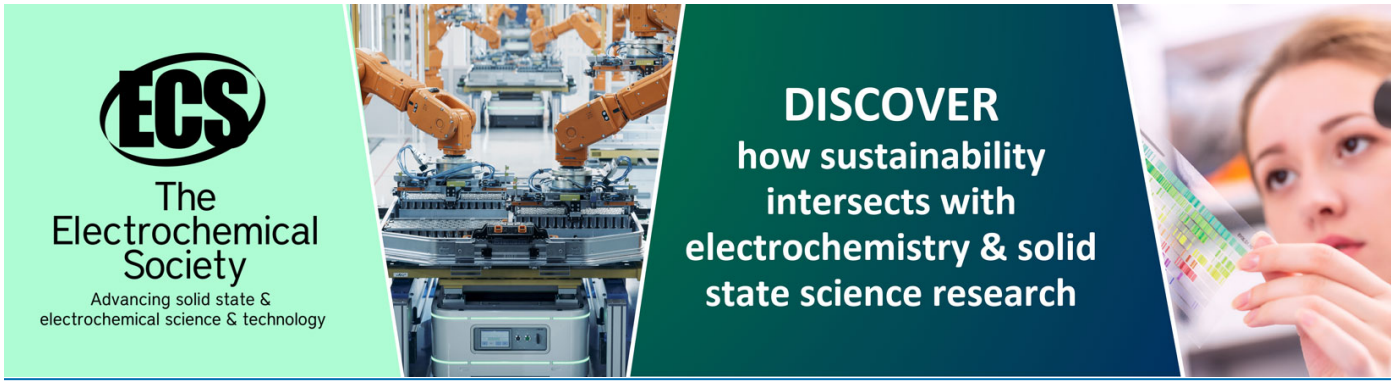
On-the-fly analysis of multi-dimensional rasters in a GIS



On-the-fly analysis of multi-dimensional rasters in a GIS

F Abdul-Kadar[1], H Xu[1] and P Gao[1]
[1]Environmental Systems Research Institute (Esri), Redlands, CA, USA

Email: akferoz@esri.com

Abstract. Geographic Information Systems and other mapping applications that specialize in image analysis routinely process high-dimensional gridded rasters as *multivariate data cubes*. Frameworks responsible for processing image data within these applications suffer from a combination of key shortcomings: inefficiencies stemming from intermediate results being stored on disk, the lack of versatility from disparate tools that don't work in unison, or the poor scalability with increasing volume and dimensionality of the data. We present raster functions as a powerful mechanism for processing and analyzing multi-dimensional rasters designed to overcome these crippling issues. A raster function accepts multivariate hypercubes and processing parameters as input and produces one output raster. Function chains and their parameterized form, function templates, represent a complex image processing operation constructed by composing simpler raster functions. We discuss extensibility of the framework via Python, portability of templates via XML, and dynamic filtering of data cubes using SQL. This paper highlights how ArcGIS employs raster functions in its mission to build actionable information from science and geographic data—by shrinking the lag between the acquisition of raw multi-dimensional raster data and the ultimate dissemination of derived image products. ArcGIS has a mature raster I/O pipeline based on GDAL, and it manages gridded multivariate multi-dimensional cubes in mosaic datasets stored within a geodatabase atop an RDBMS. Bundled with raster functions, we show those capabilities make possible up-to-date maps that are driven by distributed geoanalytics and powerful visualizations against large volumes of near real-time gridded data.

1. Introduction

Over the past three decades, the geographic information science community has been working hard towards solving the problem of efficiently managing and disseminating very large volumes of remotely sensed data, and making it available to an end user in a responsive environment, see [1, 2]. The past decade has experienced an explosion in the amount of remotely sensed earth observation data produced by organizations and government agencies around the world. The corresponding availability of mature GIS applications that are capable of handling the growing volume and variety of gridded data indicates that those earlier efforts have paid off well, see for instance, [3]. Contemporary systems are multi-tiered and distributed with well-established pipelines for efficiently streaming large volumes of data from the storage systems, NAS, SAN or object store [3], up through the software stack, over the web, and to the consumer's device.

The challenge, in this era, is to continually deliver value despite the growth along the dimensions of volume and variety. Organizations have long expended time and energy on transforming data to produce information which provide insights, enhance knowledge, and aid in decision-making. A GIS facilitates value addition through the efficient construction and timely dissemination of well-designed information products [4]. A data-derived product enables people to intuitively visualize complex phenomena or serves as an input data source in another transformation. However, traditional paradigms of workstation-based batch processing break down in the face of "big data." Systems which rely exclusively on closed frameworks that do not scale out well are unable to deliver information-based value effectively. The earth science community is in need of a unifying model that underpins a

scalable and extensible framework to drive distributed processing, desktop and cloud-based analysis, and responsive visualization of very large volumes of multivariate, multi-dimensional gridded data within a modern GIS.

We define one such model, called *raster functions*, for the dynamic transformation of rasters. The constructs described in this article, enable scientists and image analysts to codify, persist, and disseminate complex parameterized chains of transformations that are constructed using simple building blocks. These *function chains* represent an image processing or analytical model which can be applied seamlessly across the various modalities of interacting with rasters, from desktop workflows with a single dataset to cloud-based analytics against a catalog of tens of millions of rasters. The raster function framework drives the visualization and analytical engine within a GIS and enhances the system with the ability to dynamically construct a wide array of information products from raw data. The flexibility and portability of raster functions ensure that the lightweight transformations that define the final product, not the bulky input data sets, are transported around the system. Raster functions aid in modular development of functionality within a GIS, as described in [5]. Unlike systems that rely on a fixed set of operations, the corpus of raster functions is easily expanded through the use of a Python API.

In this article, we first present the basic mathematical constructs and the conceptual model for transforming rasters in the context of a GIS. We then describe the expectations and interactions between a typical application and a framework derived from that model. In this paper, the programming interface is described solely in abstract terms within the context of key high-level interactions with the framework. A more concrete programming interface of the raster function framework within ArcGIS is described in [6]. We also use instances of raster functions implemented within ArcGIS to elucidate an alternative measure of complexity of raster-based operations. We discuss the applicability of raster functions under non-traditional paradigms of data processing and analytics within a GIS.

2. Definitions and model

In this section, we revisit and extend the mathematical definition of fundamental image processing concepts. We then derive the conceptual models that embody those definitions. In an intuitive sense, we can describe a raster as a mapping of each point in a two-dimensional space to a color value. Building on the notations defined in [7] and [8], given a discrete subspace Z and a color-space C, a raster can be defined as:

$$R : Z \to C, \text{ where } Z \subseteq \mathbb{R}^d \text{ and } \mathrm{C} \subseteq \mathbb{R} \tag{1}$$

Here, Z is the domain over which the raster R is defined. Additionally, Z encapsulates all spatial and spectral aspects of a raster along each dimension such as the coordinate system, the resolution of the underlying graticule or grid, the extent, and the number of bands. C is the range of the raster R, and it encapsulates all the familiar radiometric properties of a raster such color-space, the bit depth or data type used for representing the output pixel value, and the set of valid and NoData pixel values. As an object, R incorporates key properties typically presented as metadata associated with the raster along with, basically, any parameter that serves to further constrain or describe the domain or range associated with the raster.

Next, we define a *raster dataset* as a non-transient object comprising a default instance of a raster and the associated extents. Here, we define it along the lines of a static object from [7] as a tuple:

$$\delta \triangleq (R_\delta, Z_\delta) \tag{2}$$

In practice, a raster dataset is typically a format-agnostic representation of an image file stored on disk or in a database [9]. Such an object is capable of handling transformations between the spatial domain of the dataset, Z_δ, and the spatial domain native to the raster R_δ. As eluded by [10], the raster dataset could, for instance, encapsulate the camera model of a remotely sensed image to transform coordinates from map space, through camera space, to image space. An application also relies on the dataset object for all higher-level interactions such as accessing or modifying parameters that control

metadata, projections and transformations, persistence, etc. Use of the underlying raster is reserved primarily for reading and writing pixel values.

Now, a *raster function* is a mapping of a tuple of one or more rasters to another raster:

$$F_\pi: R^n \rightarrow R \tag{3}$$

Here, $n \in \mathbb{N}_{\geq 0}$, is the rank of a raster function, and π is the set of scalar parameters that further qualifies an instance of F. Simply put, a raster function is a parameterized operation capable of producing one output raster given zero or more input rasters. Conceptually, a *function dataset* extends a raster dataset using an instance of a function and an instance of the associated argument set.

It represents a dynamically processed raster, and can be described as a tuple:

$$\phi \triangleq (F, \pi, R^n, Z) \tag{4}$$

Here, n is the rank of F, and R^n represents the n-tuple of rasters derived from various inputs to the function dataset. Given that a function dataset extends the definition of a raster dataset, and these datasets can conceptually be treated as rasters, we can rewrite (4) like this:

$$\phi_i = (F_i, \pi_i, \rho_i, Z_i), \tag{5}$$

$$\text{where } \rho_i \subseteq \{\phi_{<i}\} \cup \{\delta_k : k \in N\}, \text{and } |\rho_i| = n$$

Here, ρ_i represents the subset (of cardinality, n) containing all input rasters for the i^{th} function dataset constructed using all (N) raster datasets and using previously encountered function datasets $(\phi_{<i})$. A composition based on the above recurrence relation is called a *raster function chain* and typically serves to encode a multivariate model for raster-based analysis or processing.

A *raster function template* is an instance of a raster function chain where one or more elements of π_i or ρ_i, for some value of i, represent a place-holder *variable* and not an actual instance of the scalar or raster object. Well-crafted raster function templates make analytical models reusable across a GIS. We discuss interactions using more concrete examples in the next section.

3. Anatomy and interactions

An implementation of a raster function needs to conform to a predefined application programming interface (API) to enable effective interactions with other components in the GIS, primarily, the parent function dataset, on aspects associated with pixel data, mask, metadata, and core properties of the input and output rasters. In this section, we discuss a few key workflows that will allow us to elaborate on the ensuing interactions between the application and the raster function framework.

3.1. Adding a raster function

Consider the use case of constructing a function dataset, $\phi_i$as described in (5), for the simplified scenario where we have one input raster (δ_k) or another function dataset (ϕ_{i-1}). Basically, we extend a function chain by one new function: F_i. This interactive workflow primarily entails a user choosing from the collection of available raster functions, populating scalar values, and attaching datasets associated with the function arguments, π_i and ρ_i. Assuming the application maintains a set of references to functions that are available to the user, we would need every raster function to be able to describe each expected input parameter.

Once the advertised scalar and raster arguments are provided by the user, the application vets and then loads objects associated with the inputs, and delivers them to the raster function. The function (F_i) then provides the application with the *raster info* object (I_{F_i}) associated with the output raster. *Raster info* is a framework-level construct that encapsulates core properties associated with a concrete representation of the *raster* concept described under (1) in the preceding section. As an object, it describes the resolution and boundary of the native geospatial (or spatio-temporal, or multi-dimensional) grid over which the corresponding raster is defined. It also describes the radiometric and spectral properties such as the data type of the pixel, number of bands, the wavelengths associated with each band, and the distribution of pixel values in the raster.

Given the raster info (I_{F_i}) associated with the output raster, the function raster dataset (ϕ_i) is able to construct the dataset's domain (Z_i). Additionally, it is also able to construct the complete set of transformations necessary to map any geospatial interaction with Z_i to a corresponding interaction with the native domains associated with each input raster [10]. We then have a fully instantiated function dataset against with other basic or high-level operations may be performed.

3.2. Applying a raster function template

An alternate approach to building a function dataset (ϕ_i) is to bind the set of all expected inputs to a templatized chain of preexisting functions as described in (5). Here, the application is provided with a raster function template (τ) from which the set of input variables (v_τ) is constructed. A trivial way to construct v_τ would be to iterate over all scalar and raster inputs associated with each raster function in τ, and add to v_τ those inputs which aren't backed by actual scalar or raster objects. In practice, however, those place-holder inputs have concrete objects that further define properties like default value, aliases, and display name associated with the variable. These properties enable applications to meaningfully present users with a request for input.

The particular manner in which template variables bind with the corresponding inputs depends very much on the context or environment. Desktop applications built using sophisticated graphical frameworks might allow users to easily select one or more layers from a map, and then present them with perhaps a carousel of powerful ready-to-use templates that can be easily applied on the input layers. The subsequent result could then be previewed as another dynamic layer driven by the function dataset. On the other hand, server-based analytical operation invoked through RESTful calls from a client-side script might involve transmitting the template as well as values or URLs of all input parameters in JSON. As objects that can be defined and persisted without referencing actual data sources, function templates extend the reach and usability of the foundational concept of raster functions to various raster-based workflows in a GIS.

3.3. Reading pixels of a function dataset

Now consider a function dataset (ϕ_i) that came into existence either through the interactive workflow of adding a function to ϕ_{i-1} or through the association of a function template to its argument set ($\pi_i \cup \rho_i$). Ultimately, the function dataset (ϕ_i) represents a chain of raster functions based on (5) with the primary purpose of delivering transformed pixels and metadata. Typical application-level workflows that eventually "pull" data through a function chain might involve a user interactively roaming a map containing one or more layers driven by a data source containing a function dataset. Or, the data access may be the consequence of executing an automated script that performs a long-running analytical operation against a large catalog of imagery in a multi-core or distributed environment. The raster function framework provides a unified mechanism for requesting pixels from a dataset.

In the context of data extraction, the spatial domain (Z_i) and the function dataset (ϕ_i), together, define a virtual output raster (R_{ϕ_i}) from which pixels are read. And because the model in (1) doesn't mandate how an application specifically interacts with a concrete instance of a raster, the mid-tier framework provides applications with mechanisms that enable both sequential and ad hoc access to pixels in a raster. A request for pixels in any raster (R) is qualified by a set of properties (p) that defines the location and size of a block of pixels $\left(b_R^p\right)$ such that $Z_b \subseteq Z_R$. For the scenario of a user roaming a map, Z_b typically represents the current area of interest. Operations that stream a raster in its entirety can iterate over Z_R using Z_b, processing one pixel-block at a time. The software implementation of this approach has been described in [11].

The raster function model clearly doesn't preclude itself from the use of parallel threads of execution to consume data from a single virtual raster. This is possible when the parent dataset representing the output raster (R_{ϕ_i}) is accessible across threads, or processes, or nodes of a cluster of machines. Application-level operations can reconstruct the complete output raster that span Z_R by

parallelizing the construction of the union, $\bigcup_i Z_{b_i}$. This in turn involves building corresponding pixel blocks $b_R^{\{p_i\}}$ in parallel. We revisit this line of thought in section 5.2.

3.4. Processing multi-dimensional data

The conceptual model described in the preceding section for processing imagery lends itself to handling multi-dimensional raster data. For the sake of relevance to a GIS, we'll consider a data source or any derivative dataset to be multi-dimensional only if it is defined beyond the three basic dimensions prevalent in GIS, namely, the two planar spatial dimensions (X and Y), and the one spectral dimension (band or channel). The presence of a multi-dimensional data source driving a raster dataset does indeed mandate that the native spatial domain, Z in (1), describe a gridded hypercube with $d > 3$.

In our framework, however, the pixel block $\left(b_R^p\right)$ from section 3.3 continues to remain planar, representing a two-dimensional block of multiband pixels. This representation has remained foundational in GIS where multicriteria decision analysis has been performed through complex modelling over several planar thematic layers. Access to pixels along the non-trivial dimension is made possible through the use of properties that identify the location of a planar pixel block in some high dimensional space. These properties encode a *multi-dimensional filter* and enable an application to subset a hypercube or iterate over the discrete hyperplanes parallel to the spatial axes. This slice-based approach to interacting with a cube integrates with existing GIS implementations that support planar rasters. Implementations based on this paradigm are going to be scalable and memory efficient compared to true multi-dimensional rasters where pixel blocks also span non-spatial dimensions or where $C \subseteq \mathbb{R}^d$.

4. Methods of analysis

Image processing operations have traditionally been organized by the size of the spatial neighborhood used for deriving an output pixel value. For the purpose of this article and in the context of a mature GIS, we find it effective to also categorize raster functions by the *cardinality* of the set of input rasters and the *dimensionality* of those rasters. This serves as an alternative measure of the per-pixel complexity of an image processing operation.

4.1. Processing a single raster

The simplest class of raster functions under this categorization are those of unit rank—accepting a single input raster and set of scalar arguments. Trivial examples include functions that perform point operations on the input raster. The raster functions *Contrast and Brightness*, *Unit Conversion*, *Colormap to RGB*, *Extract Bands*, and *Apparent Reflectance*, used in ArcGIS, and the implementation of *Linear Spectral Unmixing* in [12], illustrate the breadth of complexity in spectral and radiometric transformations performed as point operations. One of the most well-known and widely used spectral transformations is the normalized difference vegetation index (NDVI) is implemented as a raster function in [12]. The implementation illustrates the simplicity of a raster function that works on a single multi-band input raster.

The *Hillshade* function [12] and its multidirectional variants are examples of neighborhood operations commonly-used for visualizing rasters that represent elevation. The *Segment Mean Shift* is a unary function that performs a spectral transformation as a zonal operation (where the per-pixel complexity in the spatial domain is not known a priori).

Also, there are raster functions in this category which perform no real processing on the underlying dataset, instead, choosing to transform non-pixel aspects of the input raster. These functions typically forward calls from the applications for pixel values to the underlying raster dataset. The *Key Metadata* [12], the *Statistics and Histogram*, and the *Attribute Table* raster functions modify corresponding properties associated with the input raster. Another set of raster functions, that are degenerate cases under this category, require no raster input. In this case, the resulting function dataset emulates a raster

dataset by mapping discrete spatial coordinates to color values with possibly some dependency on scalar input arguments. The *Random* function in [12] and *Constant* function in ArcGIS are specimens of pixel generators. Additionally, a raster function template that has no unresolved variable is an atypical instance of a degenerate zero-input raster function.

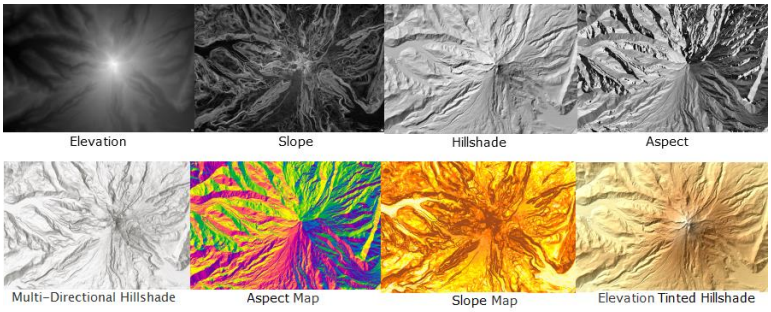


Figure 1 A subset of information products dynamically constructed from *elevation* source data (top-left) using *raster functions*.

4.2. Combining multiple rasters

The *n-ary* functions in this category construct the output raster by combining two or more single- or multi-band input rasters. Well-known examples of binary functions in ArcGIS include those that transform spectral and spatial properties of the input rasters (*Pan-sharpening*), and those that transform the color space (*Vector Field*). The per-pixel complexity of these functions is bounded by the rank of the function. Additionally, functions in this category which do not aggregate over the spatial domain expect all input rasters to geospatially overlap. In practice, this might be achieved through the application of spatially-transforming function chains on each input raster dataset. The implementations of *Heat Index*, *Wind Chill*, and *Mask* in [12] serve to demonstrate the simplicity of raster functions that rely on this assumption. The *Merge Rasters* and *Geometric* functions specialize in reconstructing the function dataset's spatial domain given two or more rasters.

4.3. Aggregating a univariate stack of rasters

In the context of this paper, we refer to a raster that spans one additional non-basic dimension (section 3.4) as a cube, such that $d = 4$ as described in (1). The unary raster functions in this category specialize in constructing the output raster, with $d = 3$, by aggregating the slices of the cube along that one dimension. Once again, such functions generally make the same assumptions on the geospatial locations of those slices, as described in section 4.2. The *Aggregate* function in [12] is a basic instance of a raster function in this category which emits summary statistics as the output raster. One could compute, mean sea-surface temperature or maximum wind-speed, given temporal slices over those variables. The *Composite Band* function can be used to transpose a stack of rasters to one virtual multi-band raster. The *ArgStatistics* raster function, a generalization of the argmax function, outputs the index of the slice that corresponds to the specified statistic.

4.4. Processing a multivariate gridded hypercube

A straightforward extension to the preceding categories is the case of an *n-ary* function with multi-dimensional input rasters. Here, $n > 1$ as described (3) and two or more input rasters have $d > 3$ as described in (1). Because the complexity of these functions stem from the cardinality and dimensionality of the input, these functions continue to build on the assumption of geospatial overlap we discussed in section 4.2. A basic example would combine the *Aggregate* and *Heat Index* functions described above to output a planar raster that represents average heat index. These functions are employed in multicriteria decision analysis using multi-dimensional raster data or in multivariate spatio-temporal clustering.

5. Discussion

5.1. Extending Analytical Capabilities

Although the raster function is designed to be a parameterized object that enables customization of core behavior, a salient feature of the framework implemented in ArcGIS is its extensibility. The application programming interface (API) defined in [6] allows scientists and image analysts to implement novel algorithms using the Python programming language and high-performance libraries like NumPy and SciPy for data manipulation and numerical computation.

5.2. Managing Performance and Scalability

In the context of a concrete instance of a raster function, the speed with which the object completes any request indirectly affects the responsiveness of the user experience or the overall throughput of a batch analytic process. Not all methods on the raster function object need equal attention to optimization, though. The *getParameterInfo* and *updateRasterInfo* methods described in [6], for instance, are only invoked by ArcGIS during the initial construction of the function dataset. The *updatePixels* method, however, is the workhorse of the raster function object and gets invoked once for every pixel block of the output raster over regions requested by the application. Idiomatic Pythonic techniques and established best practices associated with high-performance scientific computing, like minimizing data movement, using Cython, and vectorizing calculations, have proven to be effective at improving performance.

The scalability of any implementation of the conceptual model described in section 2 highly depends on the system architecture. Most modern geographic information systems, like ArcGIS, are built with the ability to leverage computational scalability through multiple cores and through distributed nodes in a cluster of machines. Such systems are able to exploit the parallelized construction of output function datasets described in section 3.3 especially in the context of batch analytics of geospatial big data. Recall that this is possible because of the model's output-centric nature wherein the application "pulls" data from the top-most function dataset causing pixels to be pulled up the chain of raster functions, and eventually from all input datasets. Such applications scale out simply by inducing parallel tiled requests over the spatial domain of the virtual output raster.

Analysts seeking to leverage the map-reduce paradigm to parallelize aggregation of pixels over any dimension, spatial or non-spatial, need to be aware of relevant properties of the measure being computed by the raster function in the context of partitioning data along the dimension of aggregation. As described in [22], distributive measures (like sum, count, or max) that are based on commutative and associative operations would present minimal challenges during the reduce phase of the operation where intermediate results from the partitions are being combined. However, solutions that rely on raster functions that compute algebraic measures (such as average or standard deviation) need to output the corresponding sufficient statistic (such as sum, count, sum of squares) as intermediate virtual rasters to aid in the computation of the final measure.

Although parallelization over the output function dataset can be achieved either through replication or through network-based sharing of all input data, scalability is dramatically improved when all input rasters come from web-based image services backed by a computer cluster.

5.3. Limitations

Based on the limited scope of this study, the authors direct the readers to [7, 8] for a more mathematically rigorous treatment of digital image processing constructs. For a more concrete discussion on analysis of raster data in ArcGIS and for detailed API we refer the reader to [6, 11]. Also, there is more work to be done to extend the utility of raster functions by supporting truly multi-dimensional pixel blocks such that pixel values span non-spatial dimensions (see section 3.4).

6. Conclusions

We described the mathematical constructs that form the basis of a transformation pipeline for multi-dimensional rasters in a GIS. An implementation underpinned by the conceptual model of a raster

function provides applications with a unified framework for dynamically processing, analyzing, and visualizing gridded geospatial big data. Among other features, this cross-paradigm unification of how rasters are transformed is pivotal in extending the state-of-the-art for how scientific investigation is conducted, how analytical models are created, and how results are disseminated atop modern geographic information systems.

Well-designed frameworks are able to handle multivariate multi-dimensional rasters even in the context of prevailing layer-based interactions in a GIS. Systems, like ArcGIS, that have solved the problem of managing large volumes of rasters and those that are primed for (or already capable of) distributed batch analytics are set to benefit most from the use of a "function-based" approach for processing gridded data.

The dynamic nature of raster functions helps elevate the utility of raw data by simplifying and speeding up dissemination of analytical information products—a sought-after quality in time-critical areas like disaster response and risk assessment. The gamut of purpose-built raster functions aid in the application of multicriteria decision analysis like in suitability modelling or urban planning. The powerful ability to interact with a virtual raster through descriptors that provide ad hoc access to pixels processed by a chain of functions enables the framework to scale out and leverage high-performance computing clusters. This scalability helps organizations lower cost and become data-volume-agnostic.

The model is flexible primarily because a composite chain of parameterized functions, persisted to well-known formats like XML and JSON, then dynamically applied by binding with raster inputs. The framework's flexibility and Python-based extensibility allow a scientist or analyst to encode their algorithm in a reusable package which aids in reproducibility of the resulting information product and overall transparency of the process.

7. Acknowledgements

We gratefully acknowledge P Becker for numerous insightful comments on this study. We thank K J Butler and P Mangtani for early comments on this study. We thank R Nalankal, J Zhang, J Drisdelle, A Jain, K Dhruv, and K Chandnani for their contributions in the development of the raster function framework in ArcGIS. ArcGIS® is the intellectual property of Esri and is used herein under license.

8. References

[1] H. M. Raafat, Q. Xiao and D. Gauthier, "An extended relational database for remotely sensed image data management within GIS," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 29, no. 4, pp. 651-655, 1991.

[2] A. Das Sarma, H. Lee, H. Gonzalez, J. Madhavan and A. Halevy, "Efficient spatial sampling of large geographical tables," *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 193-204, 2012.

[3] Z. Yu, S. Zheng and Y. Zhang, "New generation storage model for GIS spatial data based on object-based storage," *MIPPR 2005 Geospatial Information, Data Mining, and Applications*, pp. 60450M-60450M, 2005.

[4] R. F. Tomlinson, Thinking about GIS: geographic information system planning for managers, Redlands: ESRI, Inc., 2007.

[5] M. F. Goodchild, "Towards an enumeration and classification of GIS functions," *Proc. Int. GIS Symposium*, 1987.

[6] F. Abdul-Kadar and J. Drisdelle, "ArcGIS Raster Functions - Wiki," 2014. [Online]. Available: https://github.com/Esri/raster-functions/wiki#raster-functions.

[7] E. L. Fiume, The Mathematical Structure of Raster Graphics, San Diego, CA: Academic Press Professional, Inc., 1989.

[8] M. Sonka, V. Hlavac and R. Boyle, Image processing, analysis, and machine vision, Cengage Learning, 2014.

[9] H. Xu and P. Gao, "Custom Image Processing Capabilities in ArcGIS," in *The International Archives of the Photogrammetry. Remote Sensing and Spatial Information Sciences*, Beijing, 2008.

[10] H. Xu, P. Gao and J. Willison, "On-demand Projection of Large Raster Datasets in GIS," Citeseer, Redlands, 2004.

[11] H. Xu, P. Gao and R. Berger, "Developing with ArcGIS Raster APIs," 2009. [Online]. Available: http://downloads2.esri.com/campus/uploads/library/pdfs/97359.pdf.

[12] F. Abdul-Kadar, "ArcGIS Raster Functions," 2014. [Online]. Available: https://github.com/Esri/raster-functions.