for various spatial functions, which can be local (like distance), neighbourhood and connectivity functions (Tomlin, 1990; Samet, 1995). Such a template, as in the example above, will enable the storage and calculation of distance from a given point, a set of points, lines or polygons. More sophisticated templates will be needed for spatial analysis techniques such as Bi-cubic interpolation, Kriging (Oliver & Webster, 1990), or Geographically Weighted Regression (Brunsdon, Fotheringham & Charlton, 1998), although the development of the latter functions might be challenging. These templates will be used to instantiate a function which assigns values to some of the variables and prepares the system to rapidly calculate the surface value for a required location or locations. Thus, they might include references to the set of points that are being used as the source of spatial interpolation, as well as other parameters that are specific to each spatial function, such as the minimum number of points that participate in the interpolation.

A Map Calculus-enabled GIS should also support mathematical operations between function-based layers and between function-based layers and other representations of geographical objects, such as rasters. The user should be able to construct sophisticated models by stringing together layers and by setting a variety of mathematical operations which can operate on a single functional layer (unary operators) or between layers. Of course, the development of operations on functional layers follows the work of Tomlin (1990) and Berry (1993a) on Map Algebra and Cartographic Modelling. Function-based layer are very efficient in their storage and even with complex and sophisticated functions, the storage space required is significantly smaller than that of multiple rasters layers, due to the symbolic nature of functional representations. While the cost of digital storage has reduced dramatically in recent years (Figure 1), the storage and management of multiple rasters is still a technical and practical problem. ArcGIS, for example, cannot

manage rasters larger then 2.147 gigabytes or about 23,000x23,000 cells of random floating point numbers (ESRI, 2002).
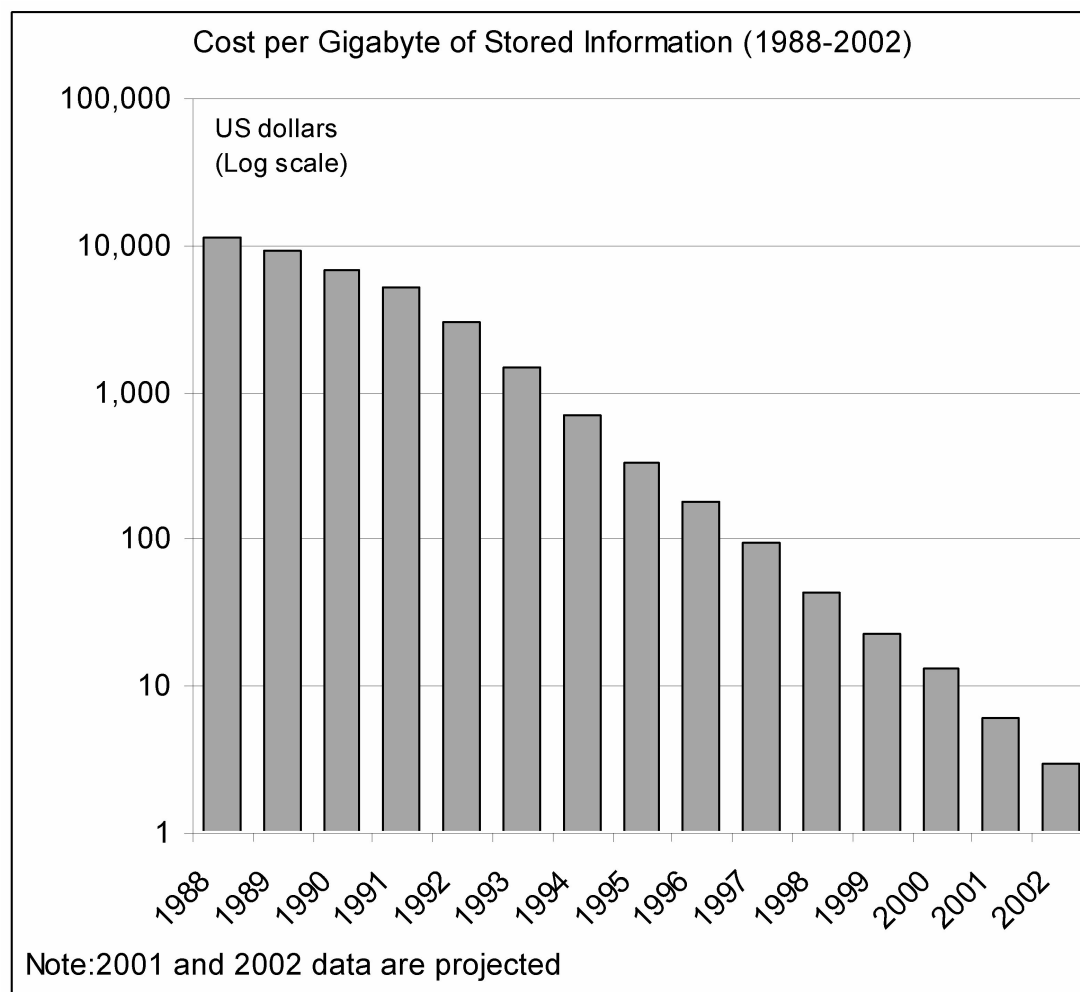

Figure 1: Cost per gigabyte of stored information: 1988-2002 (source: NSF, 2002)

The storage and representation of function-based layers is, of course, incomplete without the ability to visualise them. An important aspect of any modern GIS is its ability to visualise surfaces and images rapidly and efficiently. In a Map Calculus-enabled GIS, this can be achieved quite easily and, arguably, in a more transparent way then current visualisation of raster layers. Due to the way in which computer displays are constructed, a matrix of values must be created in the computer's memory before sending it to the screen. This requires that the GIS will calculate the value of (possibly complex) function for each pixel of the screen. However, as this is done instantaneously, the GIS can use parameters such as screen resolution and the current scale of the map that the user is viewing to minimise the amount of calculations. For example, if a user uses a common screen resolution of 1024x768, then the effective area of the map in a common GIS

package (such as ArcGIS) is about 800x600 pixels – due to the elements of the Graphical User Interface which occupy the rest of the screen. This active map area can further be divided to basic units of 2x2 pixels (usually, one pixel is too small to be visible). Thus, the active area requires about 120,000 calculations. The speed of the calculations is the most significant factor in making Map Calculus-enabled GIS useful and effective for its users. This can be accomplished by the use of efficient and rapid algorithms. As the user zooms out or in, the scale of the map changes, and new calculations for the currently displayed area are carried out. Hence, to the system's user, a Map Calculus-enabled GIS behaves at the same way as the existing implementations of GIS.

Another interesting aspect of function-based layers is the ability to extend the GIS analytical toolbox into a numerical and mathematical analysis of the functions themselves. For example, in a complex distance function (where we use anisotropic functions, which measure a weighted distance from several locations) there is an advantage in calculating and locating local and global minima. This can be carried out by analysing the behaviour of the function across the space, in a similar way to operation of packages such as Matlab™. When dealing with more complex functions, where it is impossible to determine local minima and maxima, the visualisation of the layer can be used as an indicator of a user's interest in a specific area and, while calculating the values for visualisation, the system can find and store "interesting locations" across the map. Furthermore, using free Central Processing Unit (CPU) cycles, the system can continue and investigate these interesting locations further and they can be visualised and stored.

A function-based layer should also store its relationship to the objects that are used as input. Thus, in the example of the distance function used above, the function-based layer should not just store the co-ordinates of the point that was used to calculate it, but should be linked to the spatial objects in the database. Such an implementation will enable the function-based layer (and all subsequent layers that use it) to be updated instantly whenever the location of a point is being updated. This can be exemplified by a GIS that tracks the distance amongst a set of vehicles, all of which carry Global Positioning Satellite (GPS) receivers and transmit their locations to the GIS. The combined layer (which might calculate the location of the "centre of gravity" amongst all the vehicles) is updated automatically whenever the location of a vehicle is updated.

Finally, Map Calculus-enabled GIS needs to have certain management capabilities due to the dependency between layers. In the example of the tracking function if the user

decides to stop tracking one of the vehicles, this must spread through the "layer tree" to each function that uses the altered one. The GIS can maintain links forward – from a function-based layer towards composite layers that represent mathematical operations in which the function participates – and backwards. There are also requirements to manage the memory efficiently and to deal with issues such as garbage collection, as function-based layers will generate more computations and temporary data structures, thereby making memory management critical to the performance of the computer.

This section has provided a description of the main elements of Map Calculus-enabled GIS. The next section considers the computational concepts that underpin the creation and management of such a GIS, and explains why the implementation of such a GIS is feasible today.

Concepts and consideration for the implementation of Map Calculus-enabled GIS

Many of the building blocks of Map Calculus-enabled GIS, both theoretically and practically, are based on the field of functional computer languages and are used in many applications. Hence, the development of the underlying mechanisms to support Map Calculus-enabled GIS should not represent a major theoretical challenge. This section begins with a discussion of the main theoretical elements that enable the creation of such a GIS: functional computer languages and the Lambda Calculus. After discussing these aspects, some of the computational elements required to implement Map Calculus-enabled GIS are explored. These include the ability to manage relationships amongst functions, the ability to carry out the numerical analysis of mathematical functions with computers, and the need for a computing environment that supports rapid and complex computations.

Functional Computer Languages and Lambda Calculus

To enable a GIS to handle and manipulate function-based layers, it must be capable of creating functions from other functions and hold "suspended" functions, where some of the variables have been assigned while the rest will be evaluated at run-time. These capabilities exist in functional computer languages, and originate from Church's (1941) Lambda Calculus. This section provides a brief description of the main principles of the Lambda Calculus and functional languages that are relevant for the current discussion; it

is largely based on Hudak's (1989) excellent overview, which should serve any interested reader as an introduction to these topics.

The Lambda Calculus was presented by Alonzo Church (1941) and, although not explicitly aimed at the development of computer languages (as none existed at that time), has had lasting influence, especially in the field of computer languages. The Lambda Calculus is mainly concerned with the ability to represent functions as a set of symbols and with providing rules for their manipulation. Using few simple notations and rules, the calculus provides a powerful set of tools to describe functions, function applications (where the free variables have been substituted with specific values) and recursions. Together with the Turing machine, it has influenced the creation of computer languages and compilers (Rosser, 1982; Gunter, 1992). As Hudak (1989) noted, most of the languages in use in the Geographical Information Science (GISc) field (such as C, C++, Pascal or Java) use many of the properties of the Lambda Calculus, albeit implicitly. These types of language are known as *imperative languages* as they rely on an implicit state which is modified through commands that are usually applied in a sequence. For example, the assignment in C "a = 1" binds the value 1 to the variable a and this state is used in subsequent commands, such as a++. In contrast, *declarative languages*, such as Prolog, do not have implicit states and programming is carried out by using expressions.

Functional languages use functions as a computational model. They are characterised by the use of an explicit state and the reliance on recursion to accomplish iterations (loops) (Hudak, 1989). The first functional language, Lisp, was invented by J. McCarthy in the late 1950s, although it must be noted that the influence of the Lambda Calculus on its original concepts was minimal (Stoyan, 1984). Subsequent implementations of Lisp, such as Scheme, have a clearer link to the theoretical grounding of Lambda Calculus and are used throughout the world to teach functional languages. However, for the purists, Lisp and Scheme are not "right" as they enable assignments and have other aspects that "contaminate" the implementation. Far less popular languages such as Haskel, Miranda or ML are considered "pure" functional languages. The following snippet of Scheme code demonstrates how functions that measure the length of a polyline would look in a functional language:

```
(define power2
  (lambda (x) (* x x))
```

```
)
(define distance
   (lambda (x1 y1 x2 y2)
      (sqrt (+ (power2 (- x2 x1)) (power2 (- y2 y1)))))
   )
)
(define length
   (lambda (polyline)
      (if (null? (rest(rest polyline)))
         (distance
            (first(first polyline))(second(first polyline))
            (first(second polyline))(second(second polyline))
         )
         (+ (distance
               (first(first polyline))(second(first polyline))
               (first(second polyline))(second(second polyline))
               )
               (length (rest polyline))
         )
      )
   )
)
```

`power2` and `distance` hold a function and, apart from the use of the "Polish notation" to represent mathematical operations, it is easy to see that they are similar to any other representation of these functions in conventional language. Now, assume that a polyline is represented as a list of co-ordinates in the form ((x1 y1) (x2 y2) (x3 y3) …(xn yn)). The command `first` takes the first element of a list, while `rest` extracts the rest of the list.

Notably, the function here is recursive and the values are not being assigned to temporary variables during the various calls to the function but are used directly, thus the state of the computation is explicit.

Garbage collection and interdependency

The ability to evaluate functions as symbols and the relationship to the Lambda Calculus are not the only elements that should be borrowed from functional computer languages to implement Map Calculus-enabled GIS. Other useful elements include the ability to manage memory and perform "garbage collection".

"Garbage collection" is the examination of computer memory to return allocated but unused blocks of memory to the shared memory used by the application or the operating system. This is required due to the continuous creation of temporary values and the need to manage interdependency amongst functions. In a Map Calculus-enabled GIS, if *f1, f2*

and *f3* are basic functions (i.e. they relate directly to objects in the GIS), then we can have a set of complex composite functions that might include, amongst others:

$$f4 = Minimum(f1,f2,f3)+Maximum(f1,f2,f3)$$

$$f5=f4*f1$$

For many operations of the GIS, such as changes in the underlying data set or deletion of a certain function (for example, *f1*), the system must follow the current set of relationships, alter subsequent functions and manage the computer memory. Since function-based layers rely on extensive computation (during visualisation, numerical analysis, construction of layer trees, etc.) a successful implementation must handle this in an efficient way. Furthermore, the system must identify problems such as circular references, where a function refers to itself. However, as such issues are well established in many computer packages – such as spreadsheets and programming languages – this should not pose an obstacle to implementation.

Functions optimiser

While many functions are simple and straightforward to compute, others represent a computational challenge that can be partially resolved through optimisation of the symbolic representation. For example, the calculation of slope in a common raster-based GIS is carried out as a neighbourhood operation, where the value of the surrounding cells is examined to elicit the value of the current cell. As many have noted, slope is in effect the first derivative of the original surface (Evans, 1972; Berry, 1993b). Thus, it is analytically possible to compute the first derivative from the symbols of the function (a practice that is common in mathematical analysis tools) and then store the resulting function. In case of piecewise functions, the original array of functions with their associated domains will be replaced by new array with the derivatives. Another optimisation can be done on the basis of the characteristics of the computational process itself, in a similar way to query optimisers in SQL-based databases (such as Oracle). For example, if the computed function is:

$$KDE(Set\ X,\ employment)\ +\ Dist(Road\ network)\ +\ KDE(Set\ X,\ revenue)$$

where KDE is a Kernel Density Estimator function, which, in this case are defined on a specific set of points (Set X), and Dist is a distance function from the Road Network (a set of vectors), the optimiser first retrieves the neighbourhood function and calculates

the set of points that is used in the interpolation functions, thus reducing the repeated extraction of points from the database.

Numerical analysis: finding "interesting" locations

One of the fascinating aspects of storing the functions as an element in the GIS (instead of their outputs) is the ability to carry out analysis on the nature of the functions themselves. One such example was noted above – the ability to calculate derivatives by the manipulation of symbols. Furthermore, there are many other elements that are of interest to the analyst and that Map Calculus-enabled GIS can implement better than current raster-based GIS. By linking numerical and symbolic analysis libraries to the functional engine of the GIS, it is possible to identify areas of transition, local and global properties of function either globally, or within an area of interest. Such analysis should be directed by the intrinsic behaviour of the function. For example, in a distance function, the maximum value is neither interesting, nor informative. Thus, when calculating distance, there is no need to store or evaluate maxima.

Computing needs and requirements

Probably the most limiting factor in the creation of Map Calculus-enabled GIS to date is the need to compute and re-compute functions "on the fly", while the user zooms in or out from a specific location, or when the user asks to produce the contour lines from a complex function. Users of modern GIS are accustomed to working with an interactive system that responds within a few seconds. Therefore, there is a need to rapidly compute a complex function such as Kriging, in order to make function-based layers fit into such systems. Furthermore, the computational complexity of layers can vary dramatically, as many users require the use of composite functions, such as the outcome of five or six interpolations from different sets of points. Another aspect of the computing profile of Map Calculus-enabled GIS is that it is characterised by intense peaks in computation, as the user requests a calculation with every zoom in/out operation, followed by periods of inactivity. Finally, Map Calculus-enabled GIS is likely to include utilise numerical-analysis operations, and these type of calculation utilises a lot of computing power.

 However, while only a few years ago it was impractical to suggest the implementation of Map Calculus-enabled GIS, recent advances in computing open up the possibility of implementing such a GIS today. The two most important ones are the increase in

computational capacity of inexpensive computers and the development of advanced distributed computing environments, known as "Grid computing".

The growth of the processing power of modern computers is a well-known phenomenon. Although Gordon Moore predicted the growth in the design complexity of integrated circuits, his "law" became, in effect, the benchmark for the computing industry and pushed forward the design of faster and faster computers (Brown & Duguid, 2000). The emergence of inexpensive and powerful computing environments enabled the development of "Supercomputer" scale machines from standard inexpensive computers, in configurations such as Beowulf clusters (Bell & Gray, 2002). While this classic supercomputing architecture is geared towards sustained peak performances, it is less suitable for Map Calculus-enabled GIS, as it requires peaks of computing, followed by period of minimal activity. This makes the emerging Grid computing (Foster, Kesselman & Tuecke, 2001) a more attractive environment for its development. In Grid computing architectures, a network of computers is configured in such a way that unused or underutilised computing resources are available to those users who need them. Another reason for the adequacy of Grid computing, is that it is based on the distribution of computational problems across the network. As many geographical problems are naturally divisible using a spatial decomposition, they are suitable for distributed computational models. Although knowledge of, and experience with Grid computing is in its early stages, experiences from projects like SETI@Home demonstrate that extensive computations can be performed by distributing the load across the network.

In the next section, a simple implementation of Map Calculus-enabled GIS is described, to demonstrate the concepts and principles that were described above.

Demonstrating Map Calculus-enabled GIS

To demonstrate the feasibility of Map Calculus-enabled GIS, a basic prototype was developed using Manifold System 5.0 (CDA International Ltd., 2001). This GIS package was selected because it enables the use of Perl as a scripting language and, therefore, can be enhanced quite easily using widely available Perl libraries. It is important to note that Perl (Wall & Schwartz, 1991) is not a functional language, but rather an imperative one. However, Perl has hybrid origins, and it was designed by borrowing from a wide range of languages including Lisp (Wall, 1999). Therefore, it is capable of evaluating expressions

and, in effect, implements the Lambda Calculus (Dominus, 1999). It is this property that enabled the use of Perl to imitate the implementation of Map Calculus-enabled GIS. It must be stated that the prototype uses very simple spatial functions (based on distance) and allows only three arithmetic operations (addition, subtraction and multiplication) between pairs of existing layers.

The implementation is based on two scripts, the first runs within Manifold and handles the GUI, while the second runs outside acting as a server. This client/server architecture was selected because it enables the installation of the server script on more powerful machine than that of the client. Furthermore, the server script is more sophisticated and contains some of the functionality that was mentioned in the previous sections. The interface is made up of three separate forms, shown in Figure 2. The first interface (Figure 2a) allows the user to create a new functional layer by selecting a spatial object (point or pair of points) and selecting one of several basic spatial functions. This interface is also used to delete a function from the database. The second interface (Figure 2b) allows the user to define map-algebra operations. The "Map Algebra" dialogue area enables the creation of binary linear combinations of existing layers. Finally, the "display" interface (Figure 2c) can be used to calculate the raster representation that is relevant to the area currently displayed in the main window. The following part will demonstrate some code segments that explain the implementation within the system. Some syntactic "sugar" is used to make the code more readable.

The process of creating a layer is a simple one – the client sends a message to the server that contains the layer identification (a name), the type, and a set of parameters that the server must store in order to recalculate the function. In the case of a distance from a point function, the stored data structure is defined as:
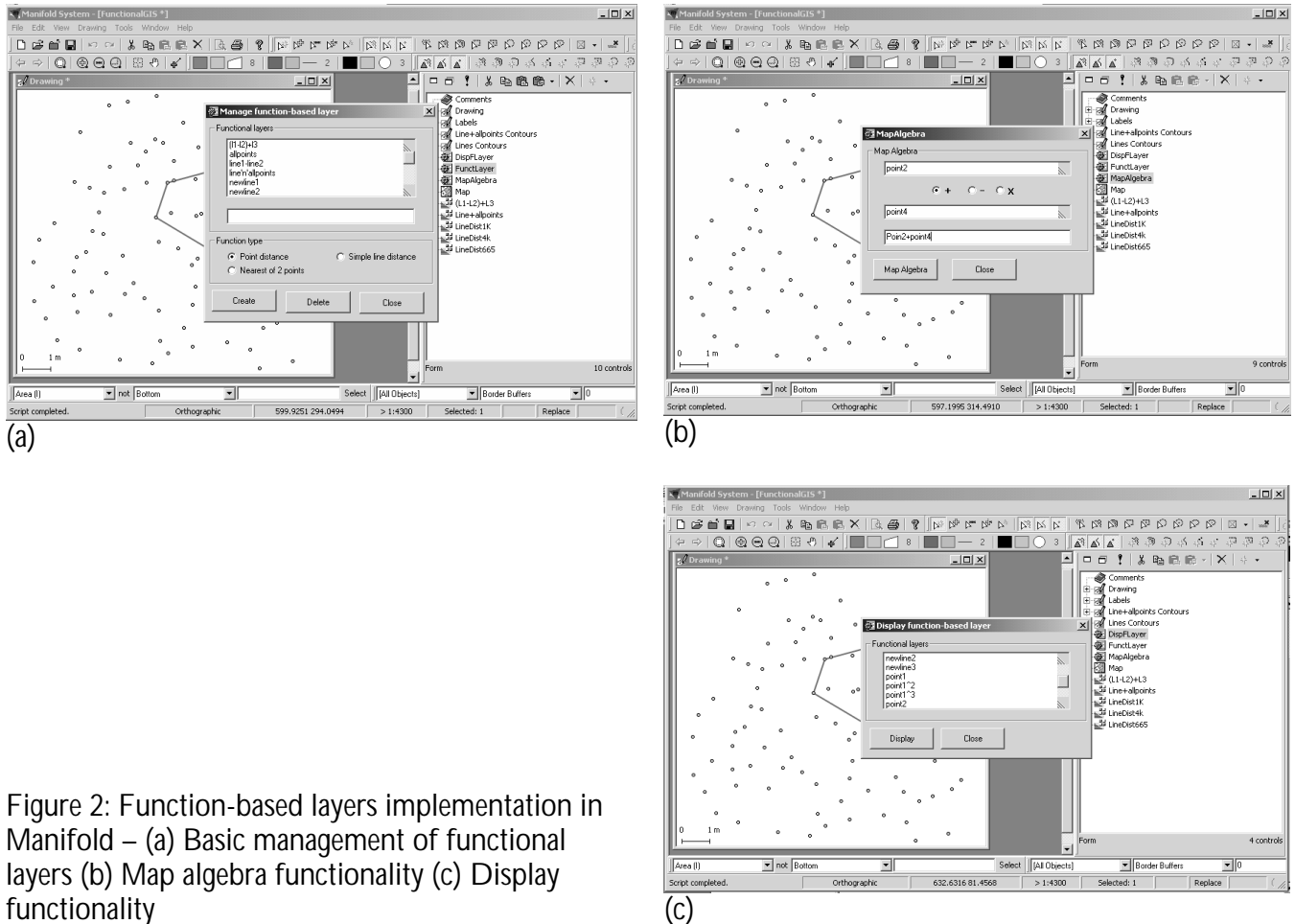
```
$new_layer->{layertype} = 1;
$new_layer->{layerid}   = $LayerID;
$new_layer->{xcoord}    = $x;
$new_layer->{ycoord}    = $y;
```

As we are interested in Euclidean distance the only parameter that is needed is the location of the point from which the measurement is carried out. A map-algebra layer is defined by three parameters – the two layers that are used for calculation, and the mathematical operation:

```
$new_layer->{layertype} = 99;
$new_layer->{layerid}   = $LayerID;
$new_layer->{operation} = $Oper;
$new_layer->{layer1}    = $Layer1;
$new_layer->{layer2}    = $Layer2;
```


(a)

(b)

(c)

Figure 2: Function-based layers implementation in Manifold – (a) Basic management of functional layers (b) Map algebra functionality (c) Display functionality

Note that the management includes a definition of "layer type", as the system must recognise what specific function is defined on this layer. The data structure for the storage of information of each functional layer must be tailored to the specific aspects of this layer – mainly the minimal set of parameters that will enable the recalculation of this layer.

Another interesting element is the construction of the lambda function. For example, using the distance function the code segment is:

```
return "sqrt((\$x-($LayerID ->{xcoord}))**2+(\$y-($LayerID ->
{ycoord}))**2)";
```

This, of course, is a basic Euclidean distance function. However, the syntax `\$x` tells Perl not to evaluate the variable x when constructing the string, but rather to postpone evaluation to a later stage. Thus, for the point (14,-128), this code segment returns the function:

```
sqrt(($x-(14))**2+($y-(-128))**2)
```

When the display function is activated, this segment of code is evaluated for every pair of co-ordinates in the display area, assigning them as `$x` and `$y`. Finally, it is worth noting the way in which composite layers are handled. This is done by recursively bringing up the function that constructs the "Lambda Function" in the following way:

```
return constructFunc($layer2) . " $layer_id->{operation} " .
constructFunc($layer2);
```

Apart from the code, the server produces a raster layer according to the function of the specific combination. The client then reads the file and it is possible to display the result for the area currently displayed. This communication between client and server is grossly inadequate in terms of speed, but it is good enough to demonstrate the visualisation advantages of the approach. Figure 3 provides a demonstration of the output that is expected from a Map Calculus-enabled GIS. Figures 3a-3c emulate the process of zoom-in to a specific segment of the map and the creation of a finer raster for the area of interest that the user has zoomed into. It is important to note that the two rasters that are on display here contain exactly the same number of cells.
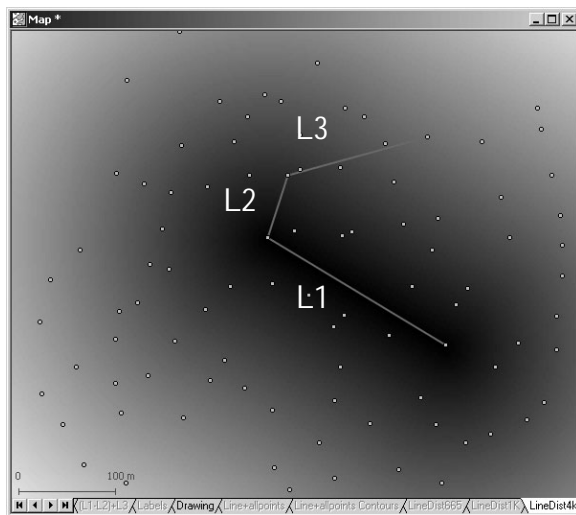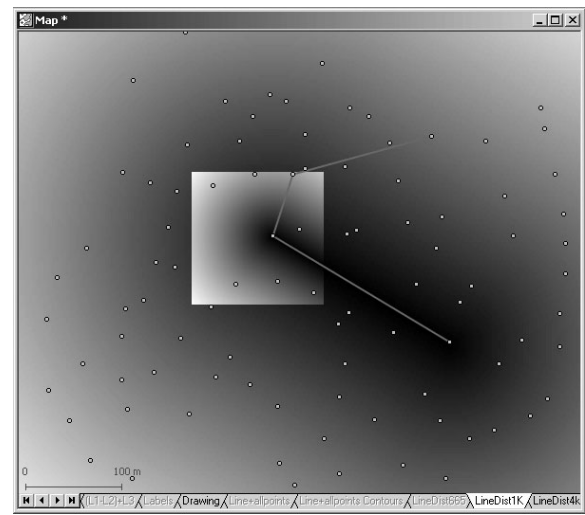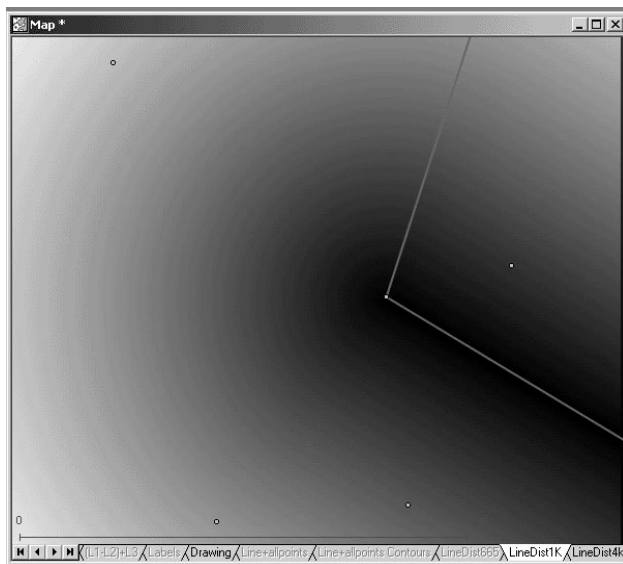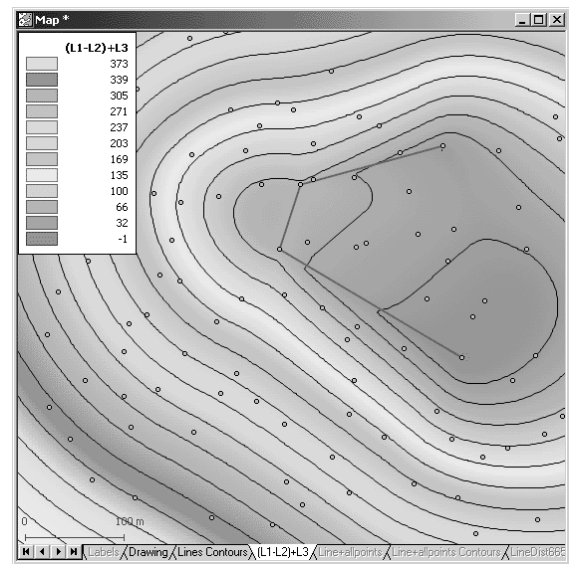
Figure 3: (3A) A polyline, made of segments L1, L2 and L3, is used to demonstrate the capabilities of Map Calculus-enabled GIS. In this image, the distance function is calculated at a scale of about 1:4000. Once the user zooms in to a smaller area (3B), the system calculates the new values of the surface for that specific area, and, as the next image (3C) demonstrates there is no "pixelation" of the surface (note the scale bar). In 3D, the following composite function was calculated: (Dist L1 – Dist L2)+Dist L3. The resulting surface was calculated directly from the final function and not by creating intermediate surfaces.

Positioning function-based layers in the current practice of GIS

When considering a significant departure from the established methods of field representations in a GIS, there is a need to weigh the advantages and disadvantages of the proposed approach and consider them in the light of existing experience. In this section, a few aspects of traditional, raster-based representations are compared with function-based representations. Raster representations are more general and can be

integrated in different systems easily; they are non-volatile, and they have a clear spatial resolution. Functional representations are more precise; they contain their own metadata as well as model lineage explicitly, and directly associate the visualisation with the current display scale. Two aspects are more complex to compare: the issue of visual and analytical resolution and the computational side of their implementations. In these topics, each implementation have some strengths in terms of efficiency and redundancy.

The first advantage of the raster representation is its generality, vividly demonstrated by Tomlin (1990). The same data structure, visualisation functionality, and storage mechanism is applicable to all forms of spatial analysis functions. Regardless of the complexity of the calculation, the output data structure and format is always the same. This makes the raster representation easy to manipulate and subsequent calculations, such as map-algebra operations, are not dependent on the functions that created the raster layers. As discussed earlier, a Map Calculus-enabled GIS must take into account the unique properties of the specific function during construction of the layer and during its visualisation and application. Yet, this is mainly an issue that is tackled during the programming of the GIS and not its use. Therefore, this aspect can be handled in a transparent way to the end-users and should not pose any difficulty or complexity for them.

Secondly, raster representations are non-volatile because they require the full computation to be carried out only once. Once the calculation is completed, visualisation and manipulation of the layers is reduced to simple input-output (I/O) operations. This is probably the major drawback of function-based layers and possibly the reason that they were impractical until computer power had reached scales that made intensive computations possible and cheap. It must be noted that while it is impossible to construct the function that created a raster from the raster itself, a user of Map Calculus-enabled GIS can create a raster at will and use this raster instead of the function.

Thirdly, in the process of creating raster layers the user is usually required to define the spatial resolution of the output layer. In some cases, this can be considered as an advantage and forces the user to consider the nature of the spatial phenomena that the raster captures. Nevertheless, on many occasions the decision on raster resolution is arbitrary and, as higher resolutions lead to exponential growth in the size of the output raster, users tend to make concessions and choose the coarsest resolution that is relevant to their problem. Thus, the use of pixels as spatial units has significant theoretical and

practical implications (Fisher, 1997). In a Map Calculus-enabled GIS, the notion of resolution is meaningless, as the function can compute the values continuously across space. This removes the limitations on precision that are inherent in raster representations – a function-based layer can be displayed at any required precision. For example, if the user zooms in to an area of 10x10 centimetres, the function is calculated on tiny cells, and thus provides a precise presentation which is impractical in a raster-based representation. It is wrong, of course, to claim that this makes function-based layers more *accurate* in all cases, although in some it will provide better accuracy. For example, when calculating absolute minima/maxima, or when calculating the average value of a field within a defined polygon, as Map Calculus-enabled GIS can calculate the value on the basis of an integral. On the other hand, there are classes of spatial functions that have some internal resolution. This is the case in raster implementation of Kernel Density Estimator function, where there is a link between the bandwidth and meaningful cell size. In such cases Map Calculus-enabled GIS can limit the spatial resolution that is displayed to the user, thus making this aspect explicit. For such classes of functions, Map Calculus-enabled GIS can store the spatial resolutions explicitly and use it when a function-based layer interacts with other layers.

As for function-based layers, they have several advantages. First, the presentation is clearer – throughout the life cycle of the layer, the user can interrogate it and view the function that was used to define it as well as its values and parameters. This explicit presentation makes the nature of the layer clearer to the user and removes ambiguities that exist with raster layers where the function and the parameters that created it are usually implicit or, at best, stored in a separated file if the user maintains a highly organised log. This aspect is directly linked to the second advantage of function-based layers – the integrated metadata and model lineage. Due to the interconnectedness of function-based layers and the way in which high-order layers rely on lower level ones, the relationships amongst layers are stored as a practical requirement of GIS management functionality. Within the area of raster-based models, a decade passed between the suggestion of a graphical representation of the modelling process (Berry, 1993a) and its implementation in packages such as Idrisi™, ArcView™ or Erdas Imagine™ (see Bruns & Egenhofer, 1997 for a review of Map Algebra interfaces). Even within these implementations, there is no direct link between the model and the output layer and they mainly geared towards general documentation and process management. It must be noted, as Van Deursen (1995) demonstrated, that it is possible to provide a tighter link

between dynamic models and a standard raster-based GIS environment. PCRaster and its modelling language combine the information about the layer with the steps that created it. Within the framework of Map Calculus-enabled GIS, the model and the layer definition are tightly linked and thus maintained more rigorously.

Finally, the visualisation of Map Calculus-enabled GIS is more explicit. As detailed above, the proposed visualisation algorithm is tightly linked to the computer's display properties and no generalisation occurs. In comparison, when a 10,000x10,000 cell raster is displayed in a common GIS, the system performs unsupervised and undocumented generalisations in order to fit the raster to the pixels on the screen. Although most users are satisfied with the performance of this generalisation, the proposed method is more straightforward and avoids such distortions.

When considering the two representations, special attention must be paid to analytical and visual resolutions. Analytical resolution is the resolution that is used during the analysis process as the computer calculates the raster or performs operations on a raster layer, while visual resolution is being used when displaying a raster layer on the computer's screen. With Map Calculus-enabled GIS, the concept of resolution becomes mainly a visual one, and the system can instruct the computer how to visualise various functions according to limitations or instructions that are integral to the specific operation, as in the case of specific bandwidth in interpolations. When performing map-algebra operations with function-based layers, the analytical resolution does not change or alter. However, when operating on a raster layer, the issue of analytical resolution is highly important, as in the case of operating on two raster layers when each is defined with different cell resolution.

Finally, the issue of efficiency and computational complexity must be considered. At first, it seems clear that Map Calculus-enabled GIS are inherently more computationally intensive than traditional GIS. While this is true with regard to real time calculations, a closer examination of traditional representations demonstrates their batch processing origins and a certain wastage of computer resources. For the following analysis, a modern raster-based spatial analysis project is used (Thurstain-Goodwin & Unwin, 2000). This project relies on a composite model whereby multiple interpolated rasters, created using the KDE algorithm, are normalised and combined to create an integrated layer which is the outcome of the whole model. Within the resulting raster the user defines a specific threshold and the areas containing values that are higher than this threshold receive

special attention. The raster layers in this project were created for the whole of London, an area of 60 km by 60 km, with a spatial resolution of 50m per pixel (the raster size is therefore 1,200x1,200). The number of pixels that falls within the defined threshold is about 23,000 out of 1,440,000 or 1.6% of the total number of pixels. Arguably, this is a massive waste of resources, if not for the insignificant costs of computer calculations. When considering the need to store all the intermediary layers on the computer's disk, this cost becomes visible (although not prohibitive). Of course, it was impossible to know the location of these "interesting areas" before creating and normalising all the input layers and combining them into the final layer. In a Map Calculus-enabled GIS, the model would be presented as a linear combination of a series of KDE functions. Although it might be computationally complicated, the user would be able to reach the output layer directly without the waste of creating intermediate layers. At the same time, there is a need to accept that most users will not create a complicated model without examining the intermediate layers. The computational advantages that are outlined here can only be realised when a model is created time and again in an automatic process. To summarise, the issue of computational efficiency is more complex then it first seems, and only with a real implementation of a Map Calculus-enabled GIS will it be possible to provide a comprehensive evaluation of this aspect.

Conclusions and future directions

In this paper, the concepts and a preliminary implementation of Map Calculus-enabled GIS were presented and demonstrated. The aim here was to demonstrate that the concept is feasible within the current generation of GIS software although, admittedly, the functions used here are rather simplistic and almost trivial. The real challenges in the development of function-based layers are connected to the development of efficient neighbourhood and global operations, such as the various interpolation methods. Further beyond this, the integration of piecewise functions in Map Calculus-enabled GIS will enable the representation of geographical objects such as DEMs or fuzzy classifications of soils.

Based on existing applications of data surfaces (Thurstain-Goodwin, 2003; Lloyd *et al.* 2003), it is likely the spatial analysis of socio-economic variables is the first area of research that will benefit from Map Calculus-enabled GIS. In recent years, modelling efforts focused on the use of disaggregated data and the combination of multiple

interpolated layers. In such models, the ability to manipulate the function and explore the results is important to the researchers (Lloyd *et al.* 2003).

Another area that can benefit from Map Calculus-enabled GIS is environmental modelling and transport modelling, where the GIS deals with complex and continuous processes in space and time. This can be achieved by extending the concepts that were presented here and adding time as a continuous variable. Yet another application area is spatial decision support systems, where functions can be used as building blocks for models, as envisaged by Densham (1994). The relative simplicity in defining such layers can support experimentation with different models within real time interactions of users with a GIS and making such systems more interactive for end users.

To help materialise these future applications, a focused effort is required to create a fully operational Map Calculus-enabled GIS, though it is accepted that some aspects of it are challenging. However, given existing knowledge in computer science about functional languages and the emerging computing environments that may provide the needed computational capacity, it is possible to realise a "Function Analyst" within existing GIS packages in the foreseeable future. In any case, the implementation of Map Calculus-enabled GIS is a challenging prospect, and can reinvigorate an interest in efficient and fit-for-purpose algorithms that can rapidly solve various spatial analysis problems.

Acknowledgments

References

Bell, G., and Gray, J., 2002, What's next in high-performance computing? *Communications of the ACM*, **45**(2), 91-95.

Berry, J. K., 1993a, Cartographic Modeling: The Analytical Capabilities of GIS. In *Environmental Modeling with GIS*, edited by M. F. Goodchild, B. O. Parks and L. T. Steyaert (New York: Oxford University Press), pp. 58-74.

Berry, J. K., 1993b, *Beyond Mapping: Concepts, Algorithms, and issues in GIS* (Fort Collins,Co: GIS World Books).

Brown, J. S., and Duguid, P., 2000, *The Social Life of Information* (Boston: Harvard Business School Press).

Bruns, H. T., and Egenhofer, M. J., 1997, User Interfaces for Map Algebra, *Journal of the Urban and Regional Information Systems Association* **9**(1), 44-45.

Brunsdon, C., Fotheringham, A. S., and Charlton, M.E., 1998, Spatial Nonstationarity and Autoregressive Models, *Environment and Planning A*, **30**, 957-973.

Burrough, P.A., 1989, Fuzzy mathematical methods for soil survey and land evaluation, *Journal of Soil Science*, **40**, 477-492.

Cheng, T., and Molenaar, M., 1999, Objects with fuzzy spatial extent, *Photogrammetric Engineering and Remote Sensing*, **65**(7), 797-801.

Church, A., 1941, *The Calculi of Lambda Conversion* (Princeton: Princeton University Press).

Densham, P. J., 1994, Integrating GIS and spatial modelling: visual interactive modelling and location selection, *Geographical Systems*, **1**(3), 203-219.

Dominus, M. J., 1999, *Perl Contains the Lambda-Calculus* (Available WWW http://perl.plover.com/lambda/ accessed 26th May 2002).

ESRI, 2002, *What is the maximum size a grid can be?* (Available WWW http://support.esri.com/search/kbdocument.asp?dbid=14575 accessed 24th June 2002).

Evans, I. S., 1972, General Geoemorphometry. Derivatives of altitude and descriptive statistics. In *Spatial Analysis in Geomorphology*. Edited by R. J. Chorley (London: Methuen), pp. 17-90.

Fisher, P.F, 1997. The pixel: a snare and a delusion. *International Journal of Remote Sensing* **18**(3), 679-685.

Foster, I. Kesselman, C. Tuecke, S., 2001, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputer Applications*, **15**(3), 200-222.

Frank, A. U., 1999, One step up the abstraction ladder: Combining algebras - From functional pieces to a whole. In *Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'99, Stade, Germany*. Edited by C. Freksa, and D.M. Mark, Lecture Notes in Computer Science, Vol. 1661, (Berlin: Springer-Verlag) pp. 95-107.

Goodchild, M. F., 1993, The State for GIS for Environmental Problem Solving, In *Environmental Modeling with GIS*, edited by M. F. Goodchild, B. O. Parks, and L. T. Steyaert (New York: Oxford University Press), pp. 8-15.

Goodman, A.,1985, Surface Analysis: a Structured Bibliography, Working paper 17, Department of Geography, Monash University (Available WWW http://www.deakin.edu.au/~agoodman/publications/biblio/surfacebiblio.html accessed 1st August 2002).

Gunter, C. A., 1992, *Semantics of Programming Languages: Structures and Techniques* (Cambridge, MA: MIT Press).

Guting, R., and Schneider, M., 1995, Realm-Based Spatial Data Types: The ROSE Algebra, *Journal on Very Large Databases (VLDB)*, **4**, 243-286.

Hudak, P., 1989, Conception, Evolution, and Application of Functional Programming Languages, *ACM Computing Surveys* **21**(3), 359-411.

Lancaster P, Salkauskas K., 1986, *Curve and Surface Fitting: An Introduction* (London: Academic Press).

Lloyd, D., Haklay, M., Thurstain-Goodwin, M. and Tobon, C., 2003, Visualising spatial structure in urban data, in *Advanced Spatial Analysysis*. Edited by P.A. Longley and M. Batty (Redlands: ESRI press). pp. 267-286.

Longley P A, Batty M. (Editors), 1996, *Spatial Analysis: Modelling in a GIS Environment* (New York: Wiley).

Longley, P. A., Goodchild, M. F., Maguire, D. J., Rhind, D. W., 2001, Geographic Information Systems and Science (London: Wiley).

Martin, D., 1996, *Geographic Information Systems – Socioeconomic applications (2nd Ed)* (London: Routledge)

Miller, H. J. and Wentz, E. A., 2003, Representation and Spatial Analysis in Geographic Information System, *Annals of the Association of American Geographers* (In press).

National Science Foundation (NSF), Division of Science Resources Statistics, *Science and Engineering Indicators–2002,*Arlington, VA (NSB 02-01) (April 2002)

Oliver, M.A. and Webster, R., 1990, *Kriging: a method of interpolation for geographical information systems*, IJGIS, **4**(3), 313-332

Pfaltz, J.L., 1976. Surface networks, *Geographical Analysis* **8**,77-93.

Rana, S. and Morley, J., 2002, Surface Networks, Working Paper 43, Centre for Advanced Spatial Analysis, University College London, London.

Rosser, J. B., 1982, Highlights of the history of the lambda-calculus, in *Proceedings of the 1982 ACM symposium on LISP and functional programming*, Pittsburgh, Pennsylvania, United States, pp. 216 – 225.

Samet, H., 1995, *Geographical Information Systems: A Technical Approach*, Lectures Notes, Computer Science Department and Centre for Automoation Research and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland.

Stoyan, H., 1984, Early LISP history (1956 - 1959), in *Proceedings of the 1984 ACM Symposium on LISP and functional programming*, Austin, Texas, United States, pp. 299-310

Thurstain-Goodwin, M. and Unwin, D, 2000, Defining and delimiting the central areas of towns for statistical monitoring using continuous surface representations, *Transactions in GIS*, **4**(4), 305-317.

Thurstain-Goodwin, M., 2003, Data surfaces for a new policy geography, in *Advanced Spatial Analysysis*. Edited by P.A. Longley and M. Batty (Redlands: ESRI press). pp. 145-169.

Tobler, W. R., 1976, Spatial Interaction Patterns, *Journal of Environmental Systems*, **6**, 271-301.

Tomlin, D., 1990, *Geographic Information Systems and Cartographic Modelling* (Englewood Cliffs: Prentice Hall).

Unwin, D., 1981, *Introductory Spatial Analysis* (London: Methuen).

Van Deursen W.P.A., 1995, *Geographical Information Systems and Dynamic Models*. Ph.D. thesis, Utrecht University, NGS Publication 190, 198 pp. Electronically available WWW www.carthago.nl

Wall, L., 1999, Uncultured Perl, Linux Magazine, October 1999 (Available WWW http://www.linux-mag.com/1999-10/toc.html accessed 26th May 2002).

Wall, L., and Schwartz, R., 1991, *Programming Perl* (Sebastopol, CA: O'Reilly & Associates).

Wood, J. D., 1998. Modelling the continuity of Surface form using Digital Elevation Models. In *Proceedings 8th International Symposium on Spatial Data Handling*, Vancouver, Canada, 725-736.