

COST SAVINGS THROUGH MULTIMISSION CODE REUSE FOR MARS IMAGE PRODUCTS

Robert G. Deen

Space Science Data Systems Section
Jet Propulsion Laboratory
Mail Stop 168-514
Pasadena, CA 91109
Bob.Deen@jpl.nasa.gov

ABSTRACT

NASA has sent, and continues to send, many missions to the Martian surface. These landers and rovers typically have onboard stereo cameras that take in-situ pictures of the surface. The Multimission Image Processing Lab (MIPL) at JPL has written a reusable software suite to handle these kinds of surface-based cameras. Called PIG (Planetary Image Geometry), it consists of an object-oriented, multimission framework which abstracts out elements common to all surface-based missions. To this are added mission-specific modules (subclasses). The application programs themselves, performing tasks such as mosaicking, pointing correction, stereo correlation, and terrain generation, do not need to be rewritten for each new mission, and contain no mission-specific references whatsoever.

The cost of adaptation to a new mission is very small compared to the cost to rewrite the application suite each time. Mission adaptation time has ranged from a few days to two months, compared to years to write the original code. This frees up a lot of resources that can be used to extend the library and provide additional functionality, rather than re-implementing the core functions each time.

So far the library has been adapted for use with Mars Pathfinder, Mars Polar Lander, the Mars '01 Athena Testbed, the FIDO test rover, and the Mars Exploration Rovers (MER). In the case of MER, basic capability was available even before ATLO, enabling use during testing and allowing time to work on functionality new to the mission, such as support for long-range traverses.

Several challenges to multimission use remain, however. Chief among them is the historic trend to redesign metadata (image labels) for each mission; support for this tends to be a driving factor in adaptation time. Future missions would

be well served to use existing metadata designs as much as possible in order to minimize costs.

An overview of the library's design will be presented, along with mission adaptation experiences and lessons learned, and the kinds of additional functionality that have been added while still retaining its multimission character. The application programs using the library will also be briefly described.

1. INTRODUCTION

NASA's exploration of Mars has increasingly focused on in-situ, landed missions. These landers and rovers have onboard stereo frame cameras that take images of the surface. The Multimission Image Processing Lab (MIPL) at JPL has been tasked with processing these images for science, operations, and public affairs. A number of products are derived from the images, including mosaics and terrain.

The current trend began with the highly successful Mars Pathfinder (MPF) in 1997. Mars Polar Lander (MPL) failed in 1999, but the ground system at MIPL was in place and ready to go. MIPL provided support for the Mars '01 Athena testbed (intended for a 2001 lander, which was cancelled), and the Field Integrated Design & Operations (FIDO) rover testbed. Most recently, MIPL is providing support for the twin Mars Exploration Rovers (MER), slated for launch in June, 2003. Looking forward, the Mars Science Laboratory and other landed missions are in the planning stages, which MIPL is planning on supporting.

While the missions have many similarities, each one is different, and the software used to produce the image products must be adapted for each mission. This paper describes how MIPL has reused the bulk of this software for each mission by creating a multimission framework.

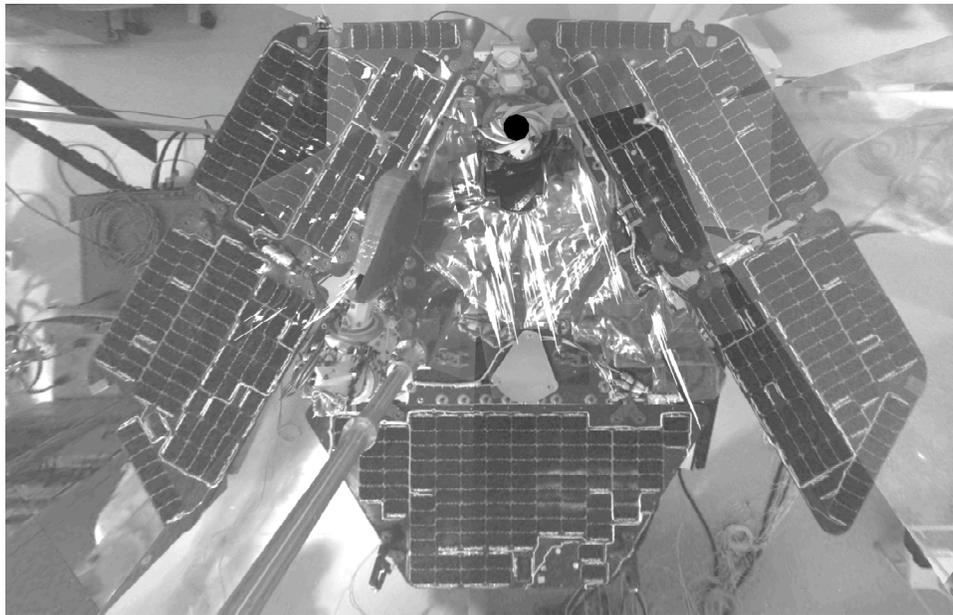
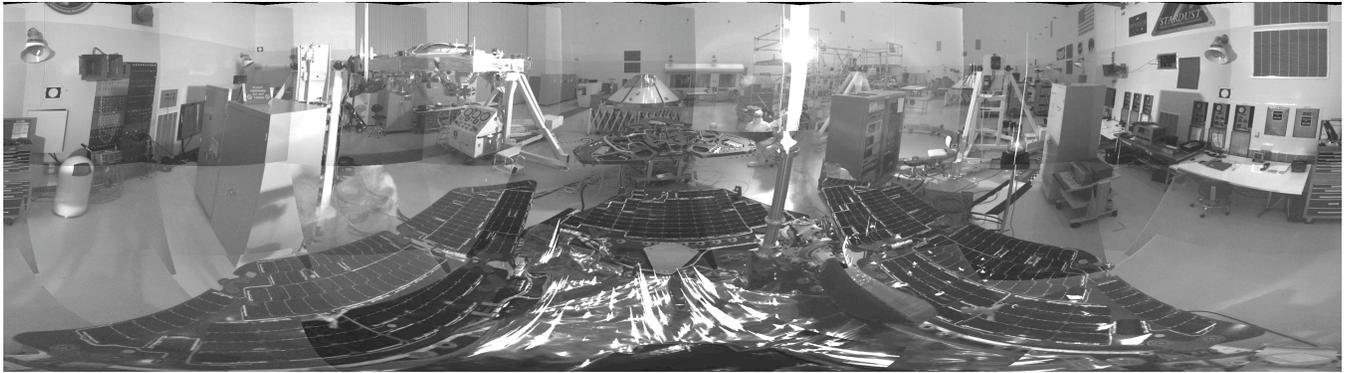


Figure 1: Mosaics taken by MER-2 during ATLO processing at Kennedy Space Center in March, 2003. Top is a 360° cylindrical panorama using 30 frames; bottom is a vertical projection (same data) showing the rover itself.

2. PROBLEM DOMAIN

Before describing the software itself, some background on the input images for these missions and the output products produced by MIPL will be helpful.

2.1. Input Images

The input images for all of these missions are similar. A CCD array in a frame camera produces monochrome images ranging from 248x256 pixels (MPF) to 1024x1024 (MER). Images typically have 8 or 12 bits per pixel. The cameras are either fixed to the body of the lander or rover, or are on an articulating device such as a mast or arm. Color images are produced from some cameras by a filter wheel in the optical path. [1,6,7,9] The images are often stereoscopic, i.e. pairs of left and right images taken at the same time from slightly different vantage points. Such stereo images are processed more than their non-stereo cousins, as terrain data can be derived from stereo pairs.

The input images all contain some sort of metadata (a.k.a. image labels) which describe the conditions under which the image was taken - exposure, pointing, compression, temperature, instrument modes, etc. [1,7,9] These labels are critical for determining how to interpret and use the images. For example, mosaics are assembled by looking at the label for each image to determine where the camera was pointing (e.g. mast articulation angles) and using that to control the mosaic ray-tracing process.

2.2. Output Products

There are many different output products that can be generated using the MIPL software suite described herein. Some examples will be shown, and the actual program names are in parentheses for reference.

Mosaics: The mosaics seem to be the most popular output products for science and public use. Since each input image has a relatively narrow field of view, many images

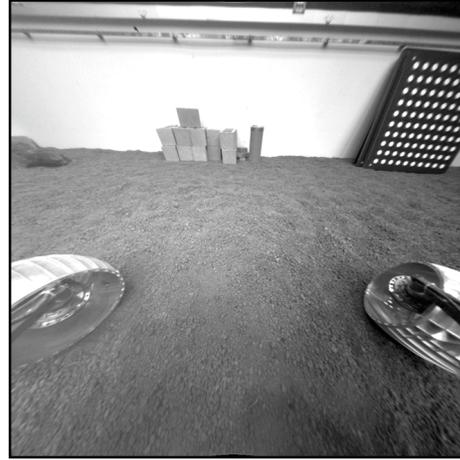
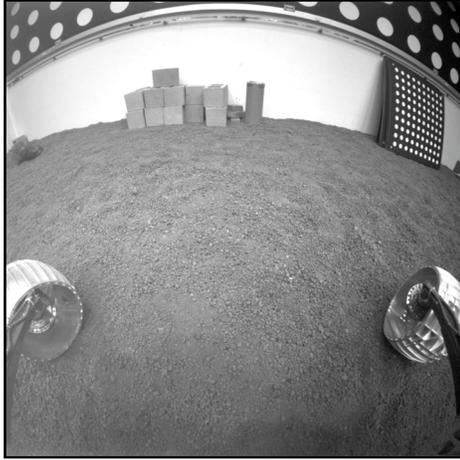


Figure 2: Images taken by a MER testbed rover by the left front hazard avoidance camera. Left is the raw image; right is the same image after linearization. Note how straight lines become straight once lens distortion is removed.

must be stitched together to create a panoramic view of the scene. It can take over 200 images to create a complete panorama for some instruments.

Mosaics are created by a ray-tracing process. Conceptually, the input images are projected out into space via their camera model (a mathematical model that describes the relationship between line/sample in the image and $x/y/z$ in 3-D space), laid on top of a surface (usually a flat plane), and then are projected back into the output mosaic. [2]

The output can be in one of five projections (using three programs): Cylindrical (marsmap), Polar (marsmap), Vertical (marsmap), Perspective (marsmos), or a hybrid Cylindrical-Perspective projection (marsmcauley). Each projection has its advantages and disadvantages. Examples of the cylindrical and vertical projections are shown in Figure 1.

Pointing Correction: The spacecraft’s knowledge of where its cameras are pointed is not precise. Mechanical backlash on articulation joints is a primary cause, but there are others. For this reason, several methods are available to correct the pointing by analyzing the images, using either automated/semi-automated (marstie, marsnav) or manual (MICA) methods. Correction can be based on physical modeling, or unconstrained. The results feed back into the mosaic program in order to reduce seams between frames.

Linearization (marscahv): Input images are often “warped” to remove lens distortion so that they can be described by a linear camera model. This model is epipolar aligned, meaning that lines in stereo images match up, making correlations easier. An example is shown in Figure 2.

Radiometric correction (marsrad, mosaics): This may be applied independently, or through the mosaic programs.

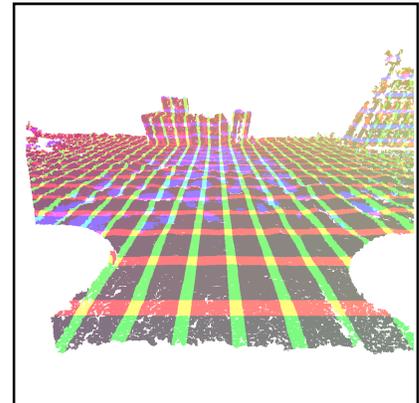
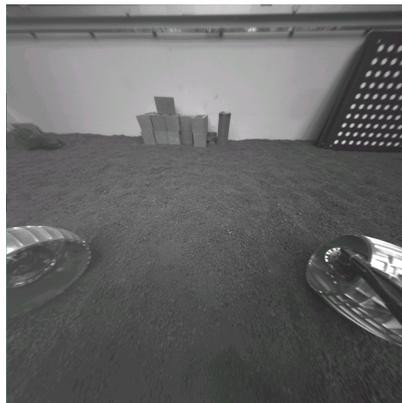
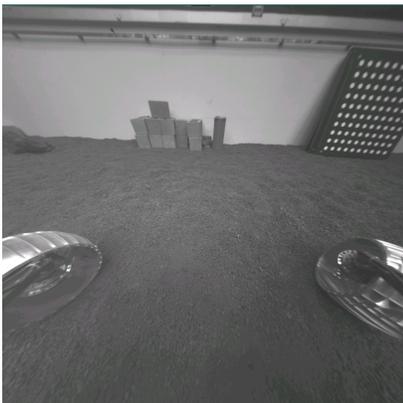


Figure 3: Stereo images from a MER testbed rover front hazard avoidance camera. Left: the linearized left eye. Middle: the linearized right eye. Right: the XYZ image derived from the stereo pair. A “contour” stretch has been applied in order to show the coordinate grid lines. Red lines represent constant X values, green lines represent constant Y, and blue lines represent constant Z.

The simplest example is exposure time compensation, but flat fields and temperature compensation are also used.

Disparity (marscorr, marscor2, marsjplstereo, marsseedgen): This is created from stereo pairs by correlating each point in the two images. A disparity image describes, for each pixel in one input image, where the corresponding pixel is in the other input image.

XYZ (marsxyz): This is an image derived from a disparity image and the camera models, describing the location of each pixel in XYZ (Cartesian) space. This is a primary product used by operations personnel for driving the rover on MER. An example is shown in Figure 3.

Range (marsrange): Derived from an XYZ image, this is simply the Cartesian distance from a point to each pixel.

Surface normal (marsuvw): Derived from XYZ, this is the surface normal for each pixel, computed over a small area.

Reachability (marsreach): Derived from XYZ and surface normal, this describes whether or not some instrument (e.g. instruments on the arm for MER) can reach the object depicted by each pixel.

3. HISTORY OF THE SOFTWARE

In early 1994, development began in MIPL on a set of software to accomplish some of the above tasks for Mars Pathfinder. The software was ready for use by the time MPF landed, after approximately 3 work-years of effort. The programs worked well, but they were specific to Mars Pathfinder. Constants such as camera models, image sizes, and mast articulation parameters were hard-coded in the software, algorithms were not flexible, and there was a lot of repetition of code among the various programs.

When development began for Mars Polar Lander, the basic requirement for MIPL could be summarized as “Do what you did for Pathfinder, just a bit better”. Knowing that future missions would also want to use the same kinds of capabilities, we realized that simply updating the programs for each mission would be quite expensive, not to mention error-prone due to the repetition of certain key functions.

After some analysis, we came up with a set of abstractions that seemed to work well for both MPF and MPL, and we thought they might apply equally well to future missions. So, we decided to restructure the MPF-specific code using these abstractions and the Planetary Image Geometry (PIG) library was born. While we were at it, we moved the code from C to C++.

The result was a set of application programs with absolutely no mission-specific references whatsoever. All mission-

specific code is encapsulated in PIG subclasses. Both MPL and MPF were supported in the first release. Since then, four additional missions have been added to PIG, proving the concept and saving a lot of development time.

4. MULTIMISSION DESIGN

The MIPL Mars in-situ image processing software is divided into two distinct pieces: the application programs themselves, and the supporting library, known as Planetary Image Geometry, or PIG.

4.1. PIG Library

The Planetary Image Geometry (PIG) library is a set of classes and an API that provides a common, multimission interface for retrieving and using image projection and pointing metadata. Its primary intent is to provide support for in-situ missions, although it should be extensible to orbital missions if needed. It provides abstractions for processing camera models, pointing models, coordinate system definitions, site/position information, Experiment Data Record (EDR) metadata (labels), surface definitions, and radiometric correction. [3]

PIG is divided into two parts: base classes, and mission-specific subclasses. The base classes provide the common interface, and are the only part the applications see. A representative selection of base and subclasses is shown in Figure 4 and several of the classes are described below.

4.2. Primary Classes

PigModelBase: All models derive from PigModelBase. This ultimate base class provides facilities for obtaining user parameters and printing messages. These facilities operate via callbacks so applications can override e.g. the message printing function so messages go into a GUI instead of the terminal. The model base also provides some utility functions such as configuration file access.

PigCameraModel: These describe the relationship between a pixel (line/sample) and a vector in 3-D space (X/Y/Z) containing points viewable by that pixel. Subclasses represent types of camera models. CAHV [8], CAHVOR [4], and CAHVORE [5] (acronyms based on mathematical model vectors) are types used in the Mars surface missions at JPL. Other types are possible, such as one based on focal length. The types represent the math used to compute the image to 3-D transformation. The term “camera model” is also used for the numeric parameters that describe a specific camera, both in its calibration position, and after pointing for a specific image. These parameters are provided when a camera model instance is created.

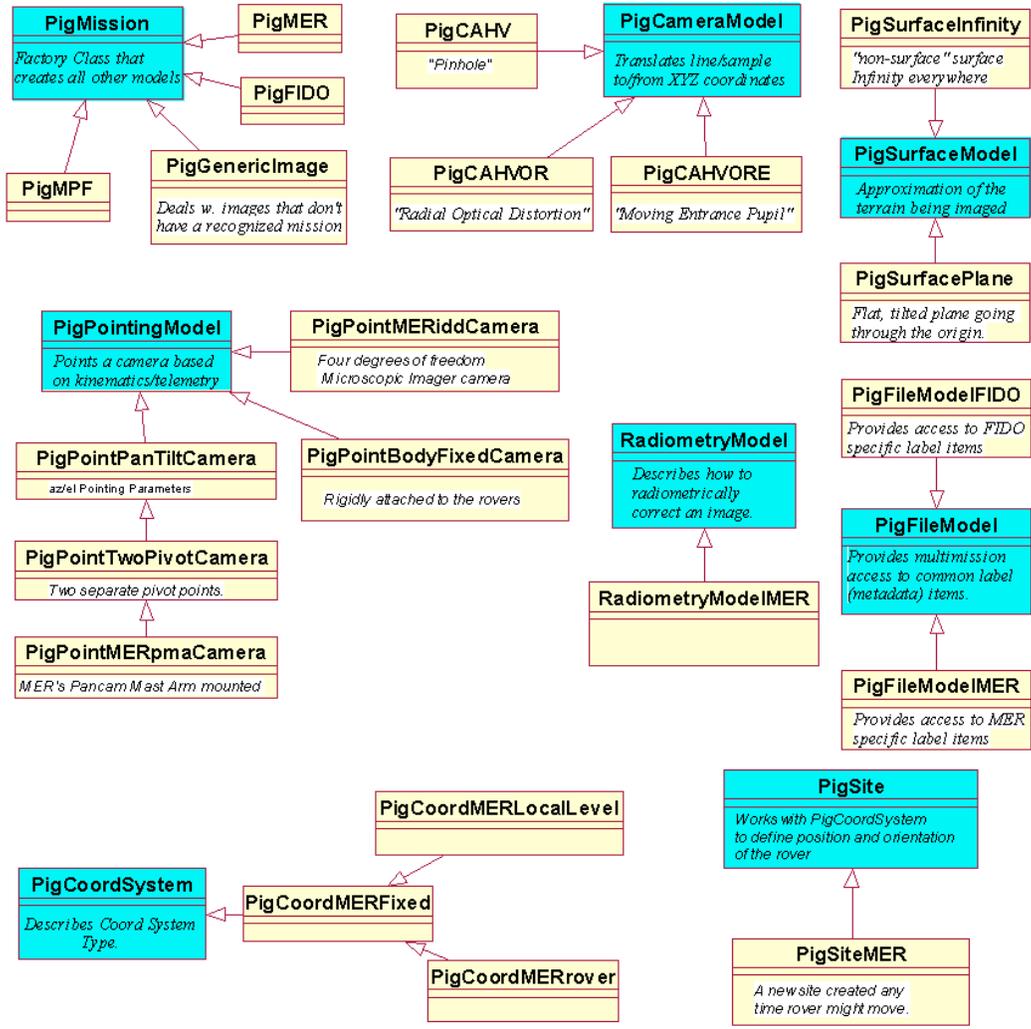


Figure 4: UML diagram depicting some of the major classes and subclasses in the PIG library. The diagram is representative, not exhaustive. For example, for each subclass named MER, there are analogous subclasses for most other missions.

PigPointingModel: These describe how to point the camera for a specific image. Subclasses are mission and camera specific. Each subclass knows how to extract pointing information from the image (actuator angles, spacecraft position, etc.) and use that to transform a calibration camera model parameter set into a model specific to a given image, which can be used for image to 3-D transformations. Pointing models also allow correction or adjustment of the pointing for an image. They expose their parameters via “pointing parameters”, which are a set of floating-point numbers that describe the pointing (e.g. a set of joint angles, or an azimuth/elevation of the camera). Each subclass can have a different set of pointing parameters. The key point is that applications can adjust these based on their effects on the image, without knowing how to interpret them.

PigSurfaceModel: These describe the ground and provide facilities to intersect view rays with it. Subclasses can include flat planes, infinity (no surface), spheres, and surfaces based on terrain models.

PigFileModel: These represent the input images and provide high-level access to the metadata, and the image data, contained within them. Subclasses are used to handle the inevitable differences in metadata across missions.

PigCoordSystem: These describe the orientation of coordinate systems, and their relationship to each other. Subclasses are used for each coordinate system defined by a mission – for example, Mars Pathfinder used Surface Fixed, Local Level, Lander, and Rover, while MER uses Site and Rover. Instances of these coordinate systems depend on a PigSite object and can be used to translate positions and

orientations from any coordinate frame to any other, at least within a single mission.

PigSite: These define the position of a movable object, such as a rover or lander, at one specific instant. They work closely with PigCoordSystem objects to provide coordinate frame conversions. The base class defines a site using a quaternion and offset; subclasses may be necessary if other representations are needed. PigSite has been greatly expanded for MER, where long-range traverses create many different Sites which must be tracked.

RadiometryModel: These describe how to correct the radiometry (or brightness) of an image. This can include things like dark current, flat field, exposure time, and temperature correction. Subclasses are generally per mission, or instrument.

PigLabelModel: A recent addition, these classes handle writing output labels (metadata) for each type of image product. Subclasses exist per mission, where the metadata format deviates from the “standard” output labels.

PigMission: Mission objects contain factory methods which create all of the other objects described above. Subclasses exist per mission, and know which specific subclass to create for any given occasion. Subclasses often examine the metadata of an image (via the PigFileModel) to determine which subclass to create (e.g. which instrument generated the image).

Component	Approx. Lines of Code
Applications	66,500
PIG library (total)	27,700
PIG multimission base	14,600
PIG MPF	2,100
PIG MPL	2,800
PIG M01	1,200
PIG Generic	1,300
PIG FIDO	2,600
PIG MER	3,100

Table 1: Lines of Code for MIPL in-situ image processing

4.3. Application Programs

As described previously, the application programs contain absolutely no mission-specific code. They work the same regardless of the mission. Furthermore, when application capabilities are enhanced, old missions are able to take advantage of the new features just as well as the new missions. The available applications are described in Section 2.2.

Lest one think that the applications are trivial, and all the code is in PIG, Table 1 shows the lines-of-code breakdown

for the software suite as of the time this paper was written. The application code is almost 2.5 times larger than the PIG library itself,

5. ADAPTATION EXPERIENCES

The PIG library has now been adapted to work with data from 6 distinct “missions” (5 real missions and a “generic” mission). Adaptation times have ranged from 2 days to a few months. Compare this with 3 years to write the original code and one finds that new missions can be supported in about 1/20 the time it took to write the original library. While algorithm development certainly contributed a lot to the time required to write the original code, the difference is still dramatic. Adaptation of the original code, without the PIG library, would probably take 4-5 times longer, and be more error-prone due to the required duplication of effort and divergent versions.

As shown in Table 1, each mission averages just 2200 lines of mission-specific code. Compare that to 14,600 for the PIG base and 66,500 for the applications, and it should be obvious the extent to which the multimission framework has saved time, money and effort.

5.1. Mars Polar Lander/Mars Pathfinder

These two missions are lumped together because they were developed simultaneously, along with the basic PIG framework. This makes determination of the time to do either adaptation by itself nearly impossible. The entire MPL task, which included creating the abstraction layer, writing PIG, adapting to MPL and retrofitting MPF, and adding additional application functionality required by MPL, took approximately 1 work year. It is this author’s estimate that the MPL adaptation, if done separately with the framework in place, would have taken perhaps 6 weeks, and MPF perhaps 3 weeks.

5.2. Mars ’01

Before MPL failed, there were plans for a lander in the 2001 time frame. A testbed for this was actually built, and the PIG library was adapted to work with this testbed. Owing to the similarity to MPF and MPL and the heavy use of cut-and-paste, the adaptation took (by actual measurement) 2 days.

5.3. Generic “Mission”

In order to support ad-hoc images, a generic “mission” was developed, which requires that the camera model be present in the image label or ancillary file, but no other information is needed. This adaptation took an estimated 1 week.

5.4. FIDO

The FIDO testbed rover is a close analogue to MER and was used to test concepts for MER. Support for this took about 3 weeks.

5.5. MER

Many enhancements have been made to the PIG library and applications for MER. Most of these are in order to support additional capabilities and requirements that MER has but previous missions don't (see Section 6). As such, it is hard to estimate the pure adaptation time (plus, as of this writing, it is not entirely complete), but it is probably around 2 work months, most of which has been spent dealing with a redesigned label format.

One nice thing about MER is that we were able to use the generic "mission" immediately with the very first images. Thus we had much of the MIPL functionality available even before ATLO, which has helped tremendously in early testing. As the development team got MER-specific capabilities working, the results simply got better.

6. EXTENDING THE LIBRARY

One measure of the quality of a design is how easy it is to add new capabilities or features to the design after the baseline has been built. Assumptions made early on in the design process can come back to haunt you if additional requirements change those assumptions. It may be easy, or almost impossible, to adapt to such changes. On this score, the PIG library design has been successful so far. To illustrate, a few of the more significant enhancements are described here.

6.1. Coordinate systems

The first version of PIG implicitly assumed one coordinate system would be used for all 3-D coordinates used in the geometry calculations. About 2 months before the landing of MPL, the science team decided they wanted to change the definitions of several important coordinate systems, with the result that they were no longer compatible and conversions would be required. A month of intense effort followed, resulting in an overhaul of the system such that every 3-D coordinate is now tagged with the coordinate system in which it is measured, and conversions between systems are handled automatically by the framework. The modifications were ready in time for operations.

6.2. New Camera Model Type

Previous missions used the CAHV (linear) camera model, [8] and the CAHVOR (adds radial distortion) model [4]. However, MER has extremely wide field-of-view hazard

avoidance cameras (close to 180 degrees), which cannot be successfully modeled by CAHV or CAHVOR. A new type of model that handles fisheye and wide field-of-view cameras, CAHVORE [5] (developed elsewhere at JPL), was integrated into the PIG framework in about 1.5 weeks.

6.3. Multiple Sites

MER is a long-range rover. As such, it can travel to areas out of view of the original landing site. In order to deal with this, the concept of multiple Site frames was introduced. Each Site frame is a reference for all activities contained within the Site. The support for this is quite involved and includes maintaining XML files containing the Site locations, as well as locations of interest within the Site. This concept fit rather well into the PIG library, at a cost of perhaps 4 work months. Interestingly enough, the application modifications to support this were extremely minor (mostly adding a few parameters and help updates); most of the changes are encapsulated in PIG itself, and are transparent to the applications.

6.4. Output Label Models

The metadata (labels) for MER are a radical departure from previous missions (see Section 7). While the input side was handled using the existing PigFileModel, output of labels had previously been something the applications themselves did. It quickly became obvious that a PIG model was needed to handle output labels as well, allowing them to be different for different missions. Total time to implement this, including the model itself and all the actual output labels for MER, was approximately 4 work weeks.

7. LESSONS LEARNED

While the PIG library concept and implementation have performed admirably, with huge cost savings, there are a few lessons that can be learned from the experience, which could save even more money in the future.

7.1. Labels (Metadata)

Most important is the design of the image labels (metadata). Historically, each mission has redesigned their label structures virtually from scratch. Label contents are often the subject of heated debate among the operations and science teams, and it is all too easy to depart from existing norms in order to make this mission "better". This wreaks havoc with multimission designs; a lot of new code must be written to accommodate the vagaries of label structures.

Well over half of the MER adaptation time, and most of the time spent creating the output label model, is attributable to changes in the MER label structure with respect to the "baseline" we hoped would be established by MPL. That's

easily over two work months just in implementation, not counting the time spent designing, debating, and documenting the label changes.

As a counter-example, the extraordinarily fast adaptation time for M01 was largely due to the fact that no new labels were designed; the MPL label structure was simply re-used.

Missions will be well served in the future to simply adopt existing metadata standards with only minor modifications. This should help *all* multimission programs, not just the MIPL software suite.

7.2. Other Lessons

In hindsight, it is easy to say that these programs should have been implemented using a multimission framework from the beginning, for Pathfinder. However, that may not have been practical. Experience derived during algorithm development in that first program set was critical in determining just what abstractions were necessary and what didn't make sense. Spending time to create a framework for functionality that is later discarded is just time wasted.

Thus, developing the algorithms first, then going back and making them reusable, seems to have been the right idea for this software set, at least. You do have to know *what* you're trying to build, before you can figure out the abstractions that will make the code reusable and adaptable.

It is worthwhile taking the time to thoroughly analyze the situation before creating a reusable framework. This author studied the situation for nearly 2 months, becoming familiar with the MPF code, before beginning the actual design and implementation of the PIG library. As detailed elsewhere in this paper, this forethought appears to have paid off.

The generic "mission" has come in quite handy. As mentioned in Section 5.5, it allowed MIPL to process MER data even before ATLO, before MER-specific development had begun.

8. CONCLUSION

The multimission framework embodied in the PIG library has had a tremendous impact on the ability of MIPL to quickly and inexpensively support new missions. As Table 1 shows, only about 2-3% of the code base needs to be touched in order to support a new mission. This is reflected in the adaptation times described in Section 5. New missions can be supported in approximately 1/20 the time it took to write the original library. This cost savings can either be returned to the customer, or invested in improving the products themselves via better algorithms or new features — an activity that benefits all prior missions as well as the one under development.

9. ACKNOWLEDGEMENTS

The original MPF code was developed by Jean Lorre. The PIG library was designed by Bob Deen. Contributors to the library also include Oleg Pariser, Justin Maki, and Anton Ivanov. All are from JPL's Space Science Data Systems Section (382). Development funding was provided by JPL's Science Instrument Services, and the Projects.

10. REFERENCES

- [1] Alexander, D., *et al*, "Mars Exploration Rover Project Software Interface Specification (SIS) Camera Experiment Data Record (EDR) and Reduced Data Record (RDR) Operations Data Products". JPL D-22846, 2003.
- [2] Deen, R., O. Pariser and J. Lorre, "Creation of Surface-Based Image Mosaics for MER/FIDO". JPL IT Symposium poster session, Nov. 4, 2002.
- [3] Deen, R. and H. Mortensen, "Planetary Image Geometry (PIG) Application Programming Interface (API) and Library Software Requirements Document (SRD) Version 1.0". JPL Internal Document, Dec. 3, 2001.
- [4] Gennery, D.B., "Least-Squares Camera Calibration Including Lens Distortion and Automatic Editing of Calibration Points", *Calibration and Orientation of Cameras in Computer Vision*, Springer-Verlag, 2001, p.123-136.
- [5] Gennery, D.B., "Generalized Camera Calibration Including Fish-Eye Lenses", to be published, 2003. <<http://eis.jpl.nasa.gov/~telitwin/public-jpl/src/ccal/ccal-references.html>>.
- [6] Maki, J.N. et al, "The Mars Exploration Rover Engineering Cameras", *Journal of Geophysical Research - Planets*, MER special issue, publication pending (2003).
- [7] Runkle, R., "Mars Pathfinder Project Imager for Mars Pathfinder (IMP) Experiment Data Record (EDR)". JPL D-12003, 1998.
- [8] Yakimovsky, Y. and R. Cunningham, "A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras", *Computer Graphics and Image Processing*, vol, 7, p. 195-210, 1978.
- [9] Zamani, P., "Mars Surveyor Project Mars Volatiles And Climate Surveyor (MVACS) Experiment Data Record (EDR) Software Interface Specification (SIS)". JPL D-17891, 1998