A Scalable Image Processing Framework for Gigapixel Mars and Other Celestial Body Images

Mark W. Powell, Ryan A. Rossi[†] and Khawaja Shams
Jet Propulsion Laboratory, California Institute of Technology,
{Mark.W.Powell, Ryan.A.Rossi, Khawaja.S.Shams}@ jpl.nasa.gov

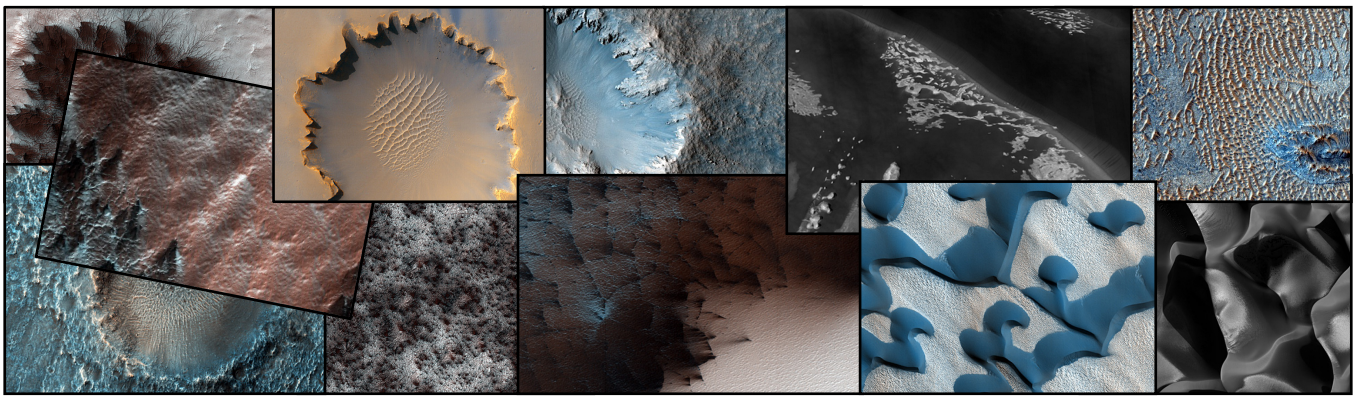
Abstract—The Mars Reconnaissance Orbiter's HiRISE (High Resolution Imaging Science Experiment) camera takes the largest images of the Martian surface. The image size is typically around 2.52 gigapixels. There is only a handful of software capable of doing a task as simple as reducing the size of the image by half and saving the result as a new image. The Scalable Image Processing Framework (SIPF) overcomes these issues by creating a generalized tile-based processing pipeline that loads only a small portion of the image into memory. This allows for the data in memory at any given time to become manageable. Image tiles are an intrinsic property that provides scalability and efficiency while processing images. Distributed computing technologies such as cloud computing can be applied naturally. A mathematical framework for scalable image operations is defined that provides insight into the scalable considerations needed with each class of operations. We also formalize the deferred execution design pattern and show how it is used as a basis for our implementation. The SIPF has the ability to perform a variety of Scalable Image Operations such as Cropping, Rotation, Scaling (Bilinear and Nearest Neighbor Interpolation), Edge Detection, Sharpening, Convolution (Filters), Brightness, Contrast, and Gaussian Blurring. The Scalable Image Processing Framework will be used to process incoming images from the Mars Exploration Rovers and eventually the Mars Science Laboratory. It will be integrated with the Maestro software (science visualization and planning tool). Maestro is used for the Mars Exploration Rover Mission and other celestial body exploratory missions. [1] [2]

TABLE OF CONTENTS

1. INTRODUCTION

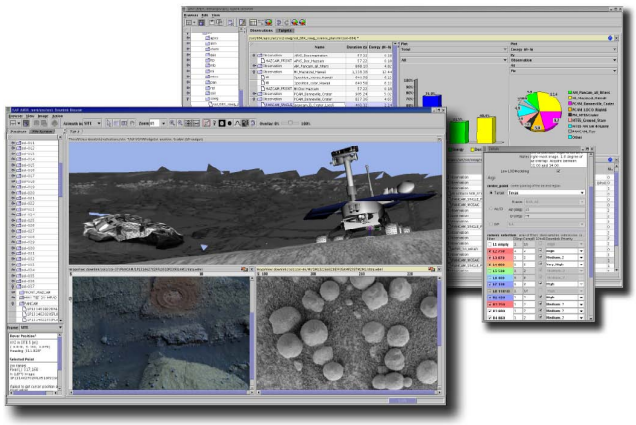
The Mars Reconnassance Orbiter's HiRISE camera takes very high resolution images up to 2.52 gigapixels (2520 megapixels). The HiRISE camera is a 0.5 m reflecting telescope and is the largest ever carried on a deep space mission. It has a resolution of about 1 microradian. The ground sample distance is 30cm per pixel from an altitude of 300km. The images can be either near infrared or red-green-blue [9, 10]. A few of the HiRISE images that we have worked on in the Operations Planning Software Research group at the Jet Propulsion Laboratory are shown below.

The Maestro team develops operations software used to conduct mission exploration of the Moon, Mars, and other celestial bodies [1-5]. Planning and operating these missions all benefit from imaging and mapping planetary surfaces in detail. As the size and volume of imagery from planetary



[1] 978-1-4244-3888-4/10/$25.00 ©2010 IEEE.
[2] IEEEAC paper #1533, Version 1, Updated August 5, 2009

missions increases, image processing software faces the challenge of keeping pace with the increase in scale while still providing the high performance that is expected of superior software.



Most image processing software assumes you can load the entire image into memory and then perform an image operation. This is typically not possible due to the size of planetary imagery. The images at HiRISE provide a good example as these are typically in the gigapixel range (one billion pixels 10^9). We overcome these issues by creating a generalized tile-based processing pipeline that loads only a small portion of tiles into memory at any given time. This creates a streaming pipeline of data for the image operations to take advantage of without having severe memory issues. The memory needed to process an arbitrary image is decoupled from the size of the image and is instead bounded by the size of the cache. This approach allows for images in the gigapixel range to be processed on consumer based computers as well as distributed across computers very naturally and efficiently.

The science activity planning for the Mars Rovers and eventually the Mars Science Laboratory requires the most recent images from the rovers to evaluate the executed science. The Scalable Image Processing Framework will be used to process operations images to support scientists in the planning and operating of the Mars Exploration Rover and the Mars Science Laboratory. The scalability of the framework is a necessity due to the massive size and amount of images we will be processing on a daily basis. We also take advantage of distributed computing technology as these technologies are intuitively applied with our framework. Our framework also supports tile-based delivery for web applications such as Google maps or our science delivery system that efficiently delivers images to planning teams located in different countries.

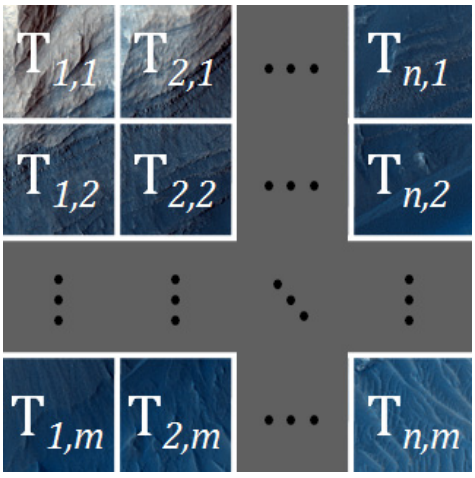
In the next section we define the mathematical framework for scalable image operations. In the third section we formalize and briefly describe the deferred execution design pattern and explain how it is used as a basis for our Scalable Image Processing Framework. In the fourth section we describe the design and implementation details. Finally in the last section we describe the distributed and cloud computing strategies used with our Scalable Image Processing Framework.

2. Mathematical Framework

In this section we describe the mathematical definitions to be used throughout the paper to define our scalable image processing framework. The definitions are used for representation and processing of tiles in a scalable manner. Informally we define a matrix of tiles where the tiles are matrices themselves. These matrices will have weaker mathematical properties. Therefore results that hold for normal matrices cannot be directly applied in the same way.

Let $T_{i,j} \in \Re^{w \times h}$ be a matrix corresponding to an image tile where (i, j) are the coordinates of the tile with respect to the image matrix denoted as I and $w \times h$ are the width and height of the tile, respectively. Furthermore, let $T_{i,j}[x, y]$ be a pixel value at location $[x, y]$ with respect to the width and height of the image tile $T_{i,j}$. Therefore a tiled image I is defined as

$$I = \begin{bmatrix} T_{1,1} & T_{2,1} & \cdots & T_{n,1} \\ T_{1,2} & T_{2,2} & \cdots & T_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ T_{1,m} & T_{2,m} & \cdots & T_{n,m} \end{bmatrix}$$



Where $n \times m$ is the width and height of the image with respect to the tiles in the image. Furthermore let $I_{i,j}$ be a tile at location i, j. Therefore $I_{i,j} = T_{i,j}$ and $I_{i,j}[x, y] = T_{i,j}[x, y]$. Now we list a few properties of this definition that are related to image processing:

1. The total amount of pixels in the image plane is defined as $(nm)(wh)$ where nm is the amount of tiles in the image and wh is the number of pixels in every tile.

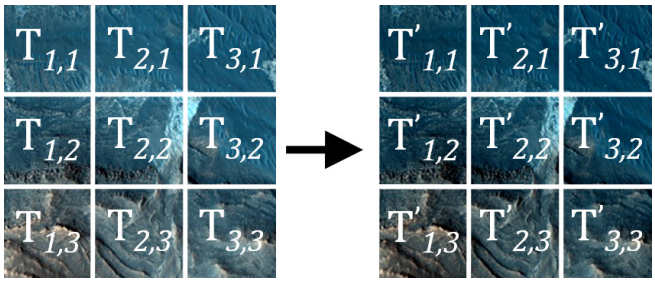
2. Given an arbitrary pixel location [x, y] then the appropriate tile can be referenced by $I_{\lceil x/w \rceil, \lceil y/h \rceil}$.
3. It is also easy to see that [*iw - w, jh – h*] is the minimum pixel location and [*iw, jh*] is the maximum pixel location of $T_{i,j}$ with respect to I.

In the next few subsections we define the main scalable image operations and discuss the scalable considerations.

2.1 Scalable Point Operations

A point operation performs a mapping of a pixel value but without changing the size, geometry, or local structure of the image. We show a simple example of how a scalable point operation is defined. Below we increase the image's intensity by a factor α where $I_{i,j}(x,y)$ is a pixel value from the tile $T_{i,j}$.

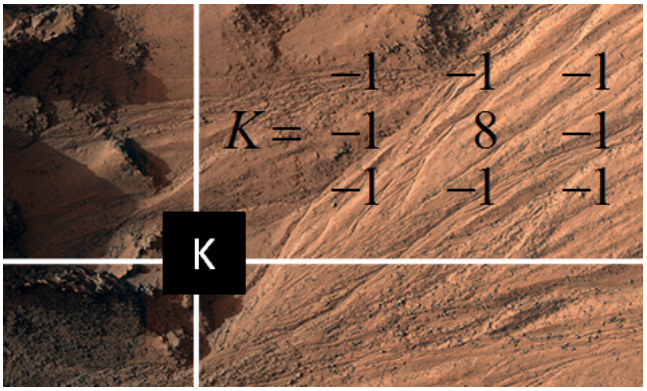
$$I'_{i,j}[x,\ y] = I_{i,j}[x,\ y] \times \alpha \qquad (1)$$



This operation is easily made scalable since each new pixel value depends exclusively on the previous value at the same position. Therefore the computation is independent from its neighboring pixel values and the geometry of the image is unchanged.

2.2 Scalable Convolution

Linear convolution is a mathematical operation that combines two functions f and g producing a third function. Convolution is the underlying concept of all filter operations in image processing and thus described as a black box operation. The results of convolution are defined by the convolution matrix or kernel. A kernel is generally defined as a nxn matrix denoted as K^{nxn}.



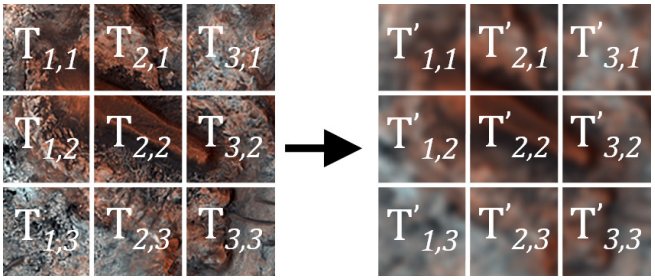
Convolution from an Image Processing perspective can be thought of as sliding a kernel K across an image I such that the middle coefficient of the kernel K is multiplied by every pixel in the image. Therefore it is straightforward to see that scalable considerations are needed to appropriately convolve tiles.

Now we define a simplified scalable smoothing operation. To compute a new pixel in the smoothed image we average the original pixel and the eight neighboring pixels. Given the pixel location $I_{i,j}[x + u, y + v]$ where we know [x,y] is in $T_{i,j}$ it is clear that if

$$\begin{aligned} iw - w > x + u > iw \\ jh - h > y + v > jh \end{aligned} \qquad (2)$$

then we are in a situation where the given pixel location is outside the tile $T_{i,j}$. Therefore let $i' = \lceil (x + u) / w \rceil$ and $j' = \lceil (y + v) / h \rceil$ be the tile corresponding with the pixel location [x+u, y+v]. Now we can define a simple scalable 3x3 smoothing operation as

$$I'_{i,j}[x,\ y] = \frac{1}{9} \sum_{v=-1}^{1} \sum_{u=-1}^{1} I_{i',j'}[x+u,\ y+v] \qquad (3)$$



Furthermore the more general notion of convolution follows naturally. Let $I_{i,j}[x, y]$ be a pixel from the image I in the tile $T_{i,j}$ and K^{nxn} be a kernel. Linear convolution of the image I with the kernel K is defined as

$$I'_{i,j}[x,\ y] = \sum_{(u,v)\ \in\ K} I_{i',j'}[x+u,\ y+v] \times K^{*}(u,\ v) \qquad (4)$$

Where K*(u, v) is equivalent to K(-u, -v) rotated by 180 degrees.

As an example the Laplace kernel is generally used for sharpening and enhancing edges. If a filter coefficient is negative it can be interpreted as the difference of two sums. The filter essentially computes the difference between the center pixel and the weighted sum of the four surrounding pixels. In our scalable framework we have used convolution for Difference Filters, Gaussian Blur, Minimum Filter (Dilation), Maximum Filter (Erosion), Sharpening, and Edge Detection. Any filter operation can be easily performed by passing the corresponding kernel to the

convolution function.

2.3 Scalable Geometric Operations

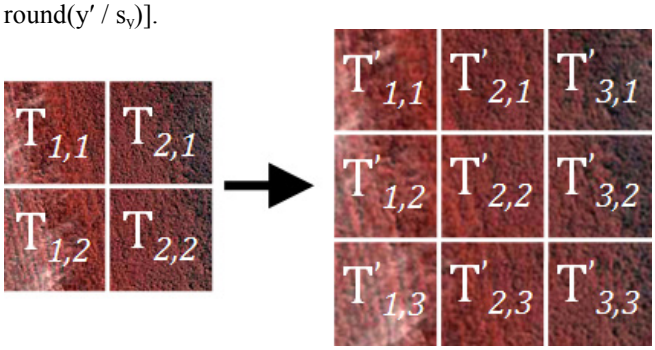
A geometric operation transforms an image I by modifying the coordinates of the pixels. Therefore given an image $I_{i,j}[x, y]$ the result of a geometric operation is denoted as $I'_{i',j'}[x', y']$ where I′ is of size n′ x m′ with respect to the transformation. We use geometric operations to scale the size of an image (Bilinear and Nearest Neighbor Interpolation) and also for rotation. We define an arbitrary continuous mapping function M() as

$$x' = M_x(x,\ y) \quad and \quad y' = M_y(x,\ y) \qquad (5)$$

Most transformed coordinates [x′, y′] will no longer fall onto the corresponding previous discrete point [x, y] in the image plane therefore interpolation is used to compute intermediate pixel values. This can be thought of as using the original discrete image I and transforming it using a continuous function into another discrete image I′ without significantly reducing the quality of the image. Interpolation can be seen as trying to reconstruct the transformed image (made by a continuous function) using the set of discrete pixel values from the original image. Therefore there is a need to estimate the intermediate pixel values of the transformed image using the original pixel values. An affine mapping function used to scale an image is denoted as

$$\begin{aligned} M_x &: x' = s_x \cdot x \\ M_y &: y' = s_y \cdot y \end{aligned} \qquad (6)$$

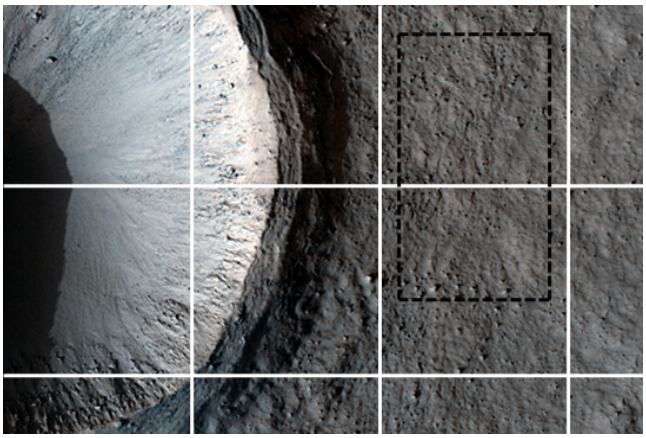
where s_x and s_y are scaling factors. We define a scalable scaling operation using nearest-neighbor interpolation. Nearest-neighbor interpolation simply rounds the continuous coordinate to the closest integer and uses this as an approximation. Let I′ be an n′ x m′ matrix of tiles where $n' = \lceil (nw)s_x / w \rceil$ and $m' = \lceil (mh)s_y / h \rceil$. Therefore given an arbitrary pixel in the transformed image $I'_{i',j'}[x', y']$ where i′ and j′ are the tile coordinates with respect to the transformed image I′ then the interpolated pixel is $I_{i,j}[\text{round}(x' / s_x), \text{round}(y' / s_y)]$.



It is straightforward to see how a form of convolution can be applied to geometric operations. The parameters used in a transformation can be found by solving a system of linear equations $x' = Ma$. An interesting future direction will be to apply Singular Value Decomposition to predict the pixel values of an unknown region in an image using the known pixel values in I as a basis.

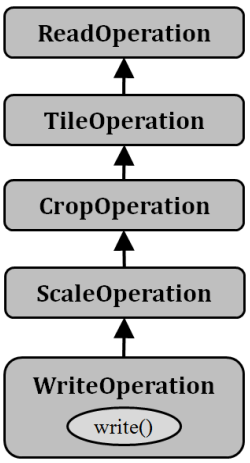
2.4 Scalable Cropping Operation

To crop an image we had to translate coordinate systems to request the appropriate tiles needed to make the crop tiles. Assuming tiles of size *256 x 256*, if we started cropping at image location $I_{1,1}[100,100]$ we would have to request tiles $\{T_{1,1}, T_{2,1}, T_{1,2}, T_{2,2}\}$ to make the first crop tile.



In the diagram above the tiles are designated by white lines where the dotted line represents the image region to be cropped. The crop operation will request the image tile $T_{3,1}$ thus calling the Image Reader where the tile will be randomly accessed and read. The crop operation will copy the appropriate data into the crop tile $C_{1,1}$ and make another request to read in the image tile $T_{3,2}$ and copy the remaining data into the tile $C_{1,1}$. If there are no more image operations to be performed on $C_{1,1}$ the writer will write the crop tile to disk. The scalable image processor will begin reading and processing the next tile $C_{1,2}$ in a similar manner until the operation is completed.

3. Deferred Execution Pattern



Design patterns originated with Christopher Alexander in 1977 as a way to describe fundamental building blocks of towns, buildings, and construction. Gamma et. al. extended the notion of design patterns to object oriented programming. A design pattern in object oriented programming can be described as a template or a reusable solution that can be applied to a similar common occurring problem in software engineering. Design patterns can speed up development by providing proven tested paradigms [11].

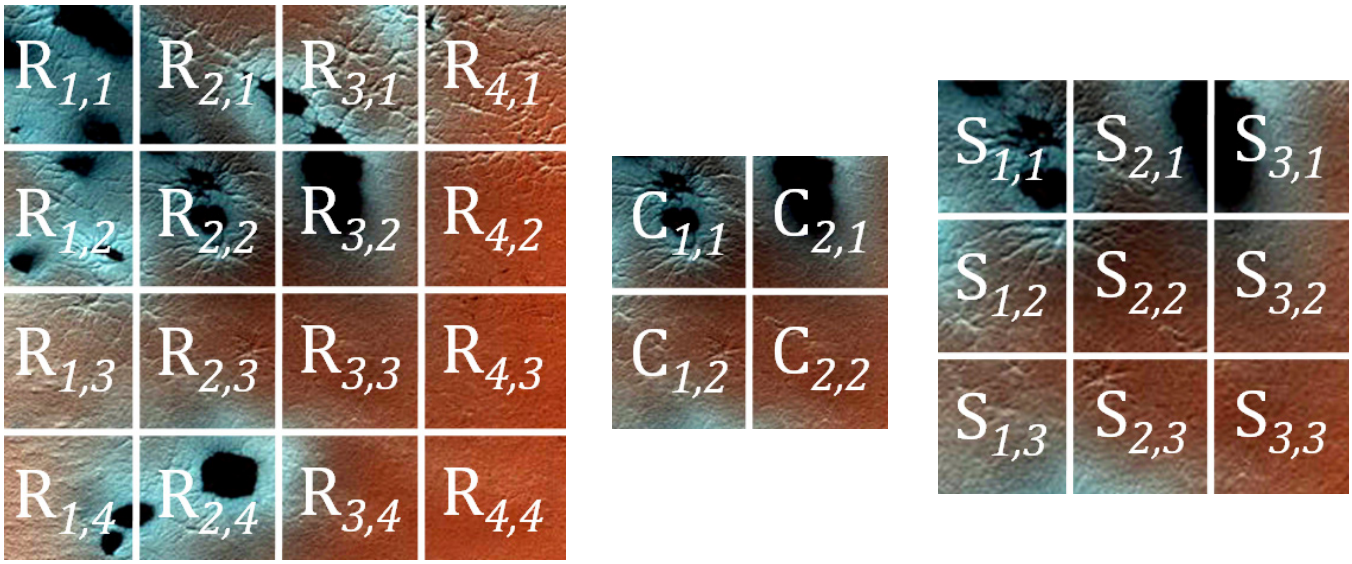


Figure 1 – Scalable Image Operation Tiles. ***The tiles from the three image operations in the pipeline are shown.***

In this section we discuss and formalize a design pattern called the Deferred Execution Pattern which is used as a basis for the design of our Scalable Image Processing Framework. The Deferred Execution Pattern is an adaptation of the Java Advanced Imaging model. We demonstrate how this design pattern is applied to our Scalable Image Processing Framework.

The deferred execution pattern allows us to process pixel information only when needed, avoiding any unnecessary computations. In the design pattern, image operations are chained together where the read tile operation is declared first and the write tile operation is declared last. There can be any number of image operations chained in between the read and write operations as shown in the diagram. Pixels are not loaded until the write operation is invoked. This gives the illusion that the image operations are performed with no time (or deferred executed). The diagram illustrates that when the write() method is invoked the ScaleOperation requests the cropped tiles needed from CropOperation which will make a tile request to TileOperation. The TileOperation makes a request to the ReadOperation where the codestream is randomly accessed and the data needed to make the requested tile is read. This pattern allows us to process pixel information only when needed. A formal example is shown. In figure 1 we have three scalable image operations chained together such that

ReadOperation ⟵ CropOperation ⟵ ScaleOperation

where the tiles of each operation are denoted as $R_{i,j}$, $C_{i,j}$ and $S_{i,j}$ respectively. In this oversimplified example the crop operations parameters are set to crop the image starting at location (256, 256) with the width and height (512, 512). Therefore assuming tiles of size *256 x 256* the read operation will simply read in the tiles $\{R_{2,2}, R_{3,2}, R_{2,3}, R_{3,3}\}$ to complete the crop operation. The scale operations parameters are set to scale the cropped image by a factor of 1.5 on the x and y axes. Now we will describe the image operation pipeline using the notion of our deferred execution pattern. To make the first scale tile $S_{1,1}$ a request is made to retrieve the crop tiles needed. In this simple example the only crop tile needed is $C_{1,1}$. Therefore a request is made from the crop operation to read in $R_{2,2}$. The process is then repeated until the scale operation is finished. One can see that $S_{2,2}$ is a slightly more expensive tile to make as the four crop tiles $\{C_{1,1}, C_{2,1}, C_{1,2}, C_{2,2}\}$ are needed in the computation. Furthermore every scale tile $S_{i,j}$ is written to disk after being processed to maintain scalability. It is also easy to see that the tiles in the operation pipeline are computed independently from one another allowing for them to be intuitively processed in parallel.

4. Design and Implementation

The Scalable Image Processing Framework uses the JPEG 2000 wavelet based standard. JPEG2000 has many advantages over the other image standards such as flexibility of the code-stream, intrinsic support for tiles, virtually unlimited file size, compression performance (wavelet based), and support for floating point numbers [7, 8]. A feature that is very important to us is the random code-stream access and processing. This allows us to perform operations such as rotation or scaling on random parts of the code-stream without having to read in the entire image into memory.

We use the Kakadu JPEG2000 encoder and decoder software library. The Kakadu Java Native Interface allows us to make calls to the native C++ libraries. We created several supporting classes to read and write JPEG 2000 images in a scalable fashion. Kakadu is the first and only available implementation of the complete standard. The Kakadu SDK has been used in medical imaging applications, geospatial imaging applications, and many other applications. We were tempted to use JJ2000 (a reference implementation of the JPEG2000 codec written in

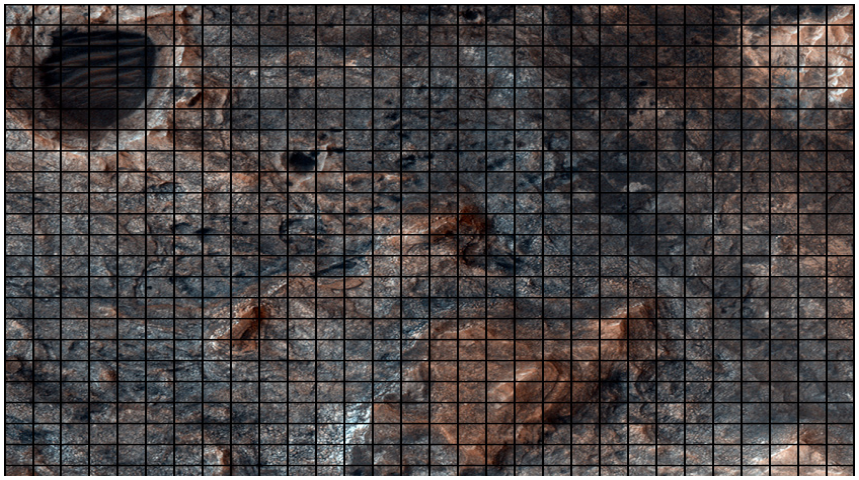


Figure 2 – Image tiles are an intrinsic property and only read into memory when they are needed by an image operation.

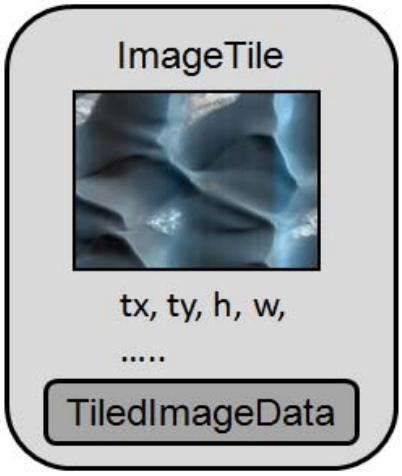


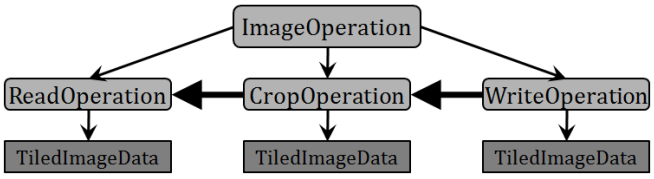
Figure 3 – ImageTile object.

Java) but we discovered problems with the random code-stream access. Additionally, we found Kakadu to be much more efficient in regards to encoding and decoding. We plan to eventually provide support for the JPEG 2000 Interactivity Protocol (JPIP) which is a client-server communication protocol used to view images or parts of images in a networked environment through randomly accessing the codestream. There is currently no support for writing defined in JPIP (part nine of the JPEG standard) or we could have used this as a basis for our scalable image processing framework.

4.1 Framework Components

The main components of the scalable image processing framework are the ImageTile, TiledImageData and ImageOperation classes.

The ImageTile object stores the tiles pixels and the tiles coordinates (i,j) as well as the pixel coordinates [x,y] with respect to the image operation. The ImageTile object also has a copy of the specific ImageOperation's TiledImageData object and various other intrinsic attributes of the tile. This class also provides useful methods used to retrieve and copy data within the image plane. It is important to note that every image operation will create a set of tiles where the tile coordinates and pixel coordinates depend exclusively on the operation that is being performed.



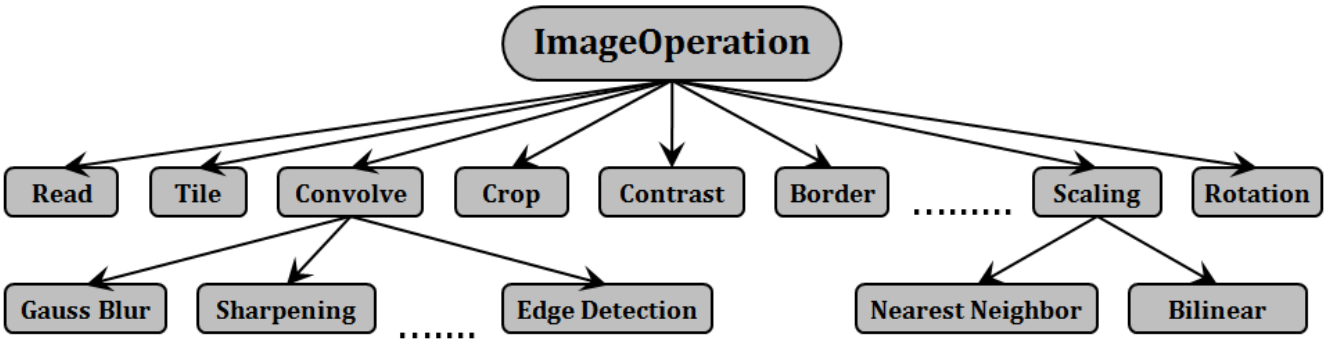
The TiledImageData object describes the entire image to be tiled with respect to the image operation in the pipeline. When an image operation is invoked a TiledImageData object is created with the appropriate attributes such as the image bounds (x, y, w, h) where (x, y) are the starting coordinates and (w, h) is the width and height of the image. Therefore if a scaleOperation is invoked a TiledImageData object is created with bounds (x, y, ws_x, hs_y) where s_x and s_y are the scaling factors. This also automatically updates image properties that are derived from the image plane such as the number of tiles in both x and y directions. The writeOperation will request a tile from the preceding image operation in the pipeline by calling the getTile(i, j) method in the specific image operations TiledImageData object. When the tile is returned it is written to disk and the process is repeated.

4.2 Reading and Writing Tiles

Tiles are read only when they are requested by an image operation. The read randomly accesses the codestream and decompresses the appropriate image region. The data is then stored in a tile object and sent to the requesting image operation. The tile will also be stored in the cache until it is removed with respect to the cache's eviction policy or used again by another operation. Conversely when a tile has been completely processed by the requesting image operations it is compressed and stored in the appropriate codestream location. The data is then written to disk. After the data is written to disk the tile is discarded.

The Scalable Image Processing Framework:

(1) Reads in a tile only when requested by another image operation. The tile size we use is generally *256 x 256 pixels* but can be set to an arbitrary size.
(2) Performs the appropriate image operation on the tile.
(3) Saves the result to the cache and the tile is sent back to the calling operation.
(4) If the resulting tile has no more image operations to be performed on it the tile is written to disk.
(5) Discards any tiles that are no longer needed.
(6) Repeats the process.



4.3 Tiling Operation

The tiling operation handles creating the tile object with the appropriate attributes. The pixels are read into the tile through the ReadOperation using the random code-stream access. The tile is then stored into the cache and passed back to the image operation that requested it. In the Mars photo in figure 2 the black squares represent tiles. Every tile has a set of tile coordinates that can be referenced and a set of x,y coordinates relative to the entire image. As an example, tile $T_{1,2}$ would have starting x,y coordinates at position [0,256] and so forth.

The Scalable Image Processing Framework uses Ehcache [13] to store the processed tiles so they can be retrieved rapidly in the future. We chose to use Ehcache because it is fast, simple, scalable, supports memory/disk stores into the gigabytes, and provides distributed caching. Once in the cache the tiles can be repeatedly accessed inexpensively. Our cache uses the Least Frequently Used eviction policy. This algorithm keeps track of when the tiles were last used and discards them based on which ones are not used frequently. Caching speeds up things by using the notion of locality of reference; data that is near other data or has just been used is more likely to be used again. The cache is defined in the parent ImageOperation class and every tile produced by the children operations (Tiling, Scaling, Sharpening, Convolution, Rotation, Cropping, Edge Detection, ….) is stored in the cache for use with other image operations. When an image operation is invoked through an API call a unique random number is generated. We use the image operations name, tile coordinates and the random number as a key for a specific tile.

4.4 Implementing Scalable Image Operations

An image is read and processed in the form of tiles. These tiles are an intrinsic design component of our Scalable Image Processing Framework. All implemented image operations must process the image in the form of tiles. This is often difficult, as many image operations require the neighboring tiles pixels as shown in the mathematical framework. As an example, suppose we use the bilinear operation to resize the image by half of its size. Bilinear interpolation is performed using the neighboring pixel values to estimate the resulting pixel value. To make the first bilinear tile $B_{1,1}$ the operation needs four tiles $\{T_{1,1}, T_{1,2}, T_{2,1}, T_{2,2}\}$ from the original image.

All scalable image operations extend the ImageOperation class that provides generic functionality for all image operations. The ImageOperation class provides functions to retrieve tiles from the cache, manage the cache, and add inputs. Image Operations can be easily added onto the existing framework in a scalable fashion.

Scalable Image Operations are designed by implementing two simple methods. The first method addInput(…) is invoked when the user is chaining together operations using our API. This method only needs to be modified if the image operation that you are designing will change the geometry of the image such as rotation, cropping or any scaling algorithm. The second method is performOperation(TileX, TileY) where it requires the image operation to be designed in a scalable manner. Every call to this method by getTile(i, j) needs to be implemented in a way that processes only the tile $T_{i,j}$. The method Inputs.get() returns the image operation in the pipeline preceding the operation to be implemented. Therefore the call to getTile(i,

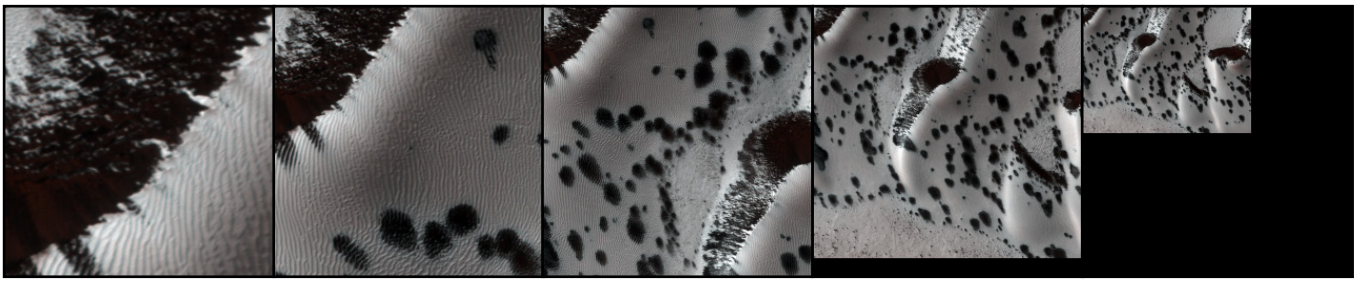


Figure 4 – Level of detail tiling. ***The first tile $T_{1,1}$ from every level of detail is shown where the leftmost tile is from the original image. A small image with only five levels of detail was used to demonstrate the algorithm.***

j) successfully returns the correct tile in the pipeline to be processed.

5. DISTRIBUTED AND CLOUD COMPUTING

We are using the Amazon Elastic Cloud Computing service as well as our own machines (at NASA JPL) to distribute the tiling and processing of images. This adds even more flexibility and scalability. The Amazon Elastic Cloud Computing is a web service that provides resizable computing capacity in the cloud. The cloud is elastic in that it can scale itself up and down in seconds depending on the needed resources of the Scalable Image Processing Framework. This provides a substantial increase in both speed and efficiency. A goal of ours is to eventually be able to process all MER (Mars Exploration Rover) images within a few hours. Implementing our image processing framework on the cloud is only natural as it is intrinsically scalable by the way of tiles where the tiles of a specific operation can be computed independently from one another.

The Maestro Science Activity Planner delivers tiles on demand to scientists only when needed by the current viewing area of the application. The tiles are sent over the internet to scientists in all facets of the world. To support viewing images at different levels of detail the following basic level of detail tiling algorithm is implemented using our application programming interface.

Let I be an n x m matrix of tiles $T_{i,j} \in \Re^{256x256}$. A one pixel border is created around I therefore the pixels in every tile $T_{i,j} \in$ I are shifted by one in the x and y directions in the image plane and written to disk. The image I is scaled by half in both the x and y axes using bilinear interpolation therefore I′ is an n′ x m′ matrix of tiles where n′ = n/2 and m′ = m/2. The process is repeated with the image I′ until it only contains one tile $T'_{n',m'}$ as shown in figure 4.

The border operation is needed for scaling the tiles at levels higher than their native resolution. Mars Rover images often need to be scaled by a factor of two or more times to carefully target the in situ science instruments. The border operation creates a tile $T_{i,j}$ where each tile overlaps with its neighbor by one pixel on each side for the interpolation to work properly without leaving artifacts when rendering the tiles in the viewer. Every tile is rendered as if it is two pixels smaller in the x and y directions.

This method of tiling an image for multiple levels of detail is processed in a scalable fashion automatically with our scalable image processing framework. It is important to note that the scaling at every level of detail is incremental for both speed and the difference in quality that we noticed. Assuming k levels of details (LODs) where the k level of detail refers to the level of detail from the original image. Therefore if we have an image with k levels of detail the k – 1 level of detail would need the scaled image from the k level of detail. This algorithm is used as a benchmark for our software. The images in figure 5 have been processed by our software using the level of detail tiling algorithm.

These images were selected because they are of interest to scientists (Future Exploration/Landing sites) and are some of the largest images. The Possible MSL Landing Surface Hazard image is 5.71GP (1,000 times the size of a standard 6MP consumer based camera).

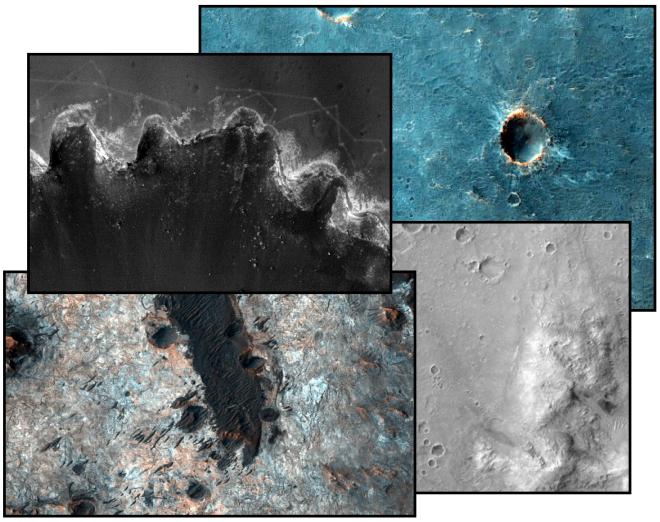


Figure 5 – Opportunity Rover Tracks at Victoria Crater (2.03GP [GigaPixels] : 67643 by 30015 : 1274.3MB), Surface Hazards of Possible MSL Rover Landing Site (5.71GP : 126021 by 45357 : 2047.4MB), Possible MSL Landing Site Mawrth Vallis (2.29GP : 71319 by 32248 : 1158.9MB) and Possible Location of Spirit Rover in Columbia Hills (1.17GP : 44364 by 26522 : 693MB).

We describe a few strategies for using distributed and cloud computing technologies to process gigapixel images using our Scalable Image Processing Framework. These strategies are described using the simple level of detail tiling algorithm for demonstration purposes. The strategies can be applied to various other problems that can be defined using our application programming interface.

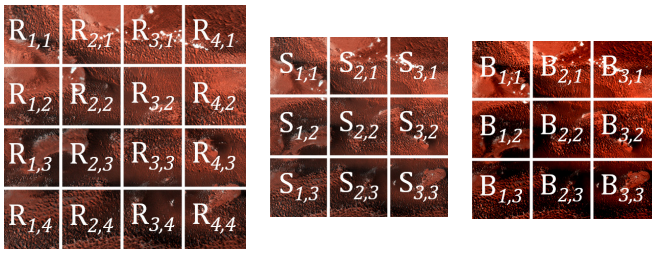
5.1 Processing Levels of Detail in Parallel

As a first instance we could have assigned every level of detail to a machine in the cloud. Since the scaling is incremental the level of details has a dependency requiring the scaling of the k level of detail to be completed before other level of details can begin processing. The processing time of all levels of details is bounded by the amount of time it takes to complete the k level of detail. Therefore given an arbitrary level of detail we can decouple the scaling and writing of tiles to disk. This strategy provides a decent solution where only a limited amount of machines are needed.

5.2 Distributing Regions of Tiles in the Cloud

A set of tiles from an image operation in the operation

pipeline can be assigned to a machine in the cloud where the number of machines is constrained by the time in which we need the task completed. This assumes the time it takes to process one tile in the operation pipeline is less than the time constraint. This strategy requires a job handler that on demand assigns machines to the processing of tile regions within the image given some time constraint.



In the above oversimplified example our image operation pipeline has only reading, scaling, brightness and writing operations. We are simply scaling the image down using bilinear interpolation with a factor of 0.75 in both the x and y axes and brightening the resulting image. Every tile $T_{i,j} \in \{R, S, B\}$ can be processed independently. Therefore we could assign a machine in the cloud to every tile in the last image operation in the pipeline. In this example we would assign a machine to each of the tiles $B_{i,j}$. We show a diagram of the process below.

The figure above shows that the tiles $\{B_{1,1}, B_{2,1}, \ldots\}$ are processed in parallel on machines in the cloud where the tiles needed from the other image operations in the operation pipeline are processed on demand and independently.

This strategy can be applied to the level of detail tiling algorithm. Suppose we are performing the level of detail tiling algorithm on an arbitrary image I of size n x m tiles where k is the maximum number of levels of detail. We still have the previously defined dependency where the k – 1 level of detail cannot begin until the k level scaling is completed. The solution is to distribute the scaling operation (Bilinear Interpolation) and the writing of tiles. The scaling and writing operations are easily distributable since they operate on tiles and not the image itself. Therefore we do not need to modify the operation or the framework. Informally we simply divide the image into regions and distribute these regions to machines on the cloud.

Let R be a matrix of tiles corresponding to an image region where (u, v) are the coordinates of the region with respect to the image matrix denoted as I and $n_r \times m_r$ are the width and height of the region, respectively. Therefore we have a new mathematical object I of regions where a region $R_{u,v}$ is a matrix of tiles $T_{i,j}$. Furthermore let c be the number of regions and consequently the initial number of machines. The number of regions can be defined as a function of time.

Every region in I is sent to a machine to scale the region and another machine writes the tiles in the region to disk. The more regions we have the less amount of time to process the image and consequently the more machines needed. As soon as these regions are scaled by half the next level of detail can begin writing the tiles and scaling these new regions.

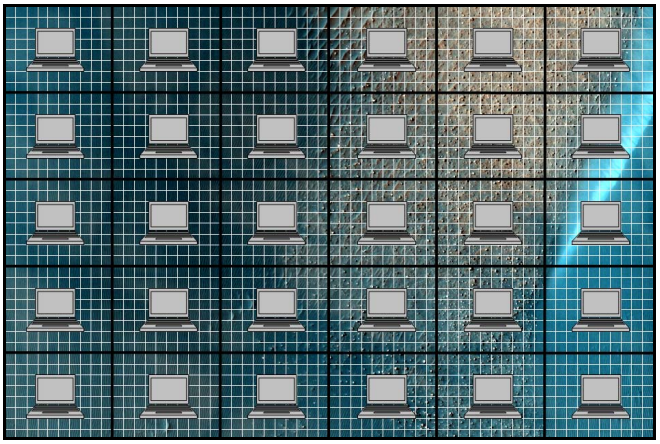


Figure 6 – Regions of an image are distributed and scalably processed by machines on a supercomputer.

Only the most significant level of detail from I should be divided into regions as combinatorial problems are encountered otherwise and resources are often wasted. At a certain level of detail we must stitch together the regions to avoid the size of the region becoming less than the size of a tile. This depends on the number of regions defined in the image. Given enough resources it is easy to see that by using this strategy the 6 gigapixel image (Mars Science Laboratory Landing Site Surface Hazards) could be processed within seconds.

6. Conclusion

We have developed a Scalable Image Processing Framework capable of performing image operations on gigapixel images. A mathematical framework for the scalable image operations is defined to give insight into the considerations needed with each class of image operations. We show how we used the Deferred Execution Pattern as a basis to design our Scalable Image Processing Framework. Distributed and Cloud Computing technologies are applied

naturally with our framework. The Scalable Image Processing Framework will be used to process incoming images from the Mars Exploration Rovers and eventually the Mars Science Laboratory. It will also be integrated with the Maestro software tools used to operate missions and technology concept studies for the Moon, Mars, and other celestial bodies.

7. FUTURE WORK

Image data is sometimes lost due to transmission problems when the data is sent from Mars to Earth using the Deep Space Network. Therefore an image region might be corrupt causing artifacts in the image or part of the image might even be lost. A future direction will be to explore the use of Singular Value Decomposition to automatically approximate the missing or corrupt pixel values. The error in approximation depends on the size of the region. If the region is small the approximation is likely to be very accurate. This would allow scientists to view data more accurately without obvious mistakes or artifacts.

We will also explore how to integrate this framework with advanced visualization hardware such as the large multi-touch display and CAVE augmented reality venues.

8. ACKNOWLEDGEMENTS

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and the NASA Undergraduate Research Fellowship.

REFERENCES

[1] Powell, M., Crockett, T., Fox, J., Joswig, C., Norris, J., Shams, K., Torres, R., Delivering Images for Mars Rover Science Planning, IEEE Aerospace, 2008.

[2] Fox, J., Norris, J., Powell, M., Rabe, K., Shams, K., Advances in Distributed Operations and Mission Activity Planning for Mars Surface Exploration, Jet Propulsion Laboratory, 2006.

[3] Powell, M., Crockett, T., Fox, J., Joswig, C., Norris, J., Rabe, J., McCurdy, M., Pyrzak, G., Targeting and Localization for Mars Rover Operations, IEEE Information Reuse and Integration, 2006.

[4] Norris, J., Powell, M., Fox, J., Rabe, J., Shu, I., Science Operations Interfaces for Mars Surface Exploration, IEEE Systems, Man, and Cybernetics, 2005.

[5] Norris, J., Powell, M., Vona, M., Backes, P., Wick, J., Mars Exploration Rover Operations with the Science Activity Planner, IEEE Robotics and Automation, 2005.

[6] McAffer, J. and Lemieux, J-M, Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications, Addison-Wesley 2005.

[7] Skodras, A., Christopoulos, T., Ebrahimi, T., The JPEG 2000 Still Image Compression Standard, IEEE Signal Processing Magazine, 2001.

[8] Kopf, J., Uyttendaele, M., Deussen, O., Cohen, MF., Capturing and viewing Gigapixel images, ACM Transactions on Graphics, 2007.

[9] McEwen, A., Delamere, W., Eliason, E., Grant, J., Gulick, V., Hansen, C., Herkenhoff, K., Keszthelyi, L., Kirk, R., Mellon, M. et al., The High Resolution Imaging Science Experiment for Mars Reconnaissance Orbiter, 33rd Lunar and Planetary Science Conference, 2002.

[10] Johnston, M., Graf, J., Zurek, R., Eisen, H., Jai, B., The Mars Reconnaissance Orbiter Mission, IEEE Aerospace Conference, 2005.

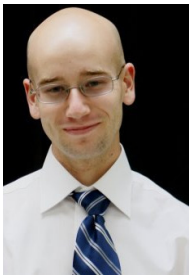
[11] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design patterns: elements of reusable object-oriented software, Addison-Wesley 1995.

[12] Taubman, D., Marcellin, M., JPEG2000: Standard for Interactive Imaging, Proceedings of IEEE, 2002.

[13] Luck, G., Ehcache 1.5 Guide & Reference, Lulu Publishing, 2008.

BIOGRAPHY

Mark W. Powell *is a Senior Member of Technical Staff at the Jet Propulsion Laboratory, Pasadena, CA since 2001. He received his Ph.D. in Computer Science and Engineering in 2000 from the University of South Florida, Tampa. His dissertation work was in the area of advanced illumination modeling, color and range image processing applied to robotics and medical imaging and received the award for Outstanding Dissertation from the University of South Florida. At JPL his area of focus is science data visualization and science planning for telerobotics. He supported the 2004 Mars Exploration Rover (MER) mission operations as a Science Downlink Coordinator, facilitating the timely downlink and analysis of science data from the rovers. He received the NASA Software of the Year Award for his work on the Science Activity Planner science visualization and activity planning software used for MER operations. He also received the Imager of the Year award from Advanced Imaging Magazine for his work on Maestro, the publicly available version of the Science Activity Planner for MER. Mark has been programming in Java and loving every minute of it since it was first used in web browsers in 1995. He, his wife Nina, and daughters Gwendolyn and Jacquelyn live in Tujunga, CA.*



Ryan A. Rossi *is a research assistant in the Operations Planning Software Research group at the NASA Jet Propulsion Laboratory and will be pursuing a Ph.D. in Computer Science at Purdue University. He received three graduate fellowships for his research in machine learning and artificial intelligence. His research is supported by the National Defense Science and Engineering Graduate Fellowship, National Science Foundation Graduate Research Fellowship and the Purdue Andrews Fellowship. He is sponsored by the Department of Defense and the Air*

Force Research Laboratory. He has worked on research developing machine learning algorithms for problems in security, link-analysis, search engines and bioinformatics. He was previously a research assistant at University of Massachusetts Amherst, New Mexico Tech and Coastal Carolina University.

Khawaja Shams *joined the Planning Software Systems group at the NASA Jet Propulsion Laboratory in 2005, and he has since been focused on development of OSGI-based web services to enable Maestro's rich client applications. His prior work experience includes employment at Malin Space Science Systems and the Internet Protocol Team at Nokia Mobile Phones. Khawaja earned a Master's degree in Computer Sciencetfom Cornell University, and a Bachelors degree in Computer Science from University of California, San Diego. Khawajaes current research interests include browser-based telemetry monitoring systems for robotics, peer-to-peer systems, and RESTful web based services.*