Multidimensional Map Algebra

Jeremy Mennis[1], Jeff Leong[2], and Rahul Khanna[1]

[1]Department of Geography and Urban Studies, Temple University,
Philadelphia, PA 19122
Tel: +1 215-204-4748
FAX: +1 215-204-7833
Email: jmennis@temple.edu

[2]Department of Risk, Insurance, and Healthcare Management, Temple University,
Philadelphia, PA 19122
Tel: +1 215-204-1935
FAX: +1 215-204-7833
Email: jeffleon@temple.edu

[3]Department of Finance, Temple University,
Philadelphia, PA 19122
Tel: +1 215-204-1935
FAX: +1 215-204-7833
Email: rkhanna@temple.edu

Abstract

This paper presents a general data processing language for spatio-temporal data called *multidimensional map algebra* (MMA). Like conventional map algebra, MMA consists of a set of local, focal, zonal, and global data manipulation functions that can be combined to form complex models. MMA functions, however, operate not only on data that are two-dimensional in space but also on data that are: 1) one-dimensional in time, 2) both two-dimensional in space and one-dimensional in time, 3) three-dimensional in space, and 4) both three-dimensional in space and one-dimensional in time. As a proof of concept, these functions are implemented in JAVA as an open source class library. Each function is implemented as a JAVA class, and is overloaded to allow for various combinations of data layers, as well as for function parameterization.

1. Introduction

Despite significant research on spatio-temporal data models (cf. Peuquet, 2001), few commercial geographic information systems (GIS) or remote sensing (RS) software packages offer robust spatio-temporal data handling capabilities. These capabilities are necessary for analyzing the growing volume of spatio-temporal data generated by remote sensing technologies and geocomputational simulations. Consequently, many researchers dealing with spatio-temporal data have opted to craft their own algorithms using development platforms outside the confines of conventional GIS and RS software (e.g. Mennis and Peuquet, 2003; Yuan 2001). These algorithms are often domain specific and hence not easily reused.

Here, we present a general data processing language for spatio-temporal data called *multidimensional map algebra* (MMA). MMA is an extension of conventional map algebra (Tomlin, 1990), an analytical framework for raster data handling implemented in many commercial software packages. Like conventional map algebra, MMA consists of a set of data manipulation functions that can be combined to form complex models. MMA functions, however, operate not only on data that are two-dimensional in space but also on data that are: 1) one-dimensional in time, 2) both two-dimensional in space and one-dimensional in time, 3) three-dimensional in space, and 4) both three-dimensional in space and one-dimensional in time. As a proof of concept, these functions are implemented in JAVA as an open source class library available for public download at http://astro.temple.edu/~jmennis/mma.

2. Extending Conventional Map Algebra to Multiple Dimensions

Conventional map algebra functions are typically grouped into three main categories: local, focal, and zonal functions (DeMers, 2002; Tomlin, 1990). Local and focal functions ingest one or more data layers and produce an output data layer. Local functions compute the value of a grid cell in the output layer based on the value(s) of the analogous cell position in the input layer(s). Focal functions compute the value of a cell in the output layer based on the cell values within a neighborhood of the analogous cell position in the input layer(s). Zonal functions compute the value of a grid cell in the output layer using two layers as input, a value layer and a zone layer, and output a table. For each zone in the zone layer, the function summarizes the values of the analogous cell positions in the value layer. For each type of a function a variety of statistical and other types of operators can be applied. A number of other map algebra can be considered *global* functions, also sometimes referred to as *incremental* functions. Global functions generally ingest one or more data layers and output a data layer in which the value of each grid cell is computed by iterating over the entire grid as, for example, in the 'least cost path' function (Tomlin, 1990).

A number of extensions to map algebra have been proposed for environmental analysis (Pullar, 2001; Wesseling et al., 1996). Some of these efforts have addressed extending map algebra specifically for analysis of three-dimensional data (Neteler, 2004; Scott, 1998). Likewise, research in image processing has also focused on specifying map algebra-like statistical manipulations on three-dimensional data (Nikolaidis and Pitas, 2000).

In our own previous research, we have shown how conventional map algebra can be extended to handle data that are both two-dimensional in space and one-dimensional in time using a space-time 'data cube' approach (Mennis et al., 2005). In this approach, the time dimension is treated as simply a third dimension equivalent to the spatial dimensions and the map algebra algorithms extended accordingly to what we called 'cubic' map algebra functions. For example, whereas a conventional local map algebra function may be conceptualized as the overlay of two registered grids, a cubic local function as applied to two data cubes may be conceptualized as the superposition of two cubes. MMA focal and zonal functions differ from their conventional map algebra analogs in that neighborhoods and zones may be defined over space and/or time in addition to the two-dimensional spatial neighborhoods and zones used in conventional map algebra. The data cube data structure and a limited set of cubic map algebra functions were implemented as a prototype in the image processing scripting language IDL (Research Systems, Inc.). The prototype implementation was tested and validated in the context of case studies focusing on the analysis of climate-vegetation-land cover dynamics in Africa using time series of satellite imagery (Mennis, 2005; Mennis and Viger, 2004).

Here, we extend the initial prototype implementation substantially. First, we have ported the code from IDL to JAVA, an open source development platform, in order to facilitate code sharing and development. Second, we have expanded the data structure and algorithms to handle not only space-time data cubes but also a variety of other temporal, spatial, and spatio-temporal data types. Third, we have developed an object-oriented code design to maximize code reusability and facilitate the continued development of the MMA JAVA library. Fourth, we have developed a standard MMA function syntax which supports the development of new MMA functions.

3. Spatio-Temporal Data Types, Neighborhoods, and Lags

3.1 Data

MMA specifies the following data types for encoding temporal, spatial, and spatio-temporal data:

TimeSeries
Grid
TimeCube
SpaceCube
HyperCube

A TimeSeries maps a set of values to a set of regular temporal positions. A Grid (i.e. a conventional raster) maps a set of values to a set of regularly spaced planimetric positions. A TimeCube maps a set of values to a set of regularly spaced planimetric and temporal positions. A SpaceCube maps a set of values to a set of regularly spaced planimetric and altitudinal positions. A HyperCube maps a set of values to a set of regularly spaced planimetric, altitudinal, and temporal positions. An individual TimeSeries, Grid, TimeCube, SpaceCube, or HyperCube data set is referred to as a *layer*. An individual position in any layer is referred to as an *element*, and each element is associated with a single variable value.

The letters *X*, *Y*, *Z*, and *T* are used in the conventional manner to refer to various spatial and temporal dimensions. The X and Y dimensions of a Grid specify planimetric position, where a [0,0] origin is specified at the lower left corner of the grid and X refers to the east-west axis and Y refers to the north-south axis. The Z dimension refers to the altitudinal axis, with the origin at the lowest altitude, and the T dimension refers to the temporal axis, with the origin at the earliest time. Thus, for example, a Grid element's position can be specified as a [X,Y] coordinate value and a HyperCube element's position can be specified as a [X,Y,Z,T] coordinate value.

Each data type is implemented as a JAVA class with a float array of one or more dimensions as an attribute of the class. Thus, each element's temporal, spatial, or spatio-temporal position in a layer is encoded as its position in the multidimensional array, and the element's value is the value encoded for that array position. The TimeSeries data type is implemented as a one-dimensional array [T], the Grid as a two-dimensional array [X,Y], the TimeCube and SpaceCube as three dimensional arrays [X,Y,T] and [X,Y,Z], respectively, and the HyperCube as a four dimensional array [X,Y,Z,T]. These classes also store information on spatial and temporal referencing and resolution, as well as a reserved value for indicating 'no data.'

3.2 File Reading and Writing

Data may be read into these data structures through file reading and writing methods. The file reading method ingests a header file (named "1.hdr") and one or more ASCII files (named "1.dat"). The header file includes information on the data type (e.g. TimeCube), the number of X, Y, Z, and T positions, the X, Y, Z, and T resolution, the value indicating 'no data,' and the coordinate reference position of the X, Y, Z, and T dimensions. If a TimeSeries is being read, the data file simply reports a value for each temporal position on a separate line in the file. If a Grid is being read, the data file is a matrix of values where each matrix position indicates the position of the value in the Grid. If a SpaceCube is being read, a set of data files is required where each file encodes data for an individual Grid at one altitudinal position. Similarly for a TimeCube, a set of data files is required where each file encodes data for an individual grid at one temporal position. For both SpaceCubes and TimeCubes, the files must be named by number (e.g. "1.dat", "2.dat", etc.) in numeric order from lowest altitude to highest (for a SpaceCube) or earliest time to latest (for a TimeCube). If a HyperCube is being read, the Grids for the first temporal position are numbered from lowest to highest altitude, then the Grids for the second temporal position are numbered from lowest to highest altitude, and so on. The file writer method can also write files that can be directly read by ArcGIS (Environmental Systems Research Institute, Inc.) as an ESRI GRID.

3.3 Neighborhoods

Other data types specified by MMA are used to define a neighborhood. A neighborhood indicates the region considered proximal to a given element position. Neighborhoods are used in conventional map algebra focal functions, where they can take the form of rectangle, circle, or wedge shapes of various sizes. MMA neighborhoods extend these two-dimensional shapes to multiple dimensions, including the following data types, where the beginning of the data type indicates its appropriate dimensionality:

TNeighborhood
XYNeighborhood
XYZNeighborhood
XYTNeighborhood
XYZTNeighborhood

A TNeighborhood is defined simply by a temporal range. XYNeighborhood is extended to represent either a rectangle or circle shape:

XYRectangle
XYCircle

XYZNeighborhood is extended to represent cubic and spherical neighborhoods:

XYZRectangle

XYZCircle

XYTNeighborhood is extended to represent neighborhoods that may best be conceptualized as a space-time cube and space-time cylinder, where the radius of the cylinder reflects a spatial radius and the length-wise axis of the cylinder reflects a temporal range:

XYTRectangle
XYTCircle

Of course, because the spatial and temporal dimensions do not use the same units of measurement (i.e. meters and hours are not equivalent), the user has the option of defining the neighborhood extent independently for the spatial and temporal dimensions.

XYZTNeighborhood is extended to represent extensions of rectangle and circle shapes to four dimensions, three spatial and one temporal:

XYZTRectangle
XYZTCircle

In addition, users can define custom neighborhoods of irregular shapes in space and space-time.

3.4 Lags

A lag is an offset from an element that certain MMA functions can utilize. For example, a focal function may compute on a neighborhood that is not centered on the element position for whose value it is calculating but rather offset a certain distance (or time) away. Lags may be specified for each type of layer depending on whether an offset in time, space, or space time is appropriate:

TLag
XYLag
XYZLag
XYTLag
XYZTLag

4. Functions

Like conventional map algebra, MMA functions can be categorized according to the local, focal, zonal, or global nature of the function. As noted above, each function (with the exception of global functions) also typically applies a mathematical, relational, or statistical calculation. The following calculations are supported in the current implementation of MMA:

Mathematical	Add, Subtract, Multiply, Divide
Comparative	GreaterThan LessThan, EqualTo
Statistical	Minimum, Maximum, Mean, Median, Range, Variance

The functions are referred to by the category name followed by the type of calculation, for example:

LocalAdd
FocalMaximum
ZonalMean

Each MMA function is implemented as a JAVA class. Each function class contains a method called 'execute' that is called to execute the function, and which is overloaded to handle different combinations of layer types as inputs to the method and for user-defined function parameterization.

4.1 Local Functions
A local function ingests at least one layer and outputs one layer. There are several types of input parameter combinations:

One layer and a scalar value
For example, a LocalAdd that ingests a spacecube and the value '5' would output a spacecube in which each element is equal to the sum of 5 + the value of the analogous element in the input spacecube.

Two layers of the same type
For example, a LocalAdd that ingests two spacecubes would output a spacecube in which each element is equal to the sum of the elements that share the same position in the two input layers.

A list of layers of the same type
For example, a LocalAdd that ingests a list of spacecubes would output a spacecube in which each element is equal to the sum of the elements that share the same position in all the input spacecubes.

One layer and another layer of fewer dimensions
For example, a LocalAdd that ingests a spacecube and a grid would output a grid in which each element is equal to the sum of the elements that share the same position in the two input layers. Note that for each element in the grid there will be many elements in the spacecube that share its position, as for every planimetric coordinate the spacecube will contain many altitudinal positions. This function is restricted to cases where the second input layer (e.g. a grid in the example given here) has a subset of the dimensions of the first input layer (e.g. a spacecube).

Within the local function category there is also another category called *rollup* functions. These functions transform an input layer into another layer type with fewer dimensions using a mathematical summarization. For example, a LocalRollUpZMean that ingests a spacecube would output a grid by taking the mean of all elements in the Z dimension for each [X,Y] element position.

The basic syntax for local functions is:

outlayer = localFunction.execute(inlayer1, inlayer2)

where 'outlayer' is the name of the layer generated by the function, 'localFunction' is the name of the local function, 'execute' is the name of the method called to invoke the function, 'inlayer1' is the name of the first input layer, and 'inlayer2' is the name of the second input layer. Other method signatures are made to operate on lists of two or more layers, to lag one layer by another in the X, Y, Z, and/or T dimensions, and to combine layers of different types.

4.2 Focal Functions

A focal function ingests at least one layer and one neighborhood and outputs one layer of the same type as the input layer. For example, a FocalAdd that ingests a spacecube and a cubic neighborhood would output a spacecube in which each element is assigned the sum of the element values located within the cubic neighborhood of each element. Focal functions can also ingest a lag to offset the neighborhood during the focal iteration. It is also possible to pass a list of neighborhoods and/or a list of lags into a focal function. If a list of neighborhoods (lags) is used, the focal function will utilize a separate neighborhood (lag) for each element's focal calculation.

The basic syntax for focal functions is:

```
outlayer = focalFunction.execute(inlayer, neighborhood, lag)
```

where 'outlayer' is the name of the layer generated by the function, 'focalFunction' is the name of the focal function, 'execute' is the name of the method called to invoke the function, 'inlayer' is the name of the input layer, 'neighborhood' is the user-specified focal neighborhood, and 'lag' is the user-specified lag.

4.3 Zonal Functions

A zonal function ingests a zone layer and a value layer and outputs a table. For example, a ZonalAdd that ingests a zone spacecube and a value spacecube will output a table in which each record represents one zone and for each zone is calculated the sum of all the value spacecube's elements contained within that zone. The zone layer must be either the same type as the value layer, or a type with a subset of the dimensions of the value layer. For example, a ZonalAdd that ingests a zone grid and a value spacecube will output a table in which each record represents a grid zone and for each zone is calculated the sum of all the value spacecube's elements contained within that zone. Note that in this case, there will be multiple spacecube elements associated with each grid element, as there will be many altitudinal positions in the spacecube associated with each planimetric position in the grid.

The basic syntax for zonal functions is:

```
outtable = zonalFunction.execute(inzonelayer, invaluelayer)
```

where 'outtable' is the name of the table generated by the function, 'zonalFunction' is the name of the zonal function, 'execute' is the name of the method called to invoke the function, 'inzonelayer' is the name of the zone layer, and 'invaluelayer' is the name of the value layer. Like MMA bcal functions, zonal functions can also combine layers of different types, provided that the dimensions of the zone layer type are a subset of the dimensions of the value layer type (e.g. a timeseries zone layer and a timecube value layer).

5. Discussion and Conclusion

We have presented the design and implementation for extending conventional map algebra to multiple dimensions for spatio-temporal data handling. Like conventional map algebra, individual MMA functions can be combined in a series to form more complex models, where the output of one function is ingested into another function, whose output is ingested into another function, and so on. MMA can thus be considered a powerful modeling language that can applied to variety of application domains where the analysis of temporal, spatial, and spatio-temporal data is important.

The design of MMA, where each function is defined as a separate JAVA class and each class has a method 'execute' that is overloaded to handle many different combinations of inputs, facilitates continued development and code sharing. We initially considered a design in which the functions were methods contained within the spatio-temporal data types classes. For example, the SpaceCube class would have had a LocalAdd function enc oded as a method within the class. While this approach is perhaps more in keeping with the principles of object-oriented design, it has some serious drawbacks. First, it is in contrast to the syntax of the conventional map algebra, a syntax with which many GIS users are familiar. Second, and more practically, in such a design every additional MMA function that is implemented would demand that the entire MMA library for that data type be recompiled. Third, such a design would demand that users intereste d in only a handful of MMA functions acquire the entire MMA library (at least for a given spatio-temporal data type).

For these reasons, we decided to implement each function as a separate class. In addition, such an implementation strategy provides a component-based design where users can acquire only the functions that are germane to their analysis and combine them however they like. New MMA functions may be developed and added to the library independent of the methods contained in any spatio-temporal data type.

We also note that MMA can be considered a spatio-temporal geoprocessing specification. There has been much recent activity on developing specifications for interoperability for spatial data by organizations such as the Open Geospatial Consortium (OGC), but specifications for data manipulation and/or analysis are for the most part still under development. By providing a formal language for spatio-temporal data processing of raster data, MMA can be considered a specification where a set of basic functions are defined and for each function both the inputs and outputs are specified. The function signatures (i.e. for the execute methods) can be considered the interface to the functions as they are, in fact, independent of the actual MMA data structures and algorithms. Thus, if two software packages were to share in a spatiotemporal analysis, one package would be able to 'borrow' geoprocessing tasks from the other if they could agree on the conceptual basis of the function and the data objects passed into and out of the individual geoprocessing functions, regardless of each package's actual function and data structure implementation.

The primary challenge facing the continued development of MMA concerns performance issues. Even two-dimensional raster data can be very large; three and four dimensional data can quickly overwhelm many desktop computing platforms even when using data with only a few time steps or altitudinal positions. Large data volumes are problematic in two ways. First, the volume of data can simply exceed the memory capacity of the hardware, whether in simply holding a single data set in memory or during function processing, where there is often the need to store multiple multidimensional arrays of the same size as the data set under analysis simultaneously. Second, even when there is enough memory to hold the data set and complete the calculation, the efficiency of many functions is very poor. Consider that many functions operate by iterating over every element in a data set to perform some kind of data retrieval and calculation. With three- and four-dimensional data, the number of elements can be quite large and the sheer number of data retrieval operations can be time-consuming.

These performance issues will be addressed in future research. Advances in two-dimensional spatial data storage and retrieval may be extended to multiple dimensions to improve MMA storage and algorithmic efficiency. In addition, we will continue to develop new MMA functions, particularly those that extend the statistical capabilities for local, focal, and zonal functions beyond the basic arithmetic and relational operators now offered. We also intend to extend certain terrain operators, such as least cost path, flow direction, and slope, to multiple dimensions.

6. Acknowledgements

Conversations with Roland Viger regarding the design and implementation of MMA contributed significantly to this research. Thanks also to C. Dana Tomlin for early suggestions on extending map algebra to multiple dimensions. This research was supported by NASA grant #NAG5-12598.

7. References

DeMers, M.N., 2002. *GIS Modeling in Raster* (Chichester: John Wiley and Sons).

Mennis, J., 2005. Spatial and temporal vegetation variability in Africa: an application of temporal map algebra. In *Proceedings of the ASPRS Annual Conference*, Baltimore, MD, March 7-11 (CD).

Mennis, J. and D.J. Peuquet, 2003. The role of knowledge representation in geographic knowledge discovery: a case study. *Transactions in GIS*, **7**, 371-391.

Mennis, J. and R. Viger, 2004. Analyzing time series of satellite imagery using temporal map algebra. In *Proceedings of the ASPRS Annual Conference,* Denver, CO, May 23-27 (CD).

Mennis, J., R. Viger, and C.D. Tomlin, 2005. Cubic map algebra functions for spatio-temporal analysis. *Cartography and Geographic Information Science*, **32**(1), 17-32..

Neteler, M. (Ed.), 2004. *GRASS 5.0 Programmer's Manual* (Trento, Italy: ITC).

Nikolaidis, N. and I. Pitas, 2000. *3-D Image Processing Algorithms* (Hoboken, New Jersey: Wiley).

Peuquet, D.J., 2001. Making space for time: issues in space-time data representation. *Geoinformatica*, **5**, 11-32.

Pullar, D., 2001. MapScript: a map algebra programming language incorporating neighborhood analysis. *GeoInformatica*, **5**, 145-63.

Scott, M.S., 1998. *The Exploration of an Air Pollution Hazard Scenario Using Dispersion Modeling and a Volumetric Geographic Information System*. Unpublished Dissertation. University of South Carolina, Columbia, South Carolina.

Tomlin, C.D., 1990. *Geographic Information Systems and Cartographic Modeling* (Englewood Cliffs, New Jersey: Prentice Hall).

Wesseling, C.G., D. Karssenberg, W.P.A. Van Deursen, and P.A. Burrough, 1996. Integrating dynamic environmental models in GIS: the development of a Dynamic Modelling language. *Transactions in GIS*, **1**, 40-48.

Yuan, M., 2001. Representing complex phenomena with both object- and field-like properties. *Cartography and Geographic Information Science*, **28**, 83-96.