Software Design Specification for

MONTAGE

An Astronomical Image Mosaic Service for the National Virtual Observatory

Version 2.1 (August 29, 2004): Detailed Design

This Document is based on the template CxTemp_SoftwareDesignSpecification.doc (Draft X; August 9, 2002), released by Construx Software (www.construx.com)

MONTAGE DESIGN REVISION HISTORY

Description of Revision	**Date**
Add MPI design, MPI flowcharts, revise existing flowcharts. Move API & return codes to web page; update delivery dates and version numbers; add design for fast algorithm and new co-addition algorithm. – Version 2.1	August 29, 2004
Detailed Design: revised to reflect architecture for Montage on the TeraGrid – Version 2.0	January 13, 2004
Detailed Design: add algorithm description, flow charts, interface specification, error handling methodology, and all success and error return codes; revised block diagrams of Montage design & process flow – Version 1.1	November 18, 2002
Initial Design – Version 1.0	August 9, 2002

Table of Contents

1. Introduction

1.1 Purpose of this Document

This document describes the design of Montage, an astronomical image mosaic service for the National Virtual Observatory. It will deliver on-demand, science-grade, astronomical image mosaics that satisfy user-specified parameters of projection, coordinates, size, rotation and spatial sampling. Science-grade in this context means that the impact of the removal of terrestrial and instrumental backgrounds on the fidelity of the data is understood and removed over different spatial scales. The service will deliver mosaics of images released by the 2 Micron All Sky Survey (2MASS), the Digital Palomar Sky Survey (DPOSS) and Sloan Digital Sky Survey (SDSS) projects. in addition, since the service will support all common projections and coordinate systems, it will therefore be capable of delivering image mosaics from observations with all major astronomical imaging surveys and CCD imagers on ground-based telescopes

A fully documented, portable version of the software will be publicly available for running on local clusters and individual workstations. The compute engine will be quite general and will allow users to build mosaics from their own data. **In addition,** Montage will run operationally on the *TeraGrid* [1]*[1]*. Users will submit requests to Montage through existing astronomical portals, and visualize and interact with the delivered mosaics through these same portals.

1.2 Schedule for Delivery of Software Design Specification for Montage

The complete design specification of Montage will contain a description of

- design considerations
- overall architectural, “high-level” design
- how the high-level design satisfies the science requirements and use cases
- Low level design, including error handling strategies and algorithms implemented by the system

This document describes all aspects of the design except the API, which is specified in the on-line Users Guide.

The complete design specification will be delivered incrementally over several releases of this document, according to the schedule described in Table 1. The Montage project aims to deliver a complete, initial design specification before the first public release of the system (February 28, 2003) and will provide updates thereafter for subsequent releases. We anticipate that the design updates will be incremental and will largely accommodate customers’ requests for functionality.

Table 1: Schedule for Major Releases of Software Design Specification of Montage

SDS Version	Actual Release Date	Montage Version (Scheduled Release Date)	SDS Contents
1.0	6/30/2002	1.0.0 (2/28/2003)	Design considerations Overall architecture for serial processing Application of design to use cases
1.1	11/18/2002	1.1.0 (2/28/2003)	Design considerations Overall architecture Application of design to use cases Interface Specifications Low level design, incl. algorithms and error handling strategies
2.x	8/30/2004	2.x (2/28/2004)	Support parallel processing computing grids (principally the Teragrid), fast reprojection algorithm, and co-addition by reading image files from disk
3.0	9/30/2004	3.0.0 (1/10/2005)	Updates to 2.x to support final performance metric [

1.3 Supporting Materials

Montage will be developed according to the style guidelines in the Construx Software Project Survival Guide **Error! Reference source not found.**.

2. Design Considerations

2.1 Drivers and Constraints

The drivers and constraints governing the design of Montage are made clear in the requirements document [3] and in the Software Engineering Plan [4]. The most important drivers and constraints are as follows:

- Montage will be able to build small mosaics on a user's Linux laptop and be able to process many simultaneous, large requests on the *TeraGrid*. It must therefore be written to ensure portability, and make use of common tools and libraries that exist on all widely-used platforms.
- Montage must be a general service, capable of generating image mosaics according to user-specified size, rotation, projection and coordinate system.
- Montage must permit exercising the processing steps manually, and must deliver intermediate products, such as re-projected images, as science products in their own right.
- Montage must permit, without modification to the reprocessing and co-addition engines, rectification of background radiation to a common level, with parameters derived from rectification algorithms supplied by data providers, and background rectification from user-supplied parameters.
- Montage has strict performance requirements, and must be scaleable. It must deliver custom mosaics from 2MASS, DPOSS and SDSS with sustained throughput of 30 square degrees (e.g. thirty 1 degree x 1 degree mosaics, one 5.4 degrees x 5.4 degrees mosaic, etc.) per minute on a 1024x400Mhz R12K Processor Origin 3000 *or machine equivalent* with a sustained bandwidth to disk of 160 MB/sec.

2.2 Use of Open Source Software

Montage will use a small set of standard open-source astronomical libraries for reading Flexible Image Transport System (FITS) image files, performing coordinate system transformations, and handling image projection/de-projection operations. These libraries are portable and well-tested, and a current version will be delivered with all releases of Montage. The libraries are:

Library	Description	Current Release	Origin
CFITSIO	FITS reader	Version 2.420	HEASARC
WCSTools	Image projection	Version 3.5.2	SAO
boundaries	Boundaries around sets of points on the sky	Version 1.0	IRSA
coord	Coordinate transformation	Version 1.5	IRSA
mtbl	ASCII table reading	Version 3.2	IRSA

pixbounds	Boundaries around sets of points in the xy-plane	Version 1.0	IRSA
svc	Process forking and control	Version 1.5	IRSA
two_plane	Plane-to-plane transformations in the tangent plane	Bundled with `mopex_S9.5.0`	Spitzer Space Telescope

2.3 Portability of Montage Software

To ensure maximal portability and use of Montage, it will not use shared memory, specific DBMS interfaces, or platform-specific libraries, and it will minimize its use of memory (as long as it does not compromise the quality and range of the algorithm). Ancillary information, such as tables of information on the collection of images that are being processed, will be captured in simple text files in column delimited format; these files can be parsed by any computer.

Montage will be constructed to operate entirely from command-line arguments, using the ancillary files described above to communicate other information needed to process a request.

Montage will be developed in ANSI-standard C. It will be guaranteed to compile with GNU gcc and to build with GNU gmake. Other compilers and IDE's will almost certainly work just as well, though we make no guarantees about testing such and will only do so as resources permit.

Montage will be built on several UNIX platforms, including but not limited to Solaris, AIX, and Linux. It will be tested and run operationally on the TeraGrid.

2.4 System Environment

Montage should run on the following platforms and Operating Systems:

Machine	**OS**
TeraGrid (Operations)	Red Hat Linux 6.2
IBM Blue Horizon	AIX 5L
Linux Cluster	Red Hat Linux 6.2
IPG SGI O2K, O3K	IRIX 6.5.x
Solaris Workstations	Solaris 2.7, 2.8
Linux workstations	Red Hat Linux 6.2, 7.x
Apple	Mac OS X (Darwin 7.4. x)

3. High-Level Architecture and Computational Algorithms

3.1 High Level Design of Montage

Processing a request for an image mosaic consists of three main steps:

- re-projection of input images to a common spatial scale, coordinate system and World Coordinate System (WCS) projection;
- modeling of background radiation in images to achieve common flux scales and background levels;
- rectification of images to a common flux scale and background level; and
- co-addition of re-projected, background-corrected images into a final mosaic.

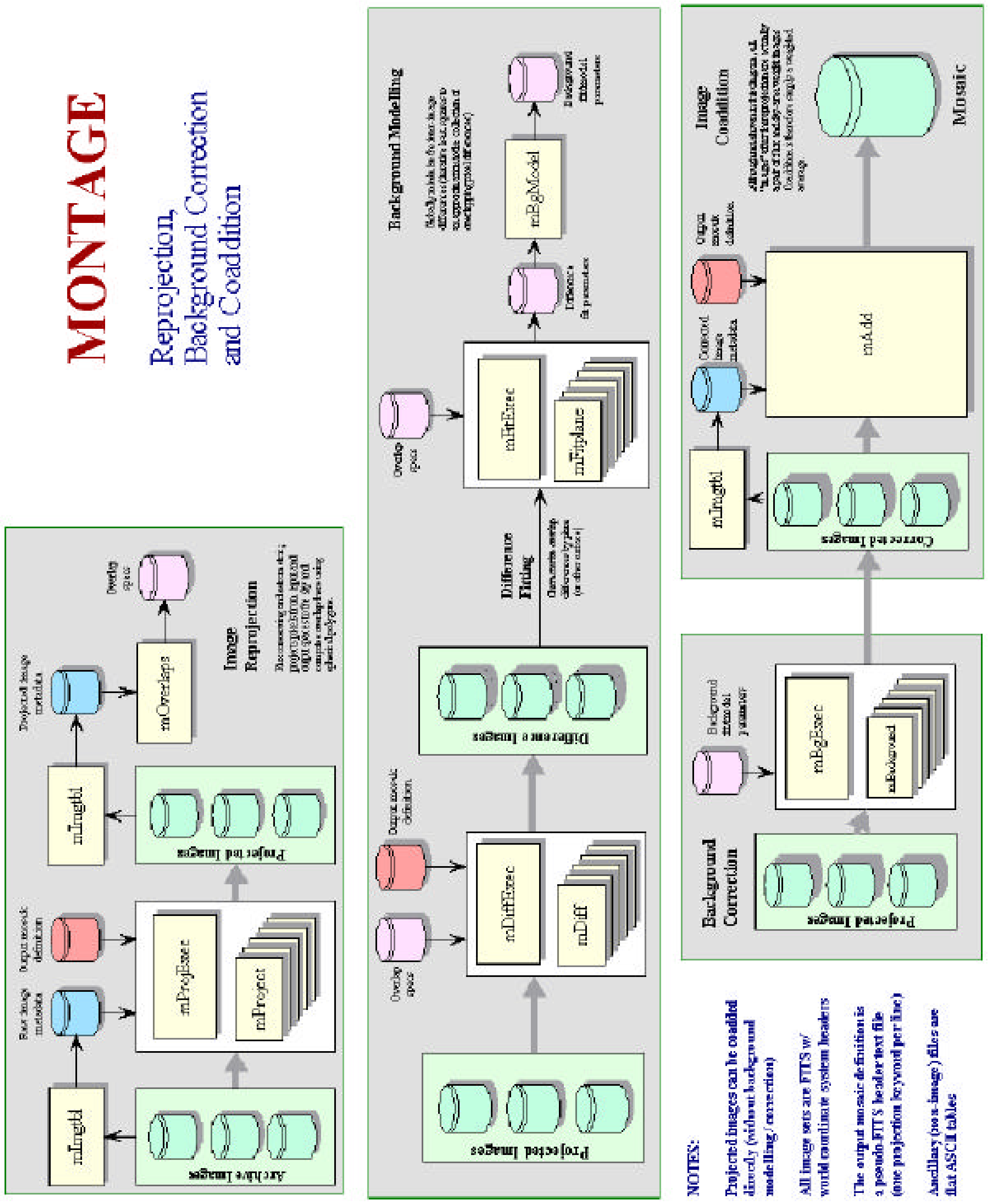
To accomplish these requests, Montage will consist of the following independent but interoperable components, illustrated in the block diagram in Figure 1:

- A compute engine that performs all re-projection and co-addition of input.
- A background modeling engine that globally minimizes the inter-image differences.
- A background rectification engine that removes background and instrumental radiation from the images.
- An image coaddition engine that calculates weighted averages of pixel fluxes in the final mosaic.

The Montage components can be called separately or in tandem. Figure 2 shows an overview of the process flow in Montage. Essentially all requests will call the reprojection and co-addition engines. These engines will process all the input images, generate custom images according to the user's specification of coordinates, sampling and projection, and co-add the fluxes in the output images. Calls to the background modeling and rectification engines are made if requested by the user. Background modeling and rectification involves fitting the differences between overlapping images on a local (for small mosaics) or global scale and determining the parameters for smooth surfaces to be subtracted from each image to bring them to a common scale. These parameters can either be determined on the fly or done once and saved in a database for any future mosaics done with the same images. The advantage of local fitting is that it allows variations in the fitting algorithms to deal with special cases and, for small regions, will probably be more sensitive to local variations than a global fit. The advantage of global fitting is that it provides a uniform view of the sky and a tested "best fit" that can be certified as such by the project. [we might want to mention that we have only implemented local fitting in Montage 2.1.]

Our design allows us to use both approaches. We will derive and store in a relational DBMS at least one set of background fit parameters for the whole sky, based on algorithms supplied by the providers of the image collections, but allowing the user the option to invoke custom background processing if they think it will provide a better mosaic for a local region. SDSC is committed to providing a DBMS for NVO-related processing, but the choice of engine is TBD.

Figure 1: Design Components of Montage – High Level Design. Montage performs three principal functions in generating an image mosaic, and the components of each are illustrated here. They are image reprojection, background modeling, background rectification, and image coaddition. labeling on the plot still looks funny to me



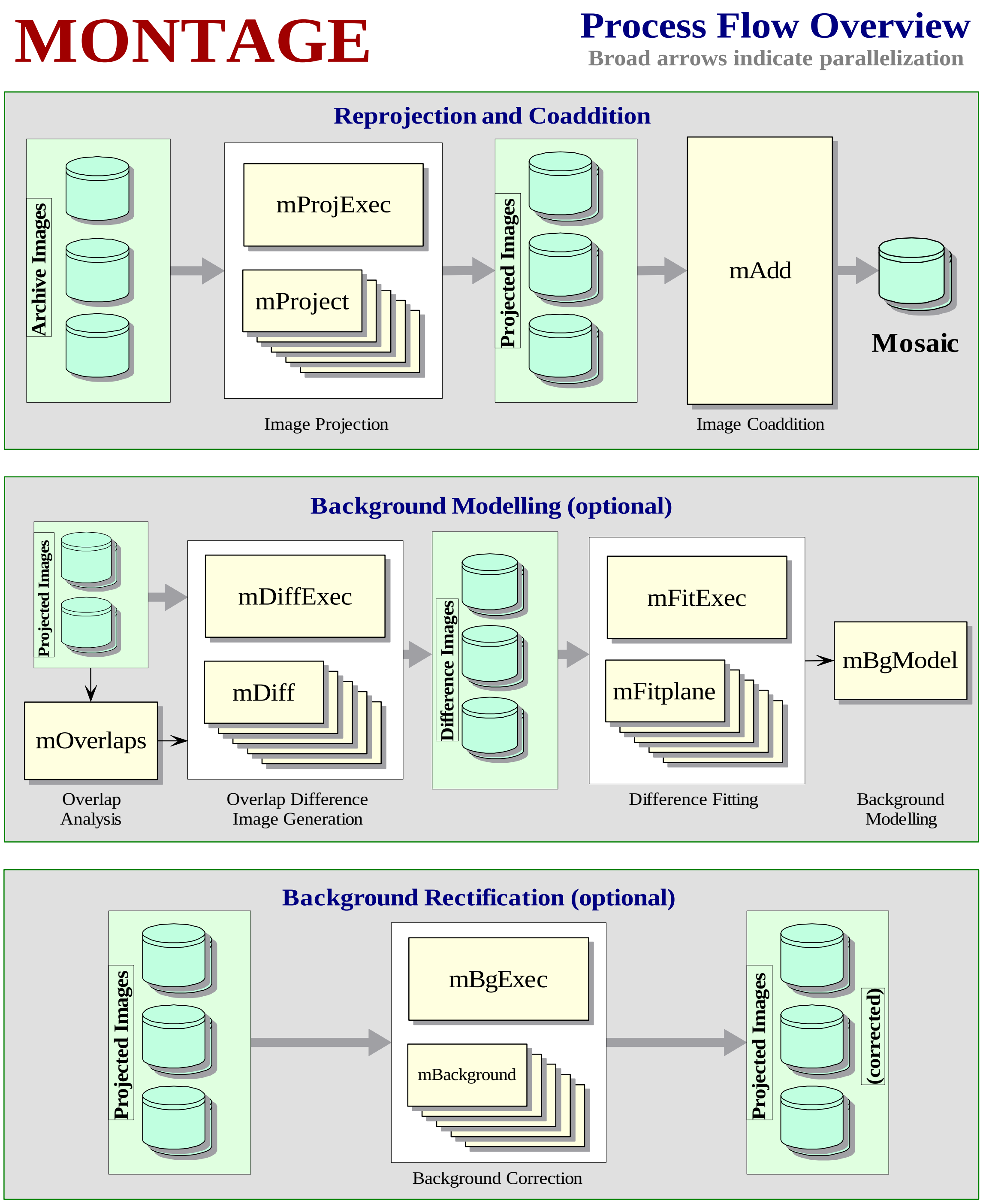


Figure 2: Design Components of Montage - Process Flow Overview.

3.2 Computational Algorithms in Montage

This section describes the innovations in computational algorithms developed to support the design of Montage.

3.2.1 Image Reprojections and Pixel Overlap

Image reprojection involves the redistribution of information from a set of input pixels to a set of output pixels. For astronomical data, the input pixels represent the total energy received from an area on the sky, and it is critical to preserve this information when redistributed into output pixels. In astronomy, it is also important to preserve the positional (astrometric) accuracy of the energy distribution, so common techniques such as adding all the energy from an input pixel to the "nearest" output pixel are inadequate.

Instead, we must redistribute input pixel energy to the output based on the exact overlap of these pixels, possibly even with a weighting function across the pixels based on the point spread function for the original instrument. *The goal is to create an output image which is as close as possible to that which would have been created if the sky had been observed using an instrument with the output image's pixel pattern*. We are also committed to building a system which handles all astronomical projections and coordinate systems equally well.

3.2.1.1 A General Reprojection Algorithm

The most common approach to determining pixel overlap is to project the input pixel into the output pixel Cartesian space. This works well for some projection transformations but is difficult for others. One example of a difficult transformation is the Aitoff projection, commonly used in astronomy, where locations at the edge of an image correspond to undefined locations in pixel space. For Montage, we have decided instead to project both input and output pixels onto the celestial sphere. Since all such "forward" projections are well defined, the rest of the problem reduces to calculating the area of overlap of two convex polygons on a sphere (with no further consideration of the projections involved). The issue of handling reprojections therefore becomes a problem of classical spherical trigonometry.

General algorithms exist for determining the overlap of polygons in Cartesian space [5]. We have modified this approach for use in spherical coordinates to determine the intersection polygon on the sphere (a convex hull) and applied Girard's Theorem [6], which gives the area of a spherical triangle based on the interior angles, to calculate the polygon's area.

The result is that for any two overlapping pixels, we can determine the area of the sky from the input pixel that contributes energy to the output pixel. This provides not only a mechanism for accurately distributing input energy to output pixels but, as we shall see, a natural weighting mechanism when combining overlapping images.

Our approach implicitly assumes that the polygon defining a single pixel can be approximated by the set of great circle segments connecting the pixel's corners. Since even the largest pixels in any realistic image are on the order of a degree across, the non-linearities along a pixel edge are insignificant. Furthermore, the only effect this would have would be to the astrometric accuracy of the energy location information and would amount to a very small fraction (typically less that 0.01) of the size of a pixel. Total energy is still conserved.

The Montage processing scheme is a natural fit with the "drizzle" algorithm developed by STScI [7]. Simply, that algorithm shrinks each input pixel's size linearly toward its center (a square pixel one arcsecond on a side becomes a square pixel a fraction of an arcsecond in size with the same center) before it is reprojected and its flux redistributed to the output pixels. In Montage, this means simply computing different corners in the input linear pixel space; the flux redistribution and appropriate area-based normalization are handled naturally by the basic Montage algorithms. There is a slight impact on processing speed since all four pixel corners must be calculated for all pixels (in the non-drizzle case there is some saving because pixels share corners). For this reason, "drizzle" has been implemented as an option in Montage from its inception.

3.2.1.2 A Fast Reprojection Algorithm

The general reprojection algorithm described above transforms pixel coordinates in the input image to coordinates on the sky, and then transforms that location to output image pixel space. Under certain circumstances, this can be replaced by a much faster algorithm which uses a set of linear equations (though not a linear transform) to transform directly from input pixel coordinates to output pixel coordinates. This alternate approach is limited to cases where both the input and output projections are “tangent plane” (gnomonic, orthographic, *etc.*), but since these projections are by far the most common, it is appropriate to treat them as a special case.

This “plane-to-plane” approach is based on a library developed at the *Spitzer Science Center* [8]. When both images are tangent plane, the geometry of the system can be viewed as in Figure 3, where a pair of gnomonic projection planes intersect the coordinate sphere. A single line connects the center of the sphere, the projected point on the first plane and the projected point on the second plane. This geometric relationship results in transformation equations between the two planar coordinate systems that require no trigonometry or extended polynomial terms. As a consequence, the transform is a factor of thirty or more faster than using the normal spherical projection formulae.

A bonus to the plane-to-plane approach is that the computation of pixel overlap is much easier, involving only clipping constraints of the projected input pixel polygon in the output pixel space.

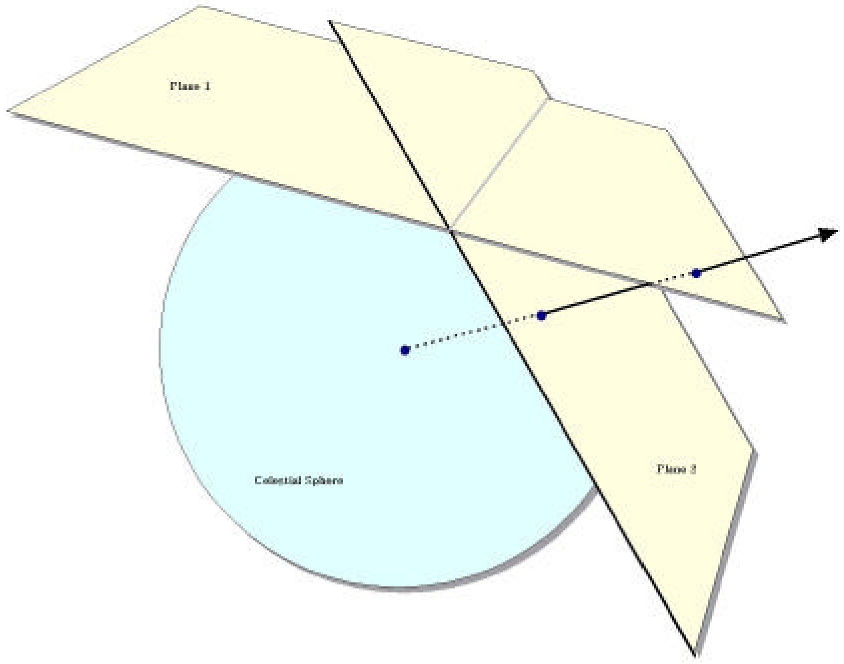


Figure 3: The Principle of Plane-to-plane Reprojection

This approach excludes many commonly-used projections such as “Cartesian” and “zenithal equidistant,” and is essentially limited to small areas of few square degrees. Processing of all-sky images, as is almost always the case with projections such as Aitoff, generally requires the slower plane-to-sky-to-plane approach.

There is, however, a technique that can be used for images of high resolution and relatively small extent (up to a few degrees on the sky). Rather than use the given image projection, we can often approximate it with a very high degree of accuracy with a “distorted” Gnomonic projection. In this case, the pixel locations are “distorted” by small distances relative to the plane used in the image projection formulae. A distorted space is one in which the pixel locations are slightly offset from the locations on the plane used by the projection formulae, as happens when detectors are slightly misshapen. This distortion is modeled by pixel-space polynomial correction terms which are stored as parameters in the image FITS header.

While this approach was developed to deal with physical distortions caused by telescope and instrumental effects, it is applicable to Montage in augmenting the plane-to-plane reprojection. Over a small, well-behaved region, most projections can be approximated by a Gnomonic (TAN) projection with small distortions. For instance, in terms of how pixel coordinates map to sky coordinates, a two-degree “Cartesian” (CAR) projection is identical to a TAN projection with a fourth-order distortion term to within about a percent of a pixel width. Figure 4 shows this in exaggerated form for clarity, with the arrows showing the sense of the distortion.

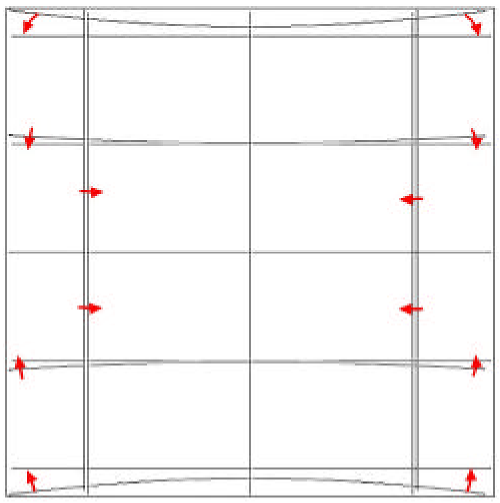


Figure 4: Representation of a WCS projection as a distorted Gnomonic (TAN) projection, exaggerated for clarity. The arrows indicate the sense of the distortions.

In the above example, the curved coordinate grid is an undistorted TAN, and the rectangular grid is both a CAR and the equivalent distorted TAN. This polynomial "correction" plus the plane-to-plane transform is still much faster than the normal reprojection. While this still does not cover all the possible transformations, it does include all those used for very large data collections.

3.2.2 Background Modeling and Rectification

If several images are to be combined into a mosaic, they must all be projected onto a common coordinate system (see above) and then any discrepancies in brightness or background must be removed. Our assumption is that the input images are all calibrated to an absolute energy scale (i.e. brightnesses are absolute and should not be modified) and that any discrepancies between the images are due to variations in their background levels that are terrestrial or instrumental in origin.

The Montage background matching algorithm is based on the assumption that terrestrial and instrumental backgrounds can be described by simple functions or surfaces (e.g. slopes and offsets). Stated more generally, we assume that the "non-sky" background has very little energy in any but the lowest spatial frequencies. If this not the case, it is unlikely that any generalized background matching algorithm will be able distinguish between "sky" and rapidly varying "background"; background removal will then require an approach that depends on detailed knowledge of an individual data set.

Given a set of overlapping images, characterization of the overlap differences is key to determining how each image should be adjusted before combining them. We take the approach of considering each image individually with respect to its neighbors. Specifically, we determine the areas of overlap between each image and its neighbors, and use the complete set of overlap pixels in a least-squares fit to determine how each image should be adjusted (e.g. what gradient and offset should be added) to bring it "best" in line with its neighbors.

In practice, we only adjust the image by half this amount, since all the neighbors are also being analyzed and adjusted and we want to avoid ringing in the algorithm. After doing this for all the images, we iterate (currently for a fixed number of times, though we may later introduce convergence criteria). The final effect is to have subtracted a low-frequency (currently a gradient/offset) background from each image in such a way that the cumulative image-to-image differences are minimized. To speed the computation (and minimize memory usage), we approximate the gradient and offset values by a planar surface fit to the overlap area difference images rather than perform a least squares fit.

3.2.3 Coadditions and Weighting of Output Pixel Fluxes

In the reprojection algorithm (described in the pixel overlap discussion above), each input pixel's energy contribution to an output pixel is added to that pixel, weighted by the sky area of the overlap. In addition, a cumulative sum of these sky area contributions is kept for the output pixels (essentially and physically an "area" image). When combining multiple overlapping images, these area images provide a natural weighting function; the output pixel value is simply the area-weighted average of the images being combined

Such images are in practice very flat (with only slight slopes due to the image projection) since the cumulative effect is that each output pixel is covered by the same amount of input area, regardless of the pattern of coverage. The only real variation occurs at the edges of the area covered, since there an output pixel may be fractionally covered by input pixels.

The limitations of available memory have been simply overcome in co-addition by reading the reprojected images one line at a time from files that reside on disk. Assuming that a single row of the output file does not fill the memory, the only limitation on file size is that imposed by the file system. Images of up to 6 GB have thus far been built with the new software. For each output line, mAdd determines which input files will be contributing pixel values, and opens only those files. Each contributing pixel value is read from the flux and area coverage files, and the value of each of these pixels is stored in an array until all contributing pixels have been read for the corresponding output row. This array constitutes a “stack” of input pixel values; a corresponding stack of area coverage values is also preserved. The contents of the output row are then calculated one output pixel (i.e., one input stack) at a time, by averaging the flux values from the stack.

Different algorithms to perform this average can be trivially inserted at this point in the program. Version 2.x of Montage supports mean and median co-addition, with or without weighting by area. The mean algorithm (default) accumulates flux values contributing to each output pixel, and then scales them by the total area coverage for that pixel. The median algorithm ignores any pixels whose area coverage falls below a specific threshold, and then calculates the median flux value from the remainder of the stack.

If there are no area files, then the algorithm gives equal weight to all pixels. This is valuable for science data sets where the images are already projected into the same pixel space (e.g., MSX). An obvious extension of the algorithm is to support outlier rejection, and this is planned for a future release as an enhancement.

3.3 Parallelization

Although Montage 1.7.1 was intended to run on a single processor, the grid portal of Montage exploits the parallelization inherent in the processing flow. The basic Montage scenario is to reproject each of the input images to a common output specification (producing reprojected image/area files), analyze the background by determining the overlap pairs, calculate and fit the difference images, and model the background corrections, subtract this model from the reprojected images, and finally perform a weighted coaddition to generate the final mosaic.

The only place in this scenario where there is more than pairwise interaction between the images is the background modeling. All the other steps can easily be parallelized across multiple processing threads or even multiple machines.

The reprojection of each image takes by far the majority of the processing time; the reprojection can be performed independently for each image, even though each image uses the same output area definition. In fact, given the area weighting approach we use, the reprojection of an individual image could be parallelized across multiple threads through a simple tiling. Similarly, once the image/image overlaps are identified (a fast process) the difference image processing can be spread out in the same way.

While the final coaddition nominally feeds into a single output memory array, it too can be parallelized by tiling (the output space), though this is rarely necessary as the coaddition step is very fast.

This leaves only the background modeling as a linear process. While this cannot be subdivided along the lines of the other steps, it would be feasible to parallelize this in a more complex way (e.g. blocking the images into regional groups and using the Message Passing Interface to manage the intergroup cross-talk). However, this component is unlikely to ever be performance-critical, so this will probably not be necessary.

We have currently parallelized Montage using two distinct paths. First, we break down the processing flow into its components, and use Grid tools to run those components on a generic Grid. Second, we parallelize the executives to spawn their worker processors on a parallel machine, and additionally parallelize the co-addition component of Montage (mAdd), in both cases using MPI as the mechanism of parallelization. The next two sections of this document discuss these two paths

3.4 Montage Workflow Parallelized with Grid Tools

Montage is a portable toolkit that processes images serially or in parallel; it is designed so it can run on any parallel environment, including a Linux cluster or a "true" grid. Montage was designed with three needs in mind: (1) portability; (2) flexibility to the user; (3) adaptability to any processing environment. A computational Grid can take many different forms including a collection of supercomputers, cluster computers, or a pool of workstations connected by a network. This section describes the design of one implementation of a grid portal for Montage. This work is in fulfillment of Milestone I [9]. This milestone calls for a prototype architecture that accepts requests for a custom 2MASS image mosaic through a web portal, processes the requests on the TeraGrid (described below), and returns the image mosaic for visualization and analysis.

The TeraGrid web site [1] describes the TeraGrid as follows:

> *TeraGrid is a multi-year effort to build and deploy the world's largest, fastest, distributed infrastructure for open scientific research. When completed, the TeraGrid will include 20 teraflops of computing power distributed at five sites, facilities capable of managing and storing nearly 1 petabyte of data, high-resolution visualization environments, and toolkits for grid computing. These components will be tightly integrated and connected through a network that will operate at 40 gigabits per second—the fastest research network on the planet.*

The Montage TeraGrid portal is using these high performance resources to construct image mosaics. Users of the portal need only have a desktop computer running any standard web browser.

The Montage TeraGrid service accepts requests from two portals, one at JPL and one at ISI, underpinned by a common, distributed architecture. Figures 3 and 4 show this common architecture. First we describe the JPL portal, shown in Figure 3. This portal is a prototype of one we will ultimately deploy for astronomers, who will submit mosaic requests through a simple web form that inputs parameters describing the mosaic (location on the sky, size, coordinate system, projection, etc). A service at JPL/Caltech is contacted to generate an abstract workflow, which specifies the processing jobs to be executed, input, output, and intermediate files to be read or written during the processing, and dependencies between the jobs. A 2MASS image list service at IPAC/Caltech is contacted to generate a list of the 2MASS images required to fulfill the mosaic request. The abstract workflow is passed to a service at the Information Sciences Institute (ISI), University of Southern California, which runs software called Pegasus [10] to schedule the workflow on the TeraGrid. The resulting "concrete workflow" includes information about specific file locations on the grid and specific grid computers to be used for the processing. The workflow is then executed on the remote TeraGrid clusters using Condor DAGMan. DAGMan is a scheduler that submits jobs to Condor in an order specified by the concrete workflow. Condor queues the jobs for execution on the

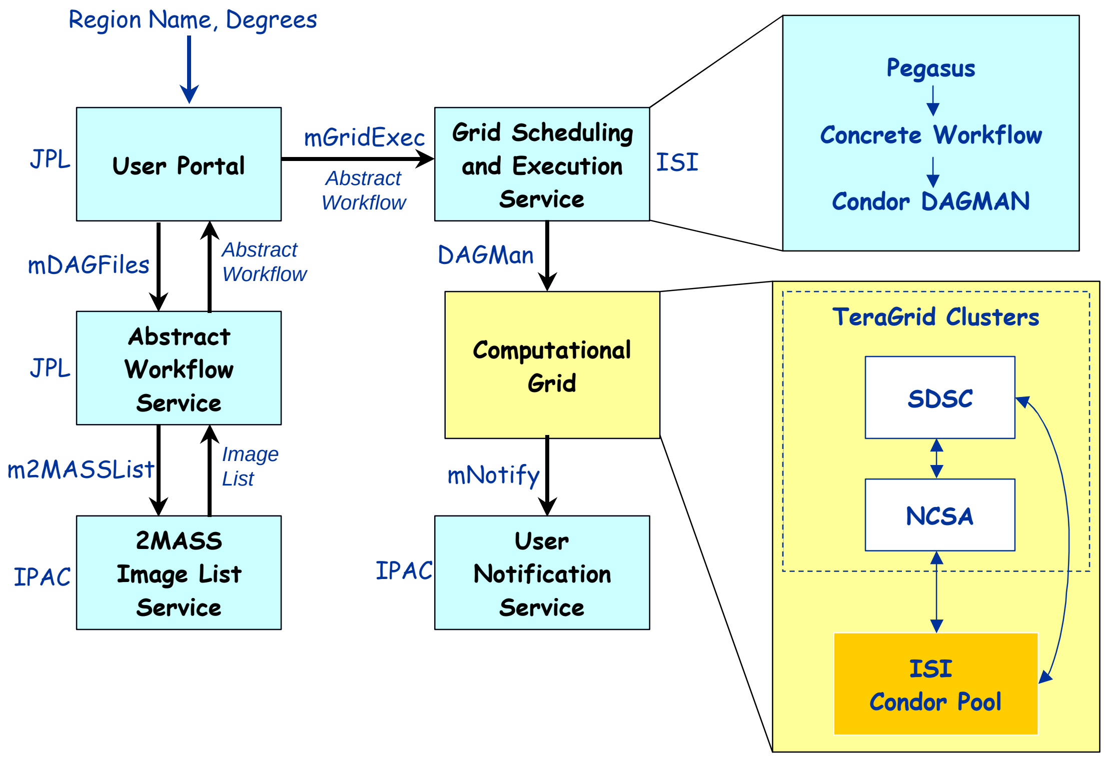


Figure 3. The distributed architecture of the Montage TeraGrid Portal.

TeraGrid. More information on Condor and DAGMan can be found on the Condor web site [11]. The last step in the mosaic processing is to contact a user notification service at IPAC/Caltech, which currently simply sends an email to the user with the URL of the Montage output.

The Montage grid portal is comprised of the following five main components, each having a client and server code:

1. User Portal
2. Abstract Workflow Service
3. 2MASS Image List Service
4. Grid Scheduling and Execution Service
5. User Notification Service

The second portal is the Pegasus portal at ISI simply takes the place of the User Portal and the User Notification Service. This portal provides complete diagnostic and status information on the processing, and returns all intermediate products. Astronomers simply wishing to receive a mosaic would find the JPL portal more useful.

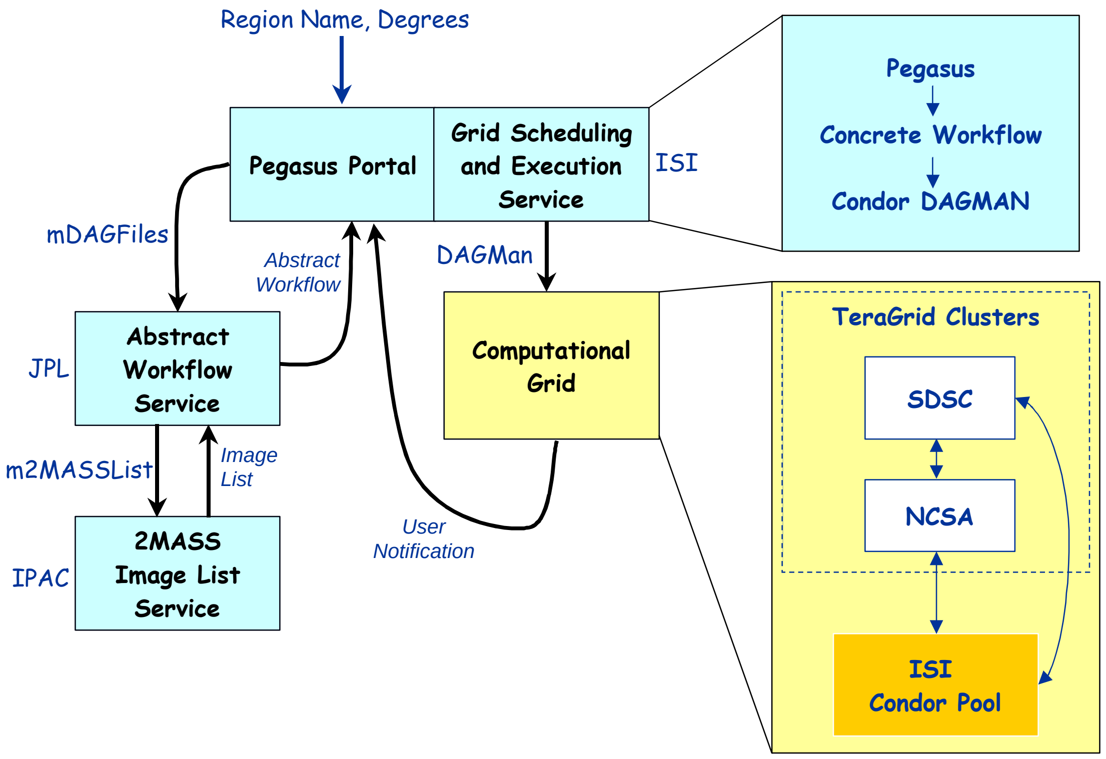


Figure 4. The preliminary design of the Montage TeraGrid Portal uses the Pegasus portal for user input and notification.

3.4.1 User Portal

The client code for the user portal is the ubiquitous web browser. Users fill out a simple web form with parameters that describe the mosaic to be constructed, including an object name or location, mosaic size, coordinate system, projection, and spatial sampling. Figure 5 shows a screen capture of the web form interface [12]. The data in the web form are submitted to the CGI program using the HTTP `POST` method.

The server side of the user portal includes two main codes, both implemented as Perl scripts: `montage-cgi` and `montaged`. The `montage-cgi` program is a CGI script that is run by the Apache web server after the user presses the "Submit" button on the web form. This CGI script checks for validity of the request parameters, and stores the validated requests to disk for later processing. The `montaged` program has no direct connection to the web server and runs continuously as a daemon to process incoming mosaic requests. The processing for a request is done in two main steps:

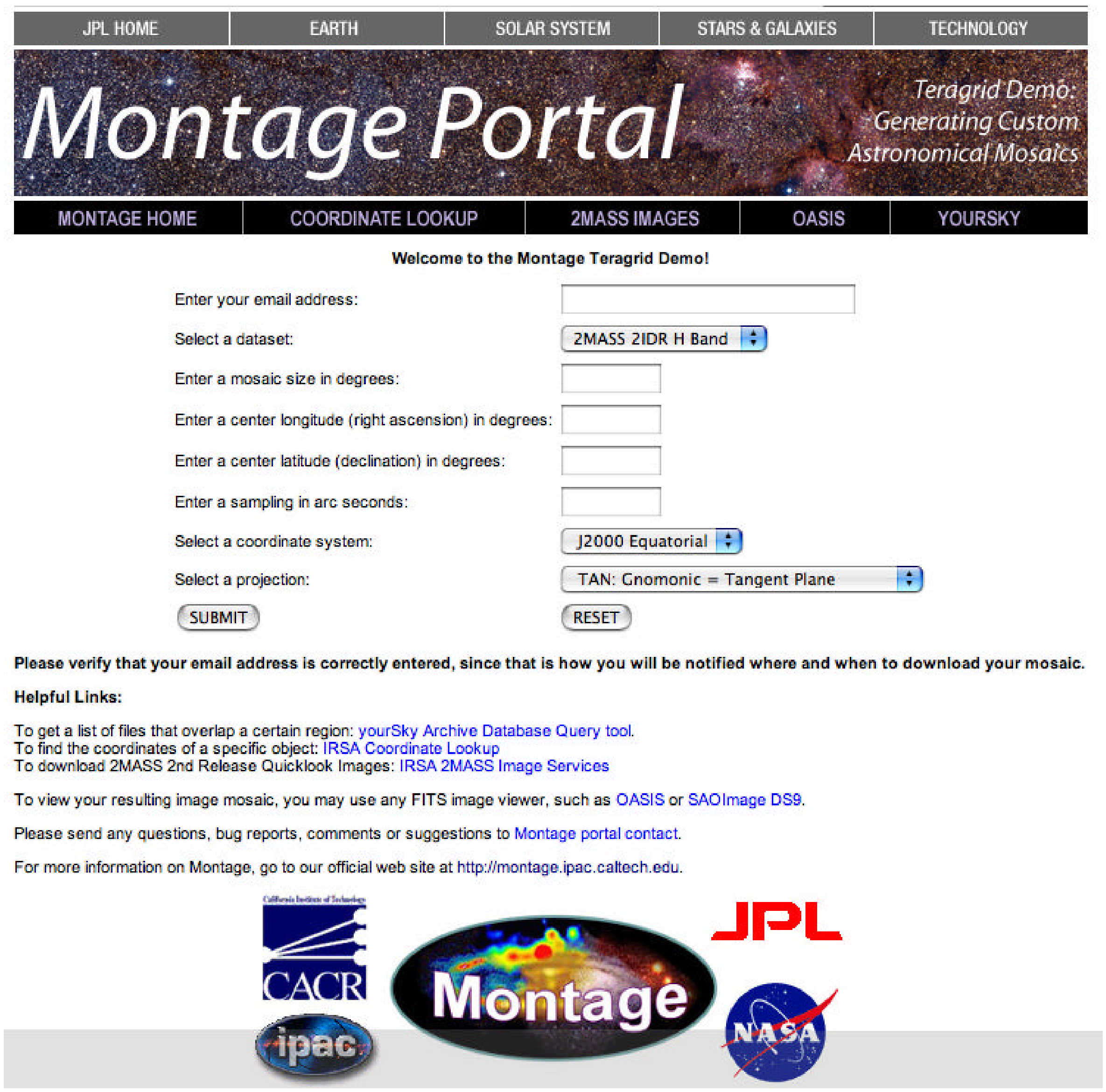
1. Call the abstract workflow service client code
2. Call the grid scheduling and execution service client code and pass to it the output from the abstract workflow service client code

3.4.2 Abstract Workflow Service

The client code for the abstract workflow service is mDAGFiles, a compiled ANSI C code, called with the following usage syntax:

```
        mDAGFiles object|location size suffix zipfile
```

The input arguments are an object name or location on the sky (which must be specified as a single argument string), a mosaic size, a filename suffix, and an output zip archive filename. These input parameters are sent to the abstract workflow server using the HTTP `POST` method.
The server is a CGI perl script called `nph-mdag-cgi`, which creates a number of files,



Welcome to the Montage Teragrid Demo!

Enter your email address:	
Select a dataset:	2MASS 2IDR H Band
Enter a mosaic size in degrees:	
Enter a center longitude (right ascension) in degrees:	
Enter a center latitude (declination) in degrees:	
Enter a sampling in arc seconds:	
Select a coordinate system:	J2000 Equatorial
Select a projection:	TAN: Gnomonic = Tangent Plane

Please verify that your email address is correctly entered, since that is how you will be notified where and when to download your mosaic.

Helpful Links:

To get a list of files that overlap a certain region: yourSky Archive Database Query tool.
To find the coordinates of a specific object: IRSA Coordinate Lookup
To download 2MASS 2nd Release Quicklook Images: IRSA 2MASS Image Services

To view your resulting image mosaic, you may use any FITS image viewer, such as OASIS or SAOImage DS9.

Please send any questions, bug reports, comments or suggestions to Montage portal contact.

For more information on Montage, go to our official web site at http://montage.ipac.caltech.edu.

Figure 5. Montage grid portal web form interface.

packs them into a zip archive file, and sends the zip file back to the calling client. The following files are included in the zip archive, with the specified filename if the `suffix` argument to `mDAGFiles` is specified as "`_SUF`" (Appendices A-G give sample files for a small mosaic of M51 built from just two images):

1. `adag_SUF.xml`: The abstract workflow as a directed acyclic graph (DAG) in XML; specifies the jobs and files to be encountered during the mosaic processing, and the dependencies between the jobs (see example in Appendix A1)
2. `images_SUF.xml`: A table containing filenames and other attributes associated with the images needed to construct the mosaic (see example in Appendix A2)
3. `pimages_SUF.xml`: A table containing the filenames to be used for the images that have been reprojected by `mProject` (see example in Appendix A3)
4. `cimages_SUF.xml`: A table containing the filenames to be used for the images that have been background corrected by `mBackground` (see example in Appendix A4)
5. `fit_list_SUF.tbl`: A table containing the filenames output by `mFitplane` for the difference image fit plane parameters (see example in Appendix A5)
6. `template_SUF.hdr`: Montage template header file describing output mosaic (see example in Appendix A6)
7. `log_SUF.txt`: A log file with time stamps for each part of the processing (see example in Appendix A7).

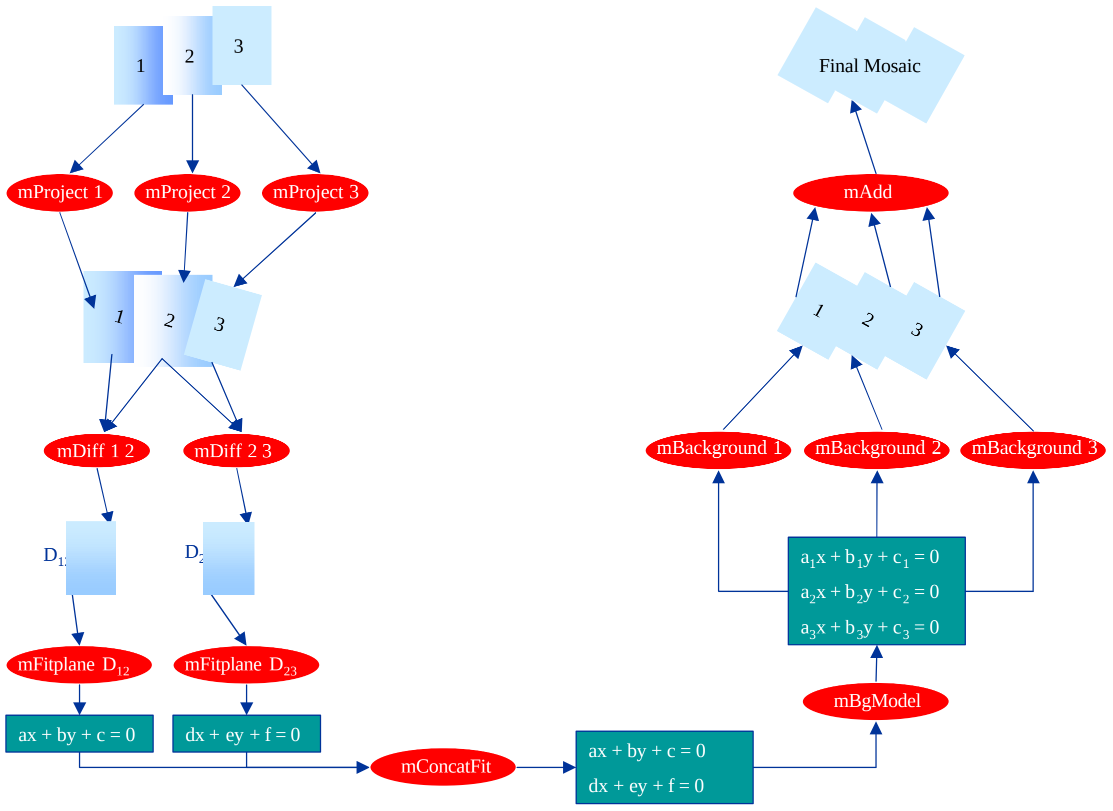


Figure 6. Example abstract workflow.

All of these files are required by the Grid Scheduling and Execution Service, described below). The abstract workflow in `adag_SUF.xml` specifies the filenames to be encountered and jobs to be run during the mosaic processing, and dependencies between jobs, which dictates which jobs can be run in parallel. A pictorial representation of an abstract workflow for a mosaic with three input images is shown in Figure 6. The `images_SUF.xml` file is created by querying the 2MASS Image List Service, described below. The `pimages_SUF.xml` and `cimages_SUF.xml` are required as input to the `mBgModel` and `mAdd` programs, respectively, and are created by simple text manipulation of the `images_SUF.xml` file. The `fit_list_SUF.tbl` file is required as input to `mConcatFit` for merging of the individual fit plane files into one file required by `mBgModel`.

3.4.3 2MASS Image List Service

The 2MASS Image List Service is accessed via a client code called `m2MASSList`, which is called with the following user syntax:

```
m2MASSList object|location size outfile
```

The input arguments are an object name or location on the sky (which must be specified as a single argument string), a mosaic size in degrees, and an output file name. The 2MASS images that intersect the specified location on the sky are returned in `outfile` in a table, with columns that include the filenames and other attributes associated with the images.

3.4.4 Grid Scheduling and Execution Service

The Grid Scheduling and Execution Service is triggered using a simple client code called `mGridExec,` with the following calling syntax:

```
mGridExec zipfile
```

The input argument, `zipfile`, is the zip archive generated with the Abstract Workflow Service, described above.

On the server side, the user is authenticated on the Grid, and the work is first scheduled on the Grid using a program called Pegasus, and then executed using Condor DAGMan.

Pegasus is a workflow management system designed to map abstract workflows onto the Grid resources to produce concrete (executable) workflows. Pegasus consults various Grid information services, such as the Globus Monitoring and Discovery Service (MDS), the Globus Replica Location Service (RLS), the Metadata Catalog Service (MCS), and the Transformation Catalog to determine the available resources and data. Pegasus reduces the abstract workflow based on the available data. For example, if intermediate workflow products are registered in the RLS, Pegasus does not perform the transformations necessary to produce these products. The executable workflow generated by Pegasus identifies the resources where the computation will take place, the data movement for staging data in and out of the computation, and registers the newly derived data products in the RLS and MCS.

Users are authenticated on the TeraGrid using their Grid security credentials. The user first needs to save their proxy credential in the MyProxy server. MyProxy is a credential repository for the Grid that allows a trusted server (like our Grid Scheduling and Execution Service) to access grid credentials on the user's behalf. This allows these credentials to be retrieved by the portal using the user's username and password. Once authentication is completed, Pegasus schedules the Montage workflow onto the TeraGrid or other clusters managed by PBS and Condor. The workflow is then submitted to Condor DAGMan for execution. Upon completion, the final mosaic is delivered to a user-specified location and the User Notification Service, described below, is contacted.

3.4.5 User Notification Service

The last step in the grid processing is to notify the user with the URL where the mosaic may be downloaded. This notification is done by a remote user notification service at Caltech IPAC so that a new notification mechanism can be used later without having to modify the Grid Scheduling and Execution Service. Currently the user notification is done with a simple email, but a later version will use the Request Object Management Environment (ROME), being developed separately for the National Virtual Observatory. ROME will extend our portal with more sophisticated job monitoring, query and notification capabilities.

The User Notification Service is accessed with a simply client code, `mNotify`, with the following usage syntax:

```
mNotify jobID userID resultsURL
```

3.4.6 Modifications from Montage_v1.7.1

The Montage Grid portal implementation required adding a number of extra codes that were not included in the first release and modifying the usage syntax for a number of the codes. The usage syntax modifications were required because Pegasus determines some dependencies by direct filename matching. Every Montage module had to output at least one file as output to signal completion. Some of the Montage modules had an optional status argument (`-s status`) added for this purpose. Also, because of the way dependencies are handled by Pegasus, if module 2 depends on module 1, module 1 had to output a file that was read by module 2. These changes are summarized here:

1. Added `mDAGFiles` client program to access the Abstract Workflow Service:

   ```
   mDAGFiles object|location size suffix zipfile
   ```

2. Added `m2MASSList` client program to access the 2MASS Image List Service:

   ```
   m2MASSList object|location size outfile
   ```

3. Added `mGridExec` client program to access the Grid Scheduling and Execution Service:

```
mGridExec zipfile
```

4. Added `mDAGTbls` program to take an image list from `m2MASSList` and the Montage template header file and produce two additional image lists, one for the projected image files and one for the background corrected image files:

```
mDAGTbls [-d][–s status] images.tbl hdr.template
            projected.tbl corrected.tbl
```

5. Added `mConcatFit` program to concatenate the output from multiple `mFitplane` jobs. This program (not shown for simplicity on the archiecture diagrams) was needed because on the Grid, each `mFitplane` job could be run on a different cluster pool. The `mBgModel` code requires the fit plane parameters in a single file so `mConcatFit` is used to concatenate the individual mFitplane output files into a single file for `mBgModel`. The calling syntax is as follows:

```
mConcatFit [-d][-s status] statfiles.tbl fits.tbl statdir
```

6. Changed `mFitplane` calling syntax to store the output in a file rather than sending it to stdout. The status file (specified with `–s outfile`) is used to store the output. The new calling syntax is:

```
mFitplane [-b border] [-d level] [-s outfile] in.fits
```

7. Changed `mBackground` calling syntax to read the correction plane parameters from a file rather than from the command line. The old calling syntax had the plane parameters on the command line as `A, B, C`. These are replaced by two parameters: the image table output by `mDAGTbls`, and a single file containing a table of the correction parameters for each image in the image list. It is assumed that both of these tables include a column with the file number and these file numbers have a correspondance across the two tables. The new calling syntax is:

```
mBackground [-t] [-d level] in.fits out.fits images.tbl
                       corrfile.tbl
```

3.5 Montage Workflow Parallelized with the Message Passing Interface (MPI)

The parallelization of Montage with MPI is fairly simple, as it just involves changing the four executives shown in Figure 2 (mProjExec, mDiffExec, mFitExec, and mBgExec) and mAdd.

The executives clearly could be considered the masters in a master-worker paradigm, and the MPI parallelization follows this strategy. All parallel processes run through all of the code of the executives, with the exception that in the main loop where the workers are run, each process only spawns on each Nth worker, where N is the number of MPI processes. For example, if mProjExecMPI is run as four processes, the process with rank 0 calls mProject for the first, fifth, ninth, etc. images that need to be processed. The process with rank 1 calls mProject for the second, sixth, tenth, etc. images, and so on. Each processor independently keeps track of the counts of its successes and failures. At the end of the processing, global sum reductions are used to calculate the total statistics, which are printed out by the process with rank 0. Finally, for the two executives that write out unified status files (mProjExec and mFitExec) another change is made so that each processor writes out its status to a local file, and when the processes are done, process 0 open the final status file, and reads from each of the temporary files in turn, in each case reading then writing all the lines to the final status file, then deleting the temporary status file.

mAdd has been parallelized with a slight different strategy. This component builds and writes the final mosaic to disk, one line at a time. The parallel version of mAdd operates similarly to the executives, where all processes do all of the work through the main loop. In that loop, each processor is assigned a range of lines to work on, and it only completes the main loop for lines in that range. In this case, for efficiency, the first process is responsible for the first 1/N lines, the second for the next 1/N, etc, rather than the round-robin distribution used for the executives.

The second set of flowcharts (section 4.3.2) shows these modifications.

This method of parallelization requires that all processes have access to a single file system, which is the case on individual TeraGrid clusters, as well as most parallel machines. This could not be used to run Montage on two independent clusters. However, this version of the parallel Montage is very similar to the sequential version. Where a user has a script designed to run a set of Montage processing, the only changes that need to be made to run the parallel Montage is to run the executives and mAdd through MPI. With the MPICH implementation of MPI, this involves changing the commands from:

```
   mModule flags&arguments
```

to:

```
   mpirun –np number-of-processes path/mModule flags&arguments
```

4. Detailed Design of Montage

4.1 Interface Specifications

Detailed information on running Montage, as well as a full API for each of its components, can be found on the project website [13][13].

4.2 Definitions of Montage File Formats

4.2.1 ASCII Table formats & the images.tbl file

The Montage modules read and generate column-delimited flat ASCII table files. One of these files, referred to in the calling syntax as “images.tbl”, is worth special discussion because it contains metadata describing the geometry on the sky of a set of image files (i.e. FITS header WCS keyword values). It is generated by mImgtbl and used by several other programs.

Montage uses a simple table reading library which looks for data in an ASCII file having a header with column names delimited by "|" characters and data records aligned in these columns.

Image metadata tables must contain the geometric information for each FITS image plus a counter and a pointer to the FITS file (In the sample file below, ns and nl are used in place of NAXIS1 and NAXIS2 to save space):

```
\datatype = fitshdr
| cntr |      ra     |     dec     |  ns |  nl | ctype1 | ctype2 |   crpix1 |   crpix2 |     crval1    |     crval2    |   cdelt1  |   cdelt2  |    crota2    | epoch
| fname                            |
| int  |    double   |    double   | int | int |  char  |  char  |   double |   double |     double    |     double    |   double  |   double  |    double    |
double| char
     0   265.1229433   -29.5911740   512  1024 RA---SIN DEC--SIN    256.50     512.50   265.1227836   -29.5910351 -2.7778e-
04  2.7778e-04     0.0011373  2000.00 ./2mass-atlas-980702s-j0830021.fits
     1   265.1229367   -29.3217296   512  1024 RA---SIN DEC--SIN    256.50     512.50   265.1227774   -29.3215907 -2.7778e-
04  2.7778e-04     0.0011343  2000.00 ./2mass-atlas-980702s-j0830033.fits
     2   265.1229302   -29.0522851   512  1024 RA---SIN DEC--SIN    256.50     512.50   265.1227713   -29.0521462 -2.7778e-
04  2.7778e-04     0.0011313  2000.00 ./2mass-atlas-980702s-j0830044.fits
```

The first line in the file is a parameter used by visualization software and can be treated as a comment in this context.

Key to the required columns in the images.tbl file

Users may specify additional columns or keywords/comments above the header. Dimensions 1 and 2 refer to axes 1 and 2 of a two-dimensional image.

Column	**Definition**	**FITS standard?**
cntr	A unique counter (row number)	N
ctype1, ctype2	The coordinate system (the first four characters) and WCS map projection (last three characters) for dimensions 1 and 2	Y
equinox	Precessional year associated with the coordinate system	Y
naxis1, naxis2	The size of the image in pixels for dimensions 1 and 2	Y
crval1, crval2	The coordinates of a reference location on the sky (often at the center of the image) for dimensions 1 and 2	Y
crpix1, crpix2	The pixel coordinates of the reference location (can be fractional and can be off the image) for dimensions 1 and 2	Y

cdelt1, cdelt2	The pixel scale (in degrees on the sky per pixel) at the reference location for dimensions 1 and 2	Y
crota2	The rotation angle from the "up" direction to the celestial pole	Y
hdu	The FITS extention number	N
fname	The path to the original FITS file	N

4.2.2 The Template.hdr file

Several Montage modules rely on a template for the output header, referred to in the calling syntax as "template.hdr," which is simply a text file containing one FITS header card per line. It looks like a FITS header, though with newlines after every card and with the trailing blanks on each line removed. It can be generated by hand or created by `mMakeHdr` to match an images.tbl file.

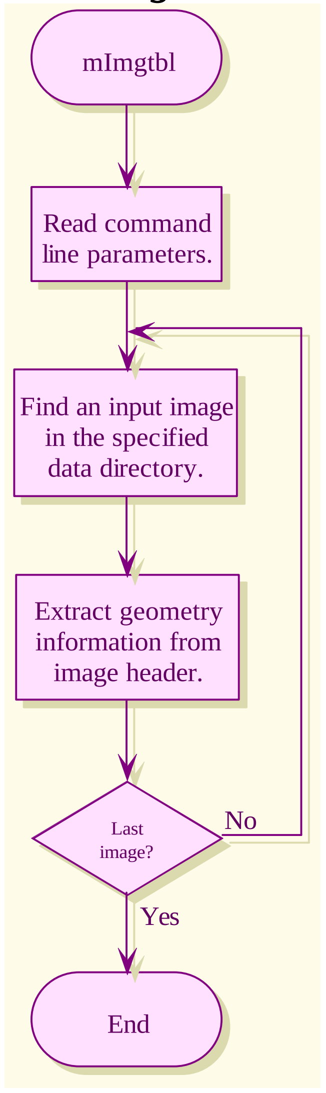
Any valid FITS header (with WCS information) is acceptable. The example below is for a Gnomonic-projection image, 3000x3000 pixels (1x1 degree) centered at 265.91334 -29.3577 Equatorial J2000.

```
SIMPLE  =                    T /
BITPIX  =                  -64 /
NAXIS   =                    2 /
NAXIS1  =                 3000 /
NAXIS2  =                 3000 /
CDELT1  =         -3.333333E-4 /
CDELT2  =          3.333333E-4 /
CRPIX1  =               1500.5 /
CRPIX2  =               1500.5 /
CTYPE1  = 'RA---TAN'           /
CTYPE2  = 'DEC--TAN'           /
CRVAL1  =            265.91334 /
CRVAL2  =            -29.35778 /
CROTA2  =                   0. /
```

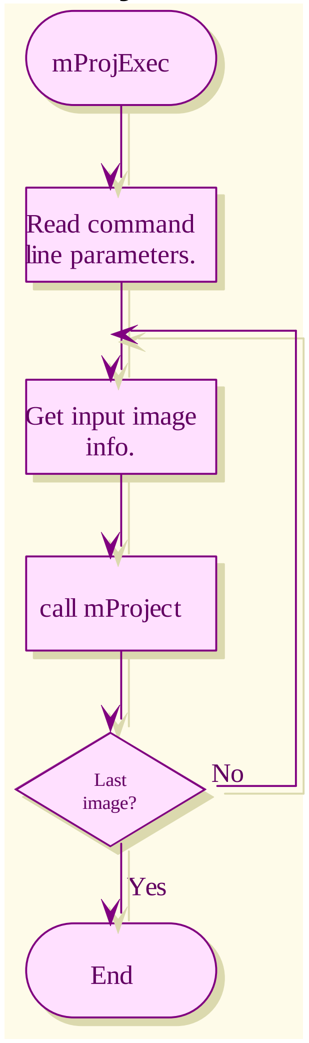
4.3 Design of Montage Modules: Flow Charts

4.3.1 Core Modules

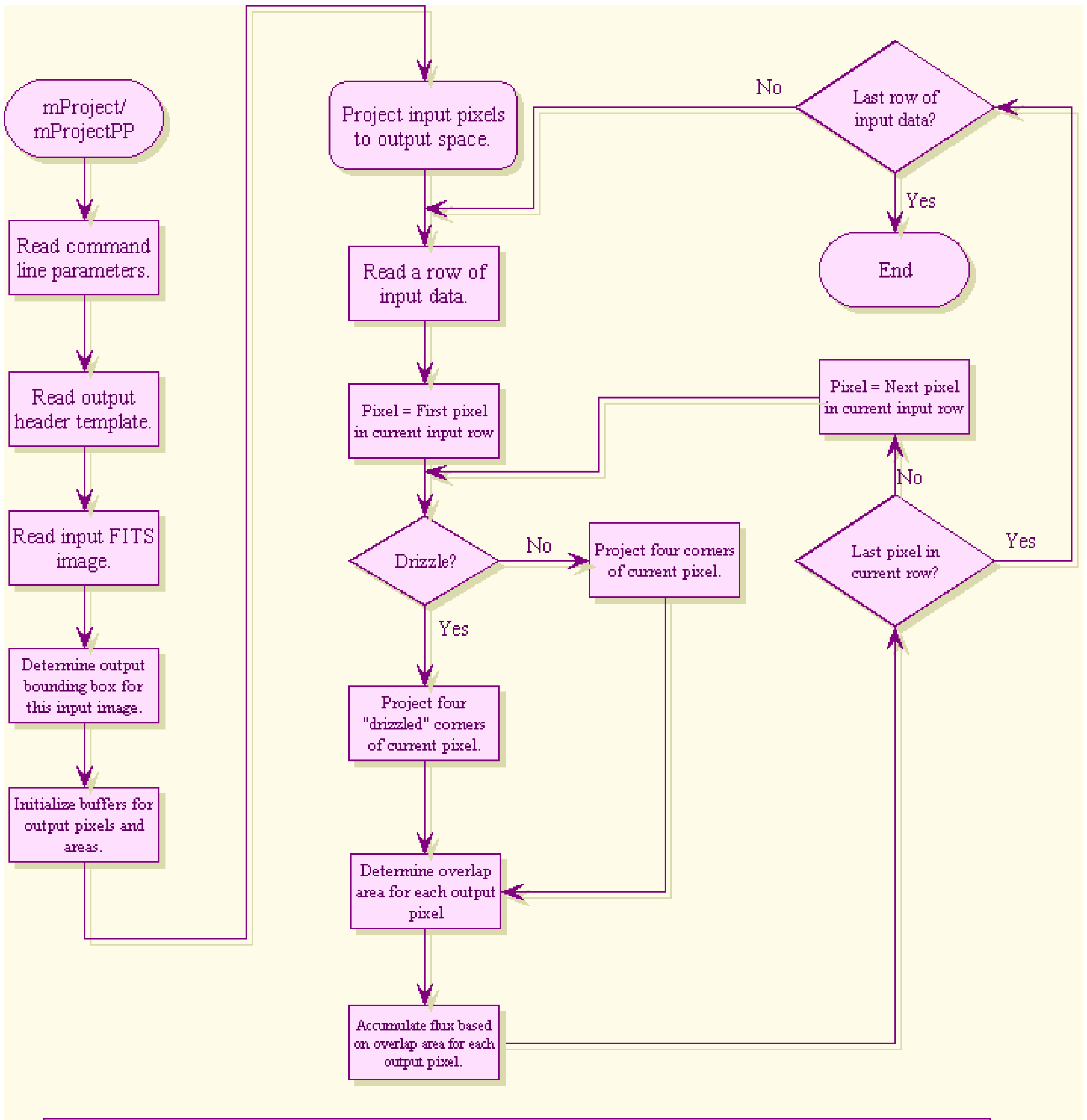
mImgtbl


mImgtbl
Read command line parameters.
Find an input image in the specified data directory.
Extract geometry information from image header.
Last image?
No
Yes
End

mProjExec

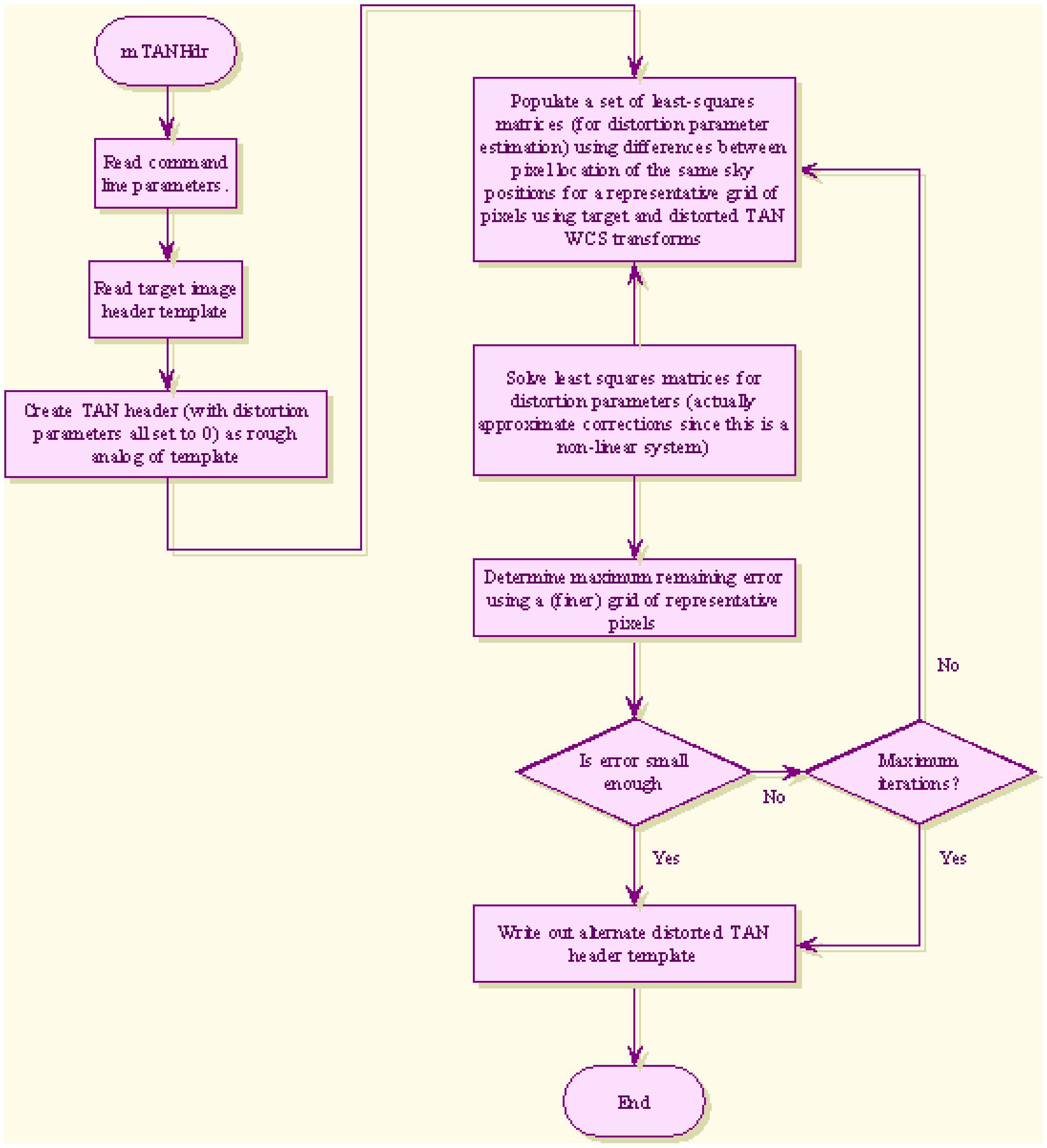


mProject/mProjectPP

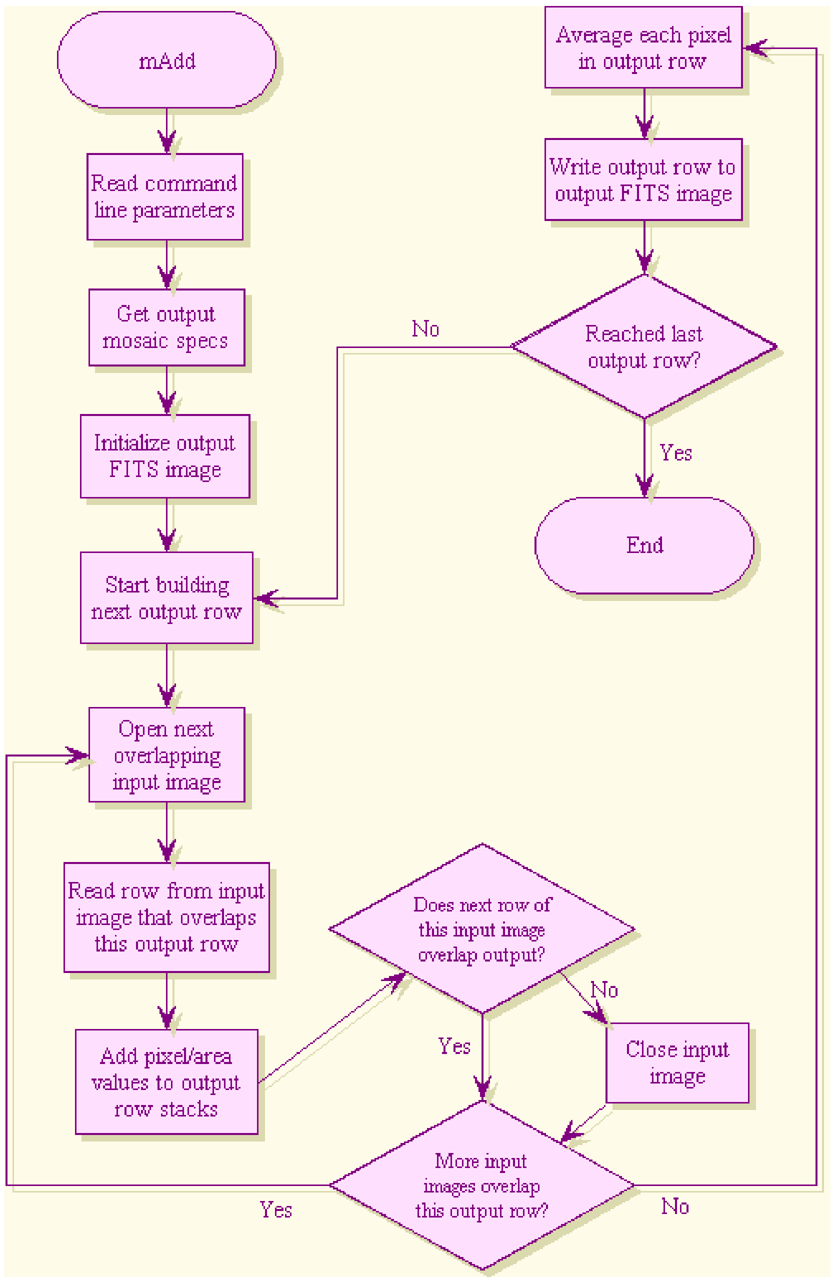

mProject/ mProjectPP
Read command line parameters.
Read output header template.
Read input FITS image.
Determine output bounding box for this input image.
Initialize buffers for output pixels and areas.
Project input pixels to output space.
Read a row of input data.
Pixel = First pixel in current input row
Drizzle?
No
Project four corners of current pixel.
Yes
Project four "drizzled" corners of current pixel.
Determine overlap area for each output pixel
Accumulate flux based on overlap area for each output pixel.
Last row of input data?
No
Yes
End
Pixel = Next pixel in current input row
No
Last pixel in current row?
Yes

mProject and mProjectPP have the same structure but use different reprojection libraries. mProject projects pixel corners from both input and output images onto the sky and computes overlap their using spherical trigonometry. It is therefore completely general (*i.e.* it can handle all projections) but is fairly slow. mProjectPP works in the output pixel coordinate space and uses a fast plane-to-plane reprojection library developed at the Spitzer Science Center. However, this library is limited to a few tangent plane projections (TAN, SIN, ZEA, STG, ARC). Other Montage modules extend this somewhat by allowing the use of alternate TAN headers with distortion parameters to be used to mimic certain other projections (*e.g.* CAR).

mTANHdr

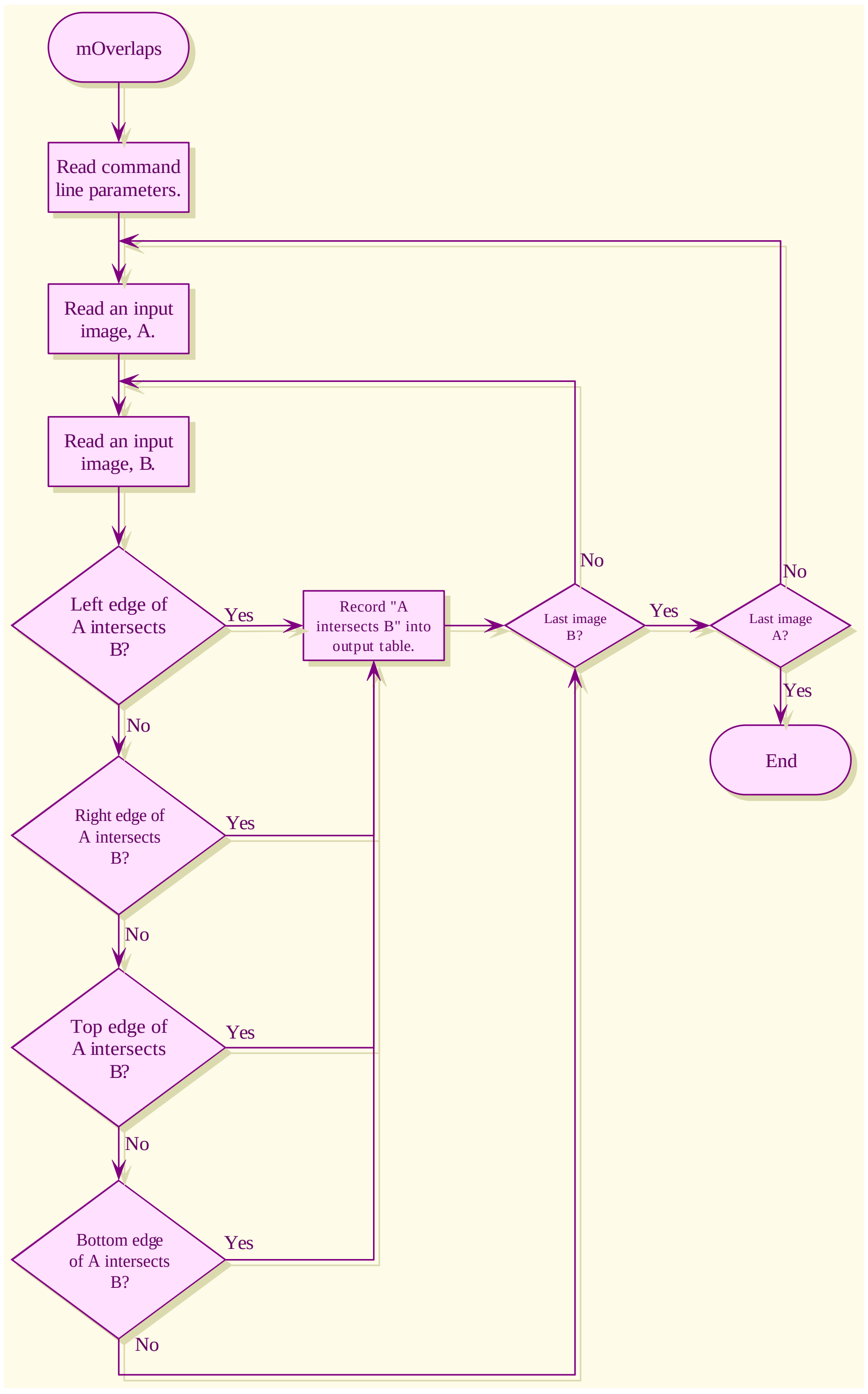

mTANHdr
Read command line parameters.
Read target image header template
Create TAN header (with distortion parameters all set to 0) as rough analog of template
Populate a set of least-squares matrices (for distortion parameter estimation) using differences between pixel location of the same sky positions for a representative grid of pixels using target and distorted TAN WCS transforms
Solve least squares matrices for distortion parameters (actually approximate corrections since this is a non-linear system)
Determine maximum remaining error using a (finer) grid of representative pixels
Is error small enough
No
Maximum iterations?
No
Yes
Yes
Write out alternate distorted TAN header template
End

mAdd



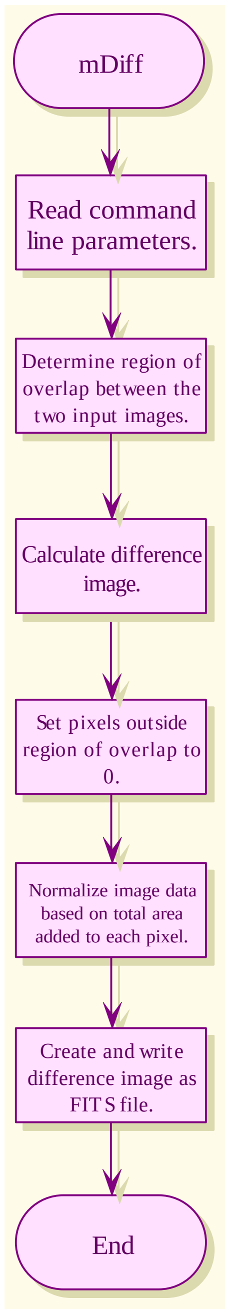
mAdd
Read command line parameters
Get output mosaic specs
Initialize output FITS image
Start building next output row
Open next overlapping input image
Read row from input image that overlaps this output row
Add pixel/area values to output row stacks
Does next row of this input image overlap output?
No
Yes
Close input image
More input images overlap this output row?
Yes
No
Average each pixel in output row
Write output row to output FITS image
Reached last output row?
No
Yes
End

mOverlaps

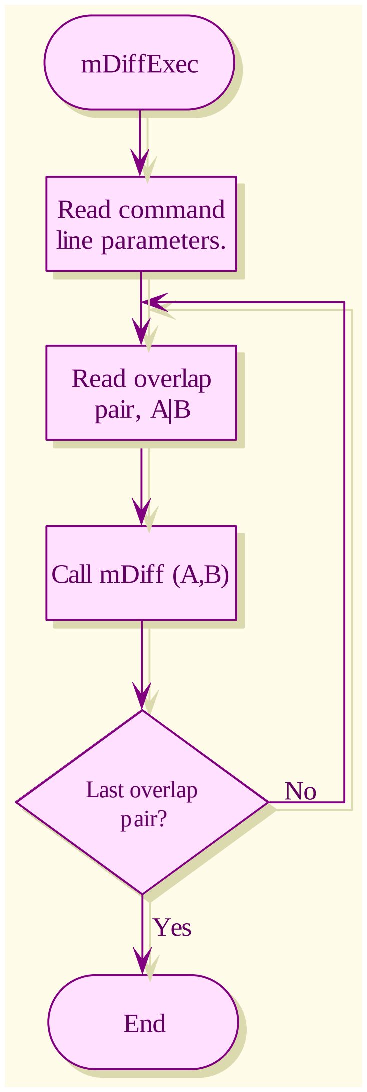


mOverlaps
Read command line parameters.
Read an input image, A.
Read an input image, B.
Left edge of A intersects B?
Yes
No
Record "A intersects B" into output table.
Last image B?
No
Yes
Last image A?
No
Yes
End
Right edge of A intersects B?
Yes
No
Top edge of A intersects B?
Yes
No
Bottom edge of A intersects B?
Yes
No

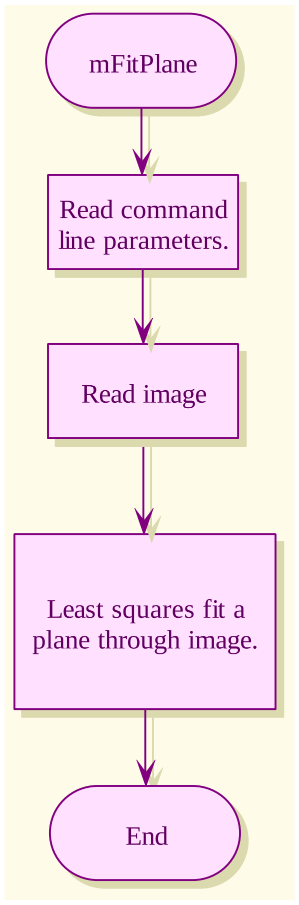
mDiff


mDiff
Read command line parameters.
Determine region of overlap between the two input images.
Calculate difference image.
Set pixels outside region of overlap to 0.
Normalize image data based on total area added to each pixel.
Create and write difference image as FITS file.
End

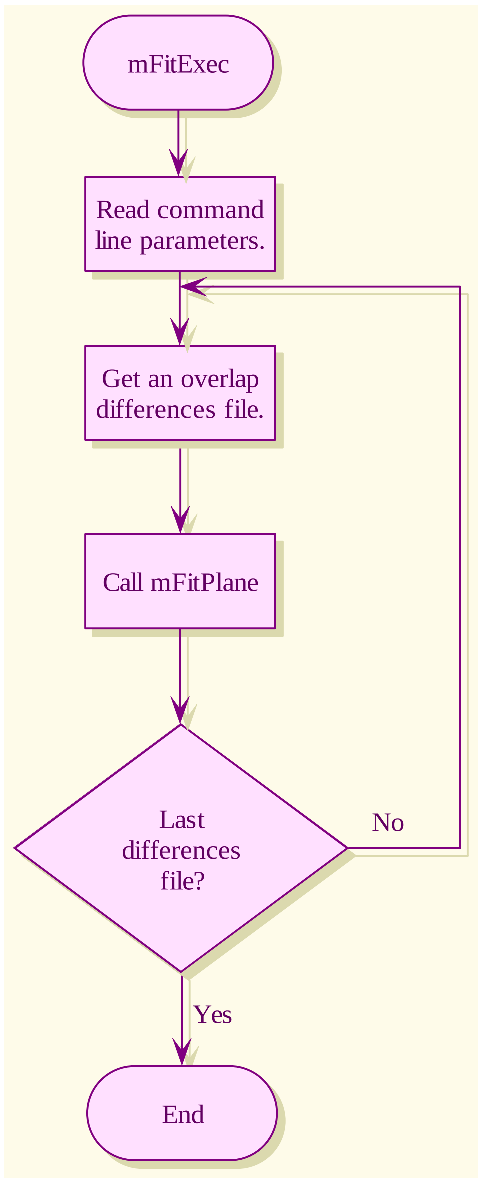
mDiffExec



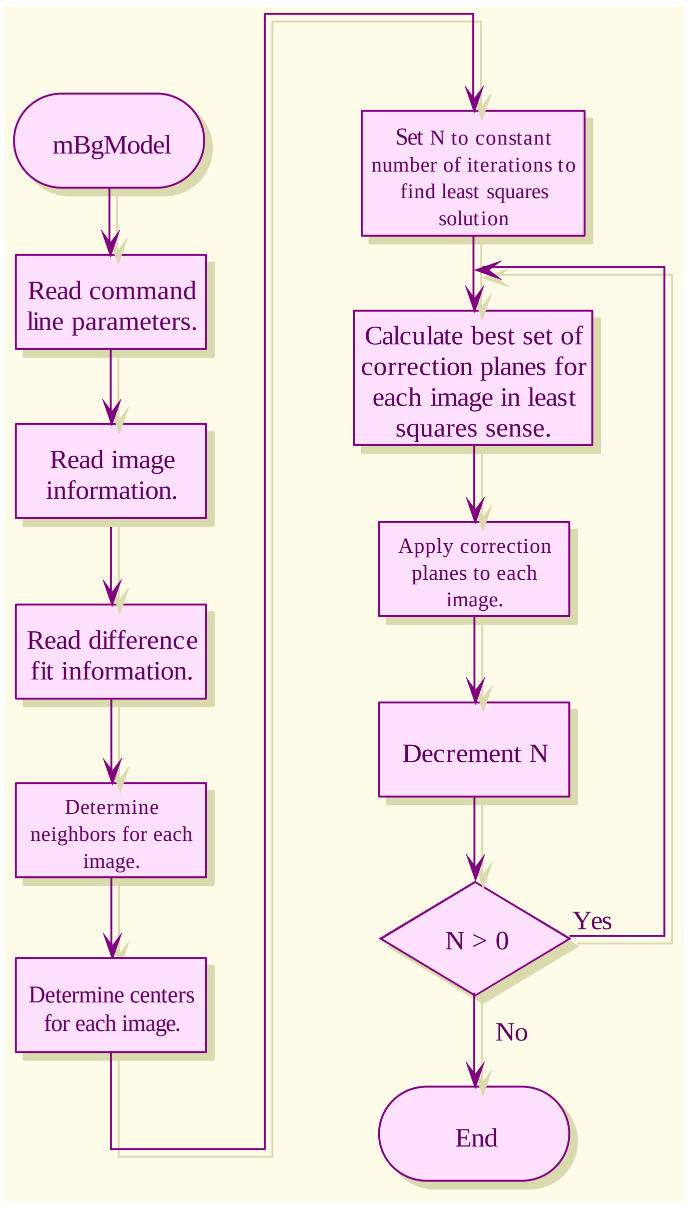
mFitPlane


mFitPlane
Read command line parameters.
Read image
Least squares fit a plane through image.
End

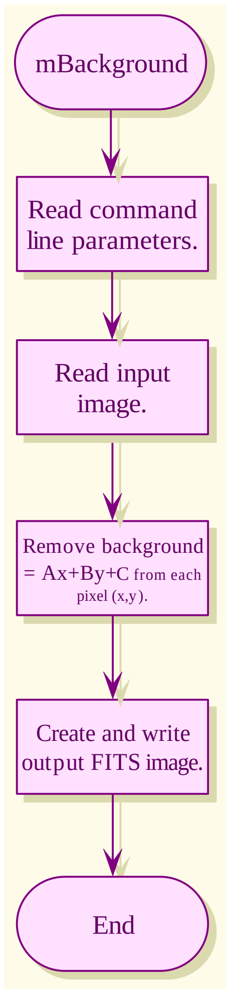
mFitExec


mFitExec
Read command line parameters.
Get an overlap differences file.
Call mFitPlane
Last differences file?
No
Yes
End

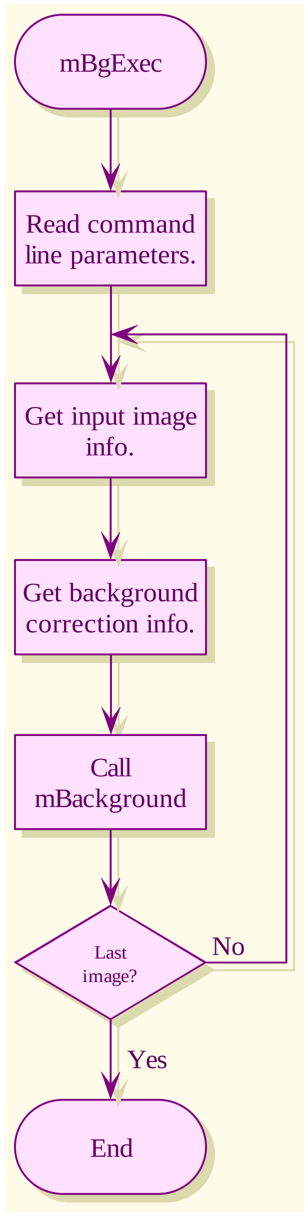
mBgModel


mBgModel
Read command line parameters.
Read image information.
Read difference fit information.
Determine neighbors for each image.
Determine centers for each image.
Set N to constant number of iterations to find least squares solution
Calculate best set of correction planes for each image in least squares sense.
Apply correction planes to each image.
Decrement N
N > 0
Yes
No
End

mBackground

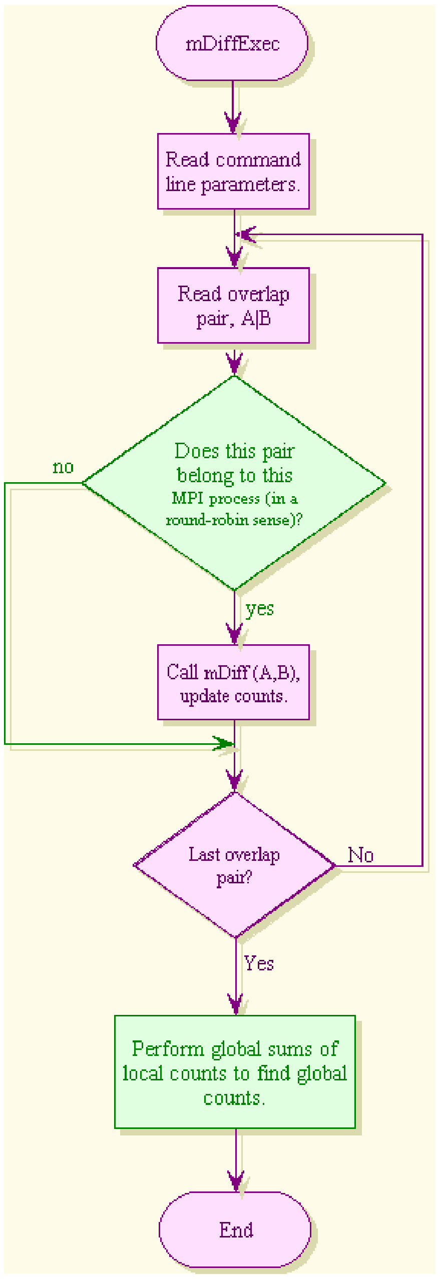

mBackground
Read command line parameters.
Read input image.
Remove background = Ax+By+C from each pixel (x,y).
Create and write output FITS image.
End

mBgExec

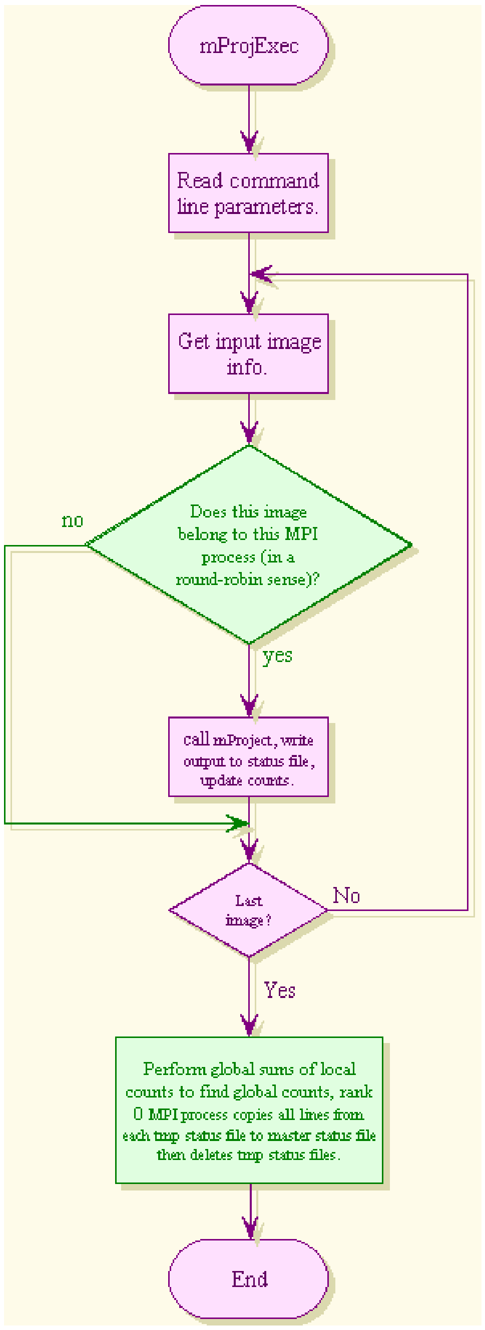

mBgExec
Read command line parameters.
Get input image info.
Get background correction info.
Call mBackground
Last image?
No
Yes
End

4.3.2 Flowcharts for Message Passing Interface Modules

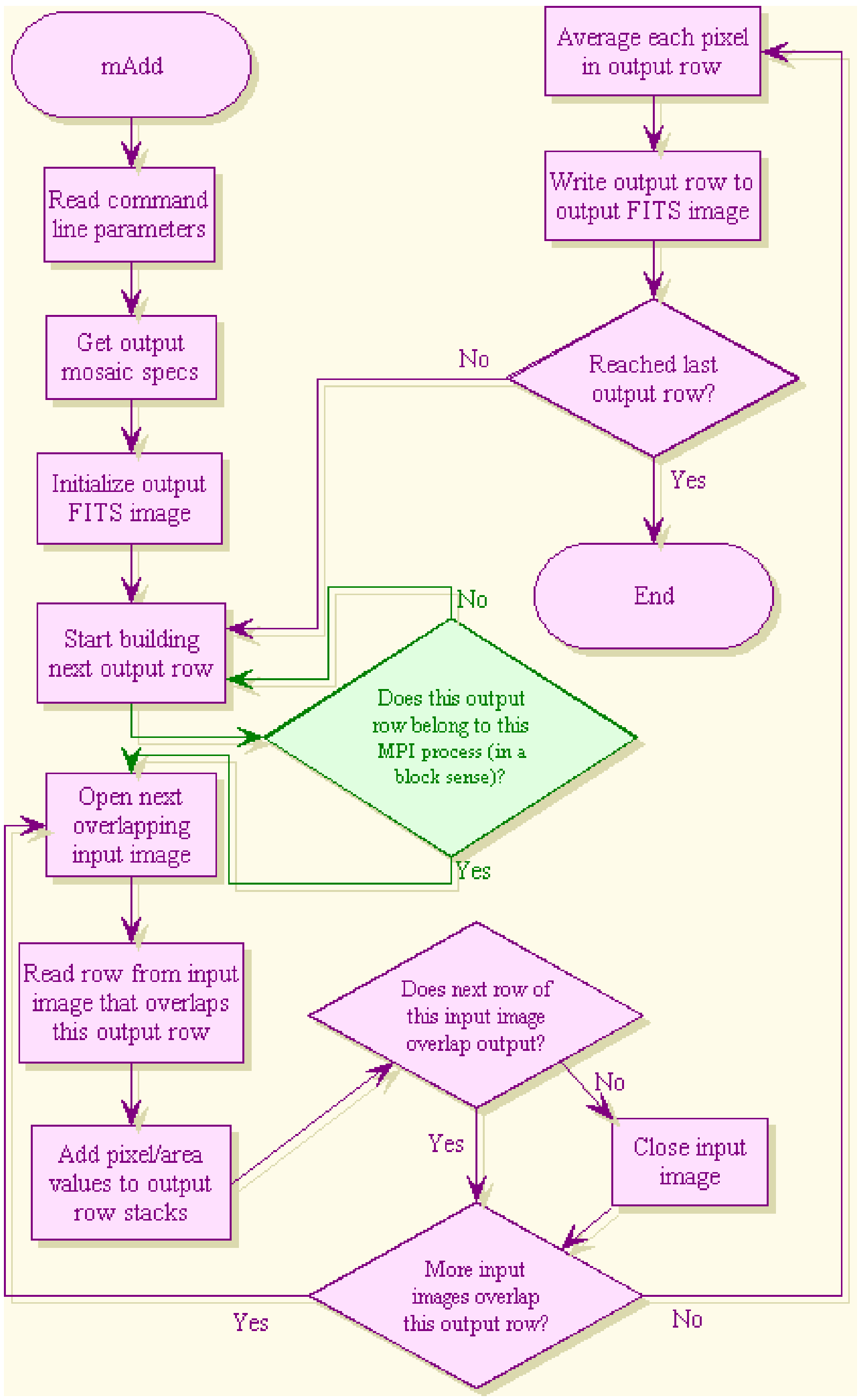
mDiffExec MPI


mDiffExec
Read command line parameters.
Read overlap pair, A|B
Does this pair belong to this MPI process (in a round-robin sense)?
no
yes
Call mDiff (A,B), update counts.
Last overlap pair?
No
Yes
Perform global sums of local counts to find global counts.
End

mProjExec MPI


mProjExec
Read command line parameters.
Get input image info.
Does this image belong to this MPI process (in a round-robin sense)?
no
yes
call mProject, write output to status file, update counts.
Last image?
No
Yes
Perform global sums of local counts to find global counts, rank 0 MPI process copies all lines from each tmp status file to master status file then deletes tmp status files.
End

mAdd MPI



mAdd
Read command line parameters
Get output mosaic specs
Initialize output FITS image
Start building next output row
Does this output row belong to this MPI process (in a block sense)?
No
Yes
Open next overlapping input image
Read row from input image that overlaps this output row
Add pixel/area values to output row stacks
Does next row of this input image overlap output?
Yes
No
Close input image
More input images overlap this output row?
Yes
No
Average each pixel in output row
Write output row to output FITS image
Reached last output row?
No
Yes
End

4.4 Error Handling Methodology

Montage employs the error handling methodology used by the Infrared Science Archive (IRSA), which uses a ‘svc’ library to fire up external processes as services, to send commands and receive structured responses, and to parse those responses to extract keyword = value pairs or the value of a particular keyword [14][14]. The on-line API includes a complete list of return codes for each module.

5. Montage Operating Under the NVO Architecture

Montage will run operationally on the Teragrid, a high performance computational grid provided by the NSF Partnership for Advanced Computational Infrastructure. The Teragrid provides aggregate computational power on the order of 10 Teraflops, aggregate disk cache on the order of 800 TB and archival storage capacity of 6 Petabytes. The details of how NVO compliant processes will be authenticated and fulfilled under the Teragrid are under development, but will follow the grid paradigm, where data needed for the request are obtained from the most convenient place, and computing is done on any available platform where the request can be authenticated.

A request to Montage must be satisfied transparently: users will only be aware that they are requesting an image mosaic according to their specification of position, size, projection *etc*. They will not be aware of where the request is performed, or if the image can be delivered or subset from a cached file. Figure 7 shows how a request to Montage will be handled when the architecture is fully deployed. The request is passed from the client to the Request Object Management Environment (ROME).

Broadly speaking, ROME is simply lightweight middleware, built with e-business Enterprise Java Bean (EJB) technology, which handles requests, responds to messages and manages pools of requests in a fault tolerant fashion [15]. A processing request to Montage will be accepted by ROME, which will register the request in the database and then send it for processing on the Teragrid. The job will be built on the Teragrid with standard Grid technologies such as the Globus, an Open Source toolkit that handles the construction and management of Grid processes, security *etc*.

Part of the request may already be satisfied in cached image mosaics. The cache will actually be part of a data management system that subsets files and constructs new mosaics from subsets, as needed. Montage will therefore search through a catalog of cached images and will satisfy such parts of the request as it can from the cached images. If cached files cannot fill the request, processing on the Teragrid will fill it.

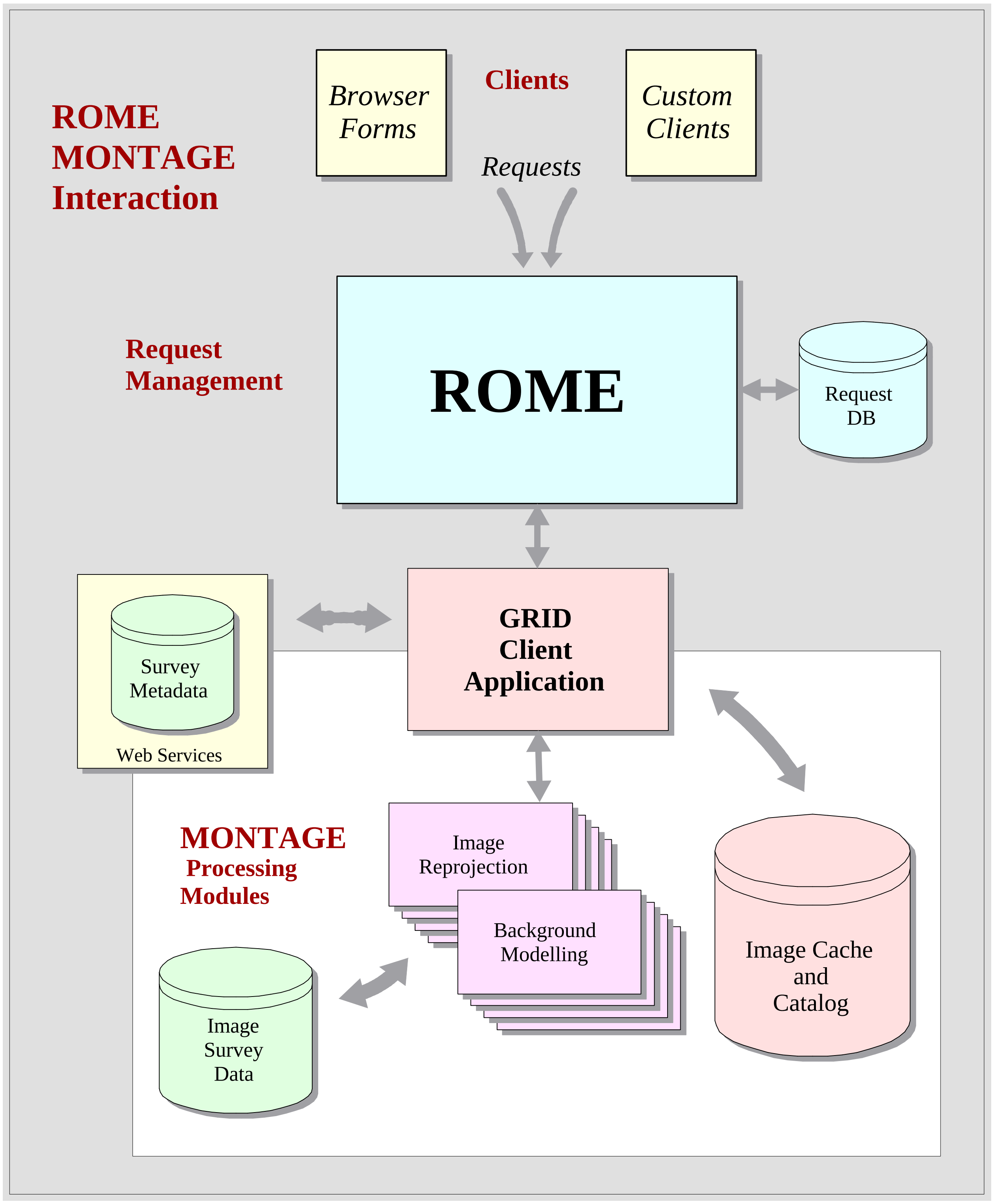


Figure 7: Montage Integrated in the NVO

An interpreter (part of grid resources such as Globus) accepts the XML request from ROME, and translates it into a suitable computational graph (directed acyclical graph, DAG) that specifies the computations that are needed and what data are needed. The DAG represents the sequence of computations needed to construct the mosaic from the input data. Montage will also perform a spatial search on the image collection metadata to find those files needed to fill the request. The data themselves will reside on high-quality disks, with high throughput I/O to the Teragrid processors that will be used by NVO services.

The result of the processing will be conveyed to the user through ROME. The user will receive a message that the data are available for pick-up until a deletion date. If the request was time intensive, the user may have logged off the portal and decided to wait for email notification. If the request could not be processed, ROME will be able to restart the job on the user's behalf. If only some intermediate products could be processed before the server failed, ROME will rerun the job, but find the intermediate products and use them as inputs. Many other partial processing examples can be handled easily within ROME.

6. Description of Data Formats and Image Data Collections

6.1 Flexible Image Transport System and the World Coordinate System

Montage will support only input and output files containing two-dimensional images that adhere to the definition of the Flexible Image Transport System (FITS) standard. FITS is the format adopted by the astronomical community for data interchange and archival storage [16][16]. All major astronomical image collections adhere to the FITS standard.

Briefly, FITS is a data format designed to provide a platform-independent means for exchange of astronomical data. A FITS data file is composed of a sequence of Header Data Units (HDUs). The header consists of "keyword=value" statements, which describe the organization of the data in the HDU and the format of the contents. It may provide additional information, for example, about instrument status or the history of the data. The data follow, structured as the header specifies.

The relationship between the pixel coordinates in the image and sky coordinates on the sky is defined by the World Coordinate System (WCS) [17]. Montage will support all the map projections supported by WCS.

All information describing the format and data type of the image, and its geometry on the sky (including WCS-supported map projection), are defined as header keywords in the FITS standard specifications. Montage will use these standard keywords to discover information on the format and geometry of an input image, and will use them to convey the corresponding information about the output images.

6.2 Image Data Collections

6.2.1 2MASS

2MASS is a ground-based survey that has imaged the entire sky at 1 arc second resolution in three near-infrared wavelengths, 1.25 μm (J Band), 1.65 μm (H Band), and 2.17 μm (K_S Band). Each positionally and photometrically calibrated 2MASS image is roughly 2 MB in size and contains 512 x 1,024 pixels covering roughly 0.15 x 0.30 degrees. The full data set, referred to as the “Atlas” images, contains 4,733, 227 images, with a total data volume of a little over 10 TB. A second image data set, called “Quicklook” images, is a compressed version of the Atlas data set. The compression factor is 20:1, but because the compression is lossy, the Quicklook images are suitable for browsing only.

6.2.2 DPOSS

DPOSS has captured nearly the entire northern sky at 1 arc second resolution in three wavelengths, 480 nm (J Band - blue), 650 nm (F Band - red), and 850 nm (N Band – near-infrared). The survey data were captured on photographic plates by the 48” Oschin Telescope at the Palomar Observatory in California **Error! Reference source not found.**. The total size of the DPOSS data accessible by *yourSky* is roughly 3 TB, stored in over 2,600 overlapping image plates. The DPOSS plates are each about 1 GB in size and contain 23,552 x 23,552 pixels covering a roughly 6.5 x 6.5 degree region of the sky.

6.2.3 SDSS

SDSS is using a dedicated 2.5 m telescope and a large format CCD camera to obtain images of over 10,000 square degrees of high Galactic latitude sky in five broad spectral bands (u', g', r', i' and z', centered at 3540, 4770, 6230, 7630, and 9130 Å, respectively). The final image data collection is scheduled for public release in July 2006. An initial public release in June 2001 covered about 460 square degrees of sky, and subsequent data releases will occur every 18 months or so until the full image collection is released in July 2006. This full collection will contain 1 billion Atlas images with a data volume of 1.5 TB.

6.3 Disposition of the Image Data Collections

6.3.1 2MASS

Currently, 47% of the 2MASS Image data collection has been released to the public, roughly 1.8 million images with a data volume of 4 TB. The images are stored on the High Performance Storage Server (HPSS) at the San Diego Supercomputer Center (SDSC), and managed by SDSC’s Storage Resource Broker (SRB). The SRB is a scalable client-server system that provides a uniform interface for connecting to heterogeneous data resources, transparently manages replicas of data collections, and organizes data into “containers” for efficient access. The *yourSky* server uses a set of client programs called SRB Tools to access selected 2MASS plates in batch mode from the SRB, and the same client is adequate to support development of Montage.

As part of the NVO project, SDSC will replicate the 2MASS data on spinning disk there and via SRB to a mirrored HPSS system at CACR. The schedule has to be determined, but it is anticipated that the replication can be performed before the end of December 2002.

6.3.2 DPOSS

The DPOSS data are currently replicated on the HPSS system at CACR. SDSC has committed to replicating the data at SDSC for processing under the NVO.

6.3.3 SDSS

The publicly released SDSS images are currently served from the SDSS archive at the MultiMission Archive at Space Telescope (MAST), where they reside on spinning disk. Our intention is to replicate the public data on spinning disk at SDSC. SDSS has informally agreed to this plan, but a formal agreement has yet to be put in place. This agreement will be negotiated by the NVO project.

7. Montage Design and Use Cases

This section demonstrates how the flexible and modular design of Montage supports the Science Use Cases described in the Software Engineering Plan [4]**Error! Reference source not found.**.

Use Case I - Science Analysis

The Spitzer First Look ancillary VLA image is a 2x2 degree radio image of a field that will be observed by Spitzer. As a field uncluttered by galactic radiation in Spitzer continuous viewing zone, it is a prime candidate for deep imaging of extragalactic sources. The VLA image contains many radio “blobs,” many of which appear to be interesting and perhaps bizarre objects. Interpretation of these objects requires multi-wavelength measurements on a common projection and spatial scale. DPOSS, SDSS and 2MASS provide the broad wavelength base for analysis of these objects, yet analysis is tedious and error prone because the images delivered by these projects have different spatial resolutions, coordinates and projections. MONTAGE will eliminate these difficulties by delivering mosaics from these data sets at a common resolution, projection and in a common coordinate system.

This is a basic small region mosaic problem and can be run on a single workstation or collection of workstations. Since the comparison will be with the VLA image, the mosaic should be constructed using the same projection and scale. The processing steps could in fact be run manually and would be as follows:

- Extract FITS header from VLA image
- Identify 2MASS (or whatever) images for the region and collect the images if running this in the standalone (*i.e.* non-GRID) mode. There are IRSA web services to do this in the case of the 2MASS images and we expect similar services to be available for the other datasets at some future date.
- Using the FITS header, reproject each of the input images to the new system using **mProject** for each one individually or **mProjExec** to process them all in a loop (based on a summary list prepared by **mImgtbl**). This step takes by far the majority of the time.
- Since this is a small region, the user will probably opt to have a custom background correction fit made. The first step in this is to determine exactly which image overlap, using **mOverlaps** acting on a summary metadata table for the reprojected images (again prepared by **mImgtbl**).
- **mDiff** is then used to actually generate the difference images for the overlapping pairs identified in the last step. This is usually run in a loop by **mDiffExec** using the table output by **mOverlaps**.
- **mFitplane** characterizes each difference image by a least-squares fit plane (excluding flux outlier pixels). This is usually run in a loop using **mFitExec**, which works off the table prepared by **mOverlaps**. The results go into a table used in the next step.
- **mBgModel** iteratively fits the table generated by **mFitplane**/**mFitExec** and determines the “best” background to remove from each of the original reprojected images.
- The final step in the background correction process is to apply the corrections to the images. This is done using **mBackground** on each image (usually by way of **mBgExec** looping over the table generated by **mBgModel**).
- These corrected/reprojected images can now be coadded into the final mosaic using **mAdd** (again using a summary metadata table for the corrected images prepared by **mImgtbl**).

Use Case II – Observation Planning

The Multiband Imaging Photometer (MIPS) [19] aboard the Spitzer Telescope has a scan length of 0.5°. Observations with MIPS must avoid bright sources that will saturate the detector, and is normally done by identifying infrared sources on 2MASS images. This is at present difficult to do because the 2MASS images are 512 x 1024 arcsec on a side and the effects of background variation from image to image complicate identification of sources in a consistent way. Mosaics of 2MASS images that have a flat background (not necessarily science grade) will make the task of identifying bright sources much easier to perform.

Here the need is for a global mosaic of the entire 2MASS dataset. While the scenario in Use Case I still applies, the processing is operationally quite different. Here, the entire 2MASS dataset should be reprojected into a regular pattern of large image outlines covering the sky, on the order of 5-10 degrees in scale. The overlap analysis and background fitting should be done once globally (or in a hierarchical local/global way) and the correction parameters for all 2MASS images stored in a permanent public database.

Since this would be done using GRID resources, the parallelization inherent in the architecture can be exploited to the maximum. Rather than use **mProjExec**, all the re-projection jobs can be added to a pool of tasks and performed by as many processors as are available. The same is true of the other "list driven" processes above (**mDiffExec**, **mFitExec**, **mBgExec**). The precise methodology to be used is TBD but will be built using standard GRID programming toolkits (Globus, Condor, DAGMAN, *etc*). The Users Guide delivered with the Montage software will give full details on how users can apply these grid resources.

Requests for mosaics of a specific location could then be satisfied by simply background subtracting (**mBackground**) and co-adding (**mAdd**) the already reprojected images (which would be kept permanently). There would also probably be standard "products"; images on the plate scale defined above covering the whole sky.

If a custom projection was desired, the original images would probably be used (to avoid losses due to repeated projection), re-projecting (**mProject**) them as desired but using the "standard" background correction parameters from the database instead of the background modeling described above.

Use Case III – Science Product Generation

The Galactic Legacy Infrared Midplane Survey Extraordinaire (GLIMPSE) will use the Spitzer Infra Red Array Camera (IRAC)[20] to survey approximately 220 square degrees of the Galactic plane, covering a latitude range of ± 1°, and a longitude range of abs(l)=10-65 °. GLIMPSE will be a confusion-limited survey of the Galactic Plane (approximately 300 mJy) in the four IRAC bands. The survey will produce several hundred GB of data in the form of catalogs and images, which will be delivered to the SIRTF Science Center for dissemination to the entire astronomical community. The GLIMPSE project requires a mosaic engine that is portable, uses only standard astronomy packages, is highly scaleable and is easy to fine-tune. These are the goals of Montage, which is therefore a serious candidate for GLIMPSE processing.

In this case, the input data set is not one of the data sets being used for Montage development and testing and the processing will be run on a custom cluster of processing engines (using home-grown pipeline executive code). The Montage modules are meant to be flexible enough to accommodate any FITS image, so the same paradigm as described in Use Case I should work. Here, however, the user would probably opt for writing their own executive logic rather than using the **mProjExec**, **mDiffExec**,

mFitExec, and **mBgExec** modules (which are simple constructs in any case) and manage parallelization themselves (or using off-the-shelf tools such as Condor). Only the executive logic needs customization: the processing modules will be used as delivered. The Montage User's Guide will give a complete description of how users can build their own executives.

Use Case IV – Outreach

Large-scale image mosaics are useful in promoting general interest in infrared astronomy through their use in local image galleries as well as the development of posters, pamphlets, and other media for both the general public and educators. Mosaics showing data at multiple wavelengths on a common projection, spatial scales etc exert a powerful influence on the imagination, especially when made part of a larger permanent display at a museum or planetarium. Access to Montage will allow production of large scale images from multiple data sets that would otherwise be very labor-intensive to accomplish.

Since such images will need to be on a common scale, much the same processing should be used as in Use Case I. Not all of these images will be mosaics, however. Some will be simple re-projections of existing images to put them all on the same scale. This can be done by running them individually through **mProject**.

References

[1] Teragrid website: http://teragrid.org/

[2] Construx Survival Guide Coding Standards: http://www.construx.com/survivalguide/

[3] "Software Requirements Specification for Montage". Version 1.0 (May 31, 2002); http://montage.ipac.caltech.edu/projectdocs/Requirements.doc

[4] "Software Engineering Plan for Montage". Version 1.0 (May 31, 2002); http://montage.ipac.caltech.edu/projectdocs/SEP.doc

[5] J. O'Rourke, *Computational Geometry in C* (Cambridge University Press, 1998). p220. (Chapter 7)

[6] Definition of Girard's Theorem http://math.rice.edu/~pcmi/sphere.

[7] A.S. Fruchter, and R.N. Hook. "Linear Reconstruction of the Hubble Deep Field," http://www.stsci.edu/~fruchter/dither/drizzle.html

[8] Mopex, the Spitzer Science Center Mosaic Engine, http://ssc.spitzer.caltech.edu/postbcd/doc/mosaicer.pdf

[9] Montage Milestones (available at http://montage.ipac.caltech.edu/ms.html)

[10] Pegasus website: http://www.isi.edu/~deelman/pegasus.htm. See also: E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, M. Livny, *Pegasus: Mapping Scientific Workflows onto the Grid*, Across Grids Conference 2004, Nicosia, Cyprus

[11] Condor website: http://www.cs.wisc.edu/condor/

[12] Montage Web Portal: http://montage.jpl.nasa.gov/

[13] Montage online documentation: http://montage.ipac.caltech.edu/docs

[14] Description of the IRSA "svc" library. http://montage.ipac.caltech.edu/Documentation/svc.html

[15] "An Architecture for Access to a Compute Intensive Image Mosaic Service in the NVO". G. Bruce Berriman , David Curkendall, John Good , Joseph Jacob, Daniel S. Katz, Mihseh Kong, Serge Monkewitz , Reagan Moore, Thomas Prince, Roy Williams. To appear in "Astronomical Telescopes & Instrumentation: Virtual Observatories," SPIE 4686-18

[16] The Flexible Image Transport System (FITS), http://fits.gsfc.nasa.gov, http://www.cv.nrao.edu/fits

[17] E.W. Greisen and M. Calabretta, *Representation of Celestial Coordinates In FITS*, http://www.atnf.csiro.au/people/mcalabre/WCS.htm

[18] The Digitized Palomar Observatory Sky Survey (DPOSS), http://www.astro.caltech.edu/~george/dposs

[19] MIPS website: http://spitzer.caltech.edu/SSC/MIPS/mips_intro.html

[20] IRAC website: http://spitzer.caltech.edu/SSC/IRAC/SSC_B4.html

Acronyms

2MASS	Two Micron All Sky Survey
ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CACR	Center for Advanced Computing Research
CCD	Charge Coupled Device
CGI	Common Gateway Interface
DAG	Directed Acyclical Graph
DBMS	DataBase Management System
DPOSS	Digital Palomar Observatory Sky Survey
EJB	Enterprise Java Beans
FITS	Flexible Image Transport System
GB	Giga Byte
GLIMPSE	Galactic Legacy Infrared Midplane Survey Extraordinaire
GNU	Gnu's Not Unix
HDU	Header Data Unit
HEASARC	High Energy Astrophysics Science ARChive
HPSS	High Performance Storage Server
HTTP	Hyper Text Transfer Protocol
IDE	Interactive Development Environment
IPAC	Infrared Processing and Analysis Center
IPG	Information Power Grid
IRAC	InfraRed Array Camera
IRSA	InfraRed Science Archive
ISI	Information Sciences Institute
JPL	Jet Propulsion Laboratory
MAST	MultiMission Archive at Space Telescope
MCS	Metadata Catalog Service
MDS	Globus Monitoring and Discovery Service
MIPS (astronomy)	Multiband Infrared Photometer for Spitzer
MIPS (computers)	Million Instructions per Second
MSX	Midcourse Space Experiment
NCSA	National Center for Supercomputing Applications
NSF	National Science Foundation
NVO	National Virtual Observatory
PBS	Portable Batch System
OASIS	On-Line Archive Science Information Services

RLS	Globus Replica Location Service
ROME	Request Object Management Environment
SAO	Smithsonian Astrophysical Observatory
SDSC	San Diego Supercomputer Center
SDSS	Sloan Digital Sky Survey
SRB	Storage Resource Broker
STScI	Space Telescope Science Institute
TB	Tera Byte
TBD	To Be Decided
URL	Uniform Resource Locator
VLA	Very Large Array
WCS	World Coordinate System
XML	eXtensible Markup Language

Glossary

Condor	A workload management system for compute-intensive jobs
DAGMan	Scheduler to submit concrete workflows to Condor
MyProxy	Credential respository for the TeraGrid
Pegasus	Scheduling software that transforms abstract workflows into concrete workflows, which can then be submitted using DAGMan
TeraGrid	A distributed computing infrastructure for open scientific research

Appendix A: Sample Files Used With TeraGrid Portal

A1. Sample XML Abstract Workflow for M51

adag_M51.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<adag xmlns="http://www.griphyn.org/chimera/DAX"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.griphyn.org/chimera/DAX
http://www.griphyn.org/chimera/dax-1.5.xsd"
      count="1" index="0" name="test">

      <filename file="2mass-atlas-980527n-j0230033.fits" link="input"
isTemporary="false"/>
      <filename file="2mass-atlas-980527n-j0240232.fits" link="input"
isTemporary="false"/>

      <filename file="p2mass-atlas-980527n-j0230033.fits" link="inout"
isTemporary="true" temporaryHint="temp"/>
      <filename file="p2mass-atlas-980527n-j0240232.fits" link="inout"
isTemporary="true" temporaryHint="temp"/>

      <filename file="diff.1.2.fits" link="inout" isTemporary="true"
temporaryHint="temp"/>

      <filename file="fit.1.2.txt" link="inout" isTemporary="true"
temporaryHint="temp"/>
      <filename file="fits_M51.tbl" link="inout" isTemporary="true"
temporaryHint="temp"/>

      <filename file="corrections.tbl" link="inout" isTemporary="true"
temporaryHint="temp"/>

      <filename file="c2mass-atlas-980527n-j0230033.fits" link="inout"
isTemporary="true" temporaryHint="temp"/>
      <filename file="c2mass-atlas-980527n-j0240232.fits" link="inout"
isTemporary="true" temporaryHint="temp"/>

      <filename file="p2mass-atlas-980527n-j0230033_area.fits"
link="inout" isTemporary="true" temporaryHint="temp"/>
      <filename file="c2mass-atlas-980527n-j0230033_area.fits"
link="inout" isTemporary="true" temporaryHint="temp"/>
      <filename file="p2mass-atlas-980527n-j0240232_area.fits"
link="inout" isTemporary="true" temporaryHint="temp"/>
      <filename file="c2mass-atlas-980527n-j0240232_area.fits"
link="inout" isTemporary="true" temporaryHint="temp"/>
      <filename file="diff.1.2_area.fits" link="output"
isTemporary="true" temporaryHint="temp"/>

      <filename file="pimages_M51.tbl" link="input"
isTemporary="false"/>
      <filename file="cimages_M51.tbl" link="input"
isTemporary="false"/>
```

```
      <filename file="fit_list_M51.tbl" link="input"
isTemporary="false"/>
      <filename file="template_M51.hdr" link="input"
isTemporary="false"/>
      <filename file="out_M51.fits" link="output" isTemporary="false"/>

      <filename file="out_M51_area.fits" link="output"
isTemporary="false"/>

      <job name="mProject" id="ID000001">
            <argument>
                  <filename file="2mass-atlas-980527n-j0230033.fits"
link="input" isTemporary="false"/>
                  <filename file="p2mass-atlas-980527n-j0230033.fits"
link="output" isTemporary="true"/>
                  <filename file="template_M51.hdr" link="input"
isTemporary="false"/>
            </argument>
            <uses file="2mass-atlas-980527n-j0230033.fits" link="input"
isTemporary="false"/>
            <uses file="p2mass-atlas-980527n-j0230033.fits"
link="output" isTemporary="true"/>
            <uses file="p2mass-atlas-980527n-j0230033_area.fits"
link="output" isTemporary="true"/>
            <uses file="template_M51.hdr" link="input"
isTemporary="false"/>
      </job>

      <job name="mProject" id="ID000002">
            <argument>
                  <filename file="2mass-atlas-980527n-j0240232.fits"
link="input" isTemporary="false"/>
                  <filename file="p2mass-atlas-980527n-j0240232.fits"
link="output" isTemporary="true"/>
                  <filename file="template_M51.hdr" link="input"
isTemporary="false"/>
            </argument>
            <uses file="2mass-atlas-980527n-j0240232.fits" link="input"
isTemporary="false"/>
            <uses file="p2mass-atlas-980527n-j0240232.fits"
link="output" isTemporary="true"/>
            <uses file="p2mass-atlas-980527n-j0240232_area.fits"
link="output" isTemporary="true"/>
            <uses file="template_M51.hdr" link="input"
isTemporary="false"/>
      </job>

      <job name="mDiff" id="ID000003">
            <argument>
                  <filename file="p2mass-atlas-980527n-j0230033.fits"
link="input" isTemporary="true"/>
                  <filename file="p2mass-atlas-980527n-j0240232.fits"
link="input" isTemporary="true"/>
                  <filename file="diff.1.2.fits" link="output"
isTemporary="true"/>
                  <filename file="template_M51.hdr" link="input"
isTemporary="false"/>
```

```
            </argument>
            <uses file="p2mass-atlas-980527n-j0230033.fits"
link="input" isTemporary="true"/>
            <uses file="p2mass-atlas-980527n-j0230033_area.fits"
link="input" isTemporary="true"/>
            <uses file="p2mass-atlas-980527n-j0240232.fits"
link="input" isTemporary="true"/>
            <uses file="p2mass-atlas-980527n-j0240232_area.fits"
link="input" isTemporary="true"/>
            <uses file="diff.1.2.fits" link="output"
isTemporary="true"/>
            <uses file="diff.1.2_area.fits" link="output"
isTemporary="true"/>
            <uses file="template_M51.hdr" link="input"
isTemporary="false"/>
      </job>

      <job name="mFitplane" id="ID000004">
            <argument>
                  -s
                  <filename file="fit.1.2.txt" link="output"
isTemporary="true"/>
                  <filename file="diff.1.2.fits" link="input"
isTemporary="true"/>
            </argument>
            <uses file="fit.1.2.txt" link="output" isTemporary="true"/>
            <uses file="diff.1.2.fits" link="input"
isTemporary="true"/>
      </job>

      <job name="mConcatFit" id="ID000005">
            <argument>
                  <filename file="fit_list_M51.tbl" link="input"
isTemporary="false"/>
                  <filename file="fits_M51.tbl" link="output"
isTemporary="true"/>
                  .
            </argument>
            <uses file="fit_list_M51.tbl" link="input"
isTemporary="false"/>
            <uses file="fits_M51.tbl" link="output"
isTemporary="true"/>
            <uses file="fit.1.2.txt" link="input" isTemporary="true"/>
      </job>

      <job name="mBgModel" id="ID000006">
            <argument>
                  <filename file="pimages_M51.tbl" link="input"
isTemporary="false"/>
                  <filename file="fits_M51.tbl" link="input"
isTemporary="true"/>
                  <filename file="corrections.tbl" link="output"
isTemporary="true"/>
            </argument>
            <uses file="pimages_M51.tbl" link="input"
isTemporary="false"/>
            <uses file="fits_M51.tbl" link="input" isTemporary="true"/>
            <uses file="corrections.tbl" link="output"
isTemporary="true"/>
      </job>

      <job name="mBackground" id="ID000007">
            <argument>
                  -t
                  <filename file="p2mass-atlas-980527n-j0230033.fits"
link="input" isTemporary="true"/>
                  <filename file="c2mass-atlas-980527n-j0230033.fits"
link="output" isTemporary="true"/>
                  <filename file="pimages_M51.tbl" link="input"
isTemporary="false"/>
                  <filename file="corrections.tbl" link="input"
isTemporary="true"/>
            </argument>
            <uses file="p2mass-atlas-980527n-j0230033.fits"
link="input" isTemporary="true"/>
            <uses file="p2mass-atlas-980527n-j0230033_area.fits"
link="input" isTemporary="true"/>
            <uses file="pimages_M51.tbl" link="input"
isTemporary="false"/>
            <uses file="corrections.tbl" link="input"
isTemporary="true"/>
            <uses file="c2mass-atlas-980527n-j0230033.fits"
link="output" isTemporary="true"/>
            <uses file="c2mass-atlas-980527n-j0230033_area.fits"
link="output" isTemporary="true"/>
      </job>

      <job name="mBackground" id="ID000008">
            <argument>
                  -t
                  <filename file="p2mass-atlas-980527n-j0240232.fits"
link="input" isTemporary="true"/>
                  <filename file="c2mass-atlas-980527n-j0240232.fits"
link="output" isTemporary="true"/>
                  <filename file="pimages_M51.tbl" link="input"
isTemporary="false"/>
                  <filename file="corrections.tbl" link="input"
isTemporary="true"/>
            </argument>
            <uses file="p2mass-atlas-980527n-j0240232.fits"
link="input" isTemporary="true"/>
            <uses file="p2mass-atlas-980527n-j0240232_area.fits"
link="input" isTemporary="true"/>
            <uses file="pimages_M51.tbl" link="input"
isTemporary="false"/>
            <uses file="corrections.tbl" link="input"
isTemporary="true"/>
            <uses file="c2mass-atlas-980527n-j0240232.fits"
link="output" isTemporary="true"/>
            <uses file="c2mass-atlas-980527n-j0240232_area.fits"
link="output" isTemporary="true"/>
      </job>

      <job name="mAdd" id="ID000009">
```

```
            <argument>
                  <filename file="cimages_M51.tbl" link="input"
isTemporary="false"/>
                  <filename file="template_M51.hdr" link="input"
isTemporary="false"/>
                  <filename file="out_M51.fits" link="output"
isTemporary="false"/>
            </argument>
            <uses file="cimages_M51.tbl" link="input"
isTemporary="false"/>
            <uses file="template_M51.hdr" link="input"
isTemporary="false"/>
            <uses file="out_M51.fits" link="output"
isTemporary="false"/>
            <uses file="out_M51_area.fits" link="output"
isTemporary="false"/>
            <uses file="c2mass-atlas-980527n-j0230033.fits"
link="input" isTemporary="true"/>
            <uses file="c2mass-atlas-980527n-j0230033_area.fits"
link="input" isTemporary="true"/>
            <uses file="c2mass-atlas-980527n-j0240232.fits"
link="input" isTemporary="true"/>
            <uses file="c2mass-atlas-980527n-j0240232_area.fits"
link="input" isTemporary="true"/>
      </job>

      <child ref="ID000003">
            <parent ref="ID000001"/>
            <parent ref="ID000002"/>
      </child>

      <child ref="ID000004">
            <parent ref="ID000003"/>
      </child>

      <child ref="ID000005">
            <parent ref="ID000004"/>
      </child>

      <child ref="ID000006">
            <parent ref="ID000005"/>
      </child>

      <child ref="ID000007">
            <parent ref="ID000001"/>
            <parent ref="ID000006"/>
      </child>

      <child ref="ID000008">
            <parent ref="ID000002"/>
            <parent ref="ID000006"/>
      </child>

      <child ref="ID000009">
            <parent ref="ID000007"/>
            <parent ref="ID000008"/>
      </child>
```

```
</adag>
```

A2. Sample Image Table

images_M51.tbl:

```
\datatype=fitshdr
| cntr|  ctype1|  ctype2|naxis1|naxis2|    crval1|    crval2| crpix1|
crpix2|     cdelt1|     cdelt2|  crota2|
SRB|                                        Quicklook|
file|
|  int|    char|    char|   int|   int|    double|    double| double|
double|     double|     double|  double|
char|                                                  char|
char|
     1 RA---SIN DEC--SIN    512   1024 202.361114  47.326477  256.50
512.50 -0.00027778  0.00027778 -0.02065 /home/thh.caltech/n-213031332-
980527-023-0033-j \2MASSDataPath\/980527n/s023/image/ji0230033.fits.H
2mass-atlas-980527n-j0230033.fits
     2 RA---SIN DEC--SIN    512   1024 202.521394  47.113192  256.50
512.50 -0.00027778  0.00027778  0.00339 /home/thh.caltech/n-213031332-
980527-024-0232-j \2MASSDataPath\/980527n/s024/image/ji0240232.fits.H
2mass-atlas-980527n-j0240232.fits
```

A3. Sample Projected Image Table

pimages_M51.tbl:

```
\datatype=fitshdr
| cntr|  ctype1|  ctype2|naxis1|naxis2|    crval1|    crval2|
crpix1|    crpix2|     cdelt1|     cdelt2|  crota2|equinox|
file|
|  int|    char|    char|   int|   int|    double|    double|
double|    double|     double|     double|  double|    int|
char|
     1 RA---TAN DEC--TAN    516   1027 202.441157  47.219954      63.50
130.00 -0.00027778  0.00027778  0.05936    2000 p2mass-atlas-980527n-
j0230033.fits
     2 RA---TAN DEC--TAN    513   1024 202.441157  47.219954     452.50
896.00 -0.00027778  0.00027778  0.05936    2000 p2mass-atlas-980527n-
j0240232.fits
```

A4. Sample Corrected Image Table

cimages_M51.tbl:

```
\datatype=fitshdr
| cntr|  ctype1|  ctype2|naxis1|naxis2|    crval1|    crval2|
crpix1|    crpix2|     cdelt1|     cdelt2|  crota2|equinox|
file|
|  int|    char|    char|   int|   int|    double|    double|
double|    double|     double|     double|  double|    int|
char|
     1 RA---TAN DEC--TAN    516   1027 202.441157  47.219954      63.50
130.00 -0.00027778  0.00027778  0.05936    2000 c2mass-atlas-980527n-
j0230033.fits
```

```
     2 RA---TAN DEC--TAN    513   1024 202.441157  47.219954     452.50
896.00 -0.00027778  0.00027778  0.05936    2000 c2mass-atlas-980527n-
j0240232.fits
```

A5. Sample Fit Plane File Table

fit_list_M51.tbl:
```
|cntr1|cntr2|   stat    |
   1     2   fit.1.2.txt
```

A6. Sample Montage Template File

template_M51.hdr:
```
SIMPLE  = T
BITPIX  = -64
NAXIS   = 2
NAXIS1  = 906
NAXIS2  = 1793
CTYPE1  = 'RA---TAN'
CTYPE2  = 'DEC--TAN'
EQUINOX = 2000
CRVAL1  =  202.441156834
CRVAL2  =   47.219953535
CDELT1  =   -0.000277780
CDELT2  =    0.000277780
CRPIX1  =       453.5000
CRPIX2  =       897.0000
CROTA2  =    0.059358733
END
```

A7. Sample Abstract Workflow Service Log File

log_M51.txt:
```
Fri Jan  9 09:33:34 PST 2004 : Started DAG processing for location
"m51" and size ".2"
[struct stat="OK", count="3"]
Fri Jan  9 09:33:38 PST 2004 : Created input image table with
m2MASSList
[struct stat="OK", count=2]
Fri Jan  9 09:33:38 PST 2004 : Created Montage template header file
with mMakeHdr
[struct stat="OK", count="2"]
Fri Jan  9 09:33:38 PST 2004 : Created projected and background
corrected image tables with mDAGTbls
[struct stat="OK", count=1]
Fri Jan  9 09:33:38 PST 2004 : Created list of overlapping images with
mOverlaps
Fri Jan  9 09:33:38 PST 2004 : Finished DAG processing for location
"m51" and size ".2"
```