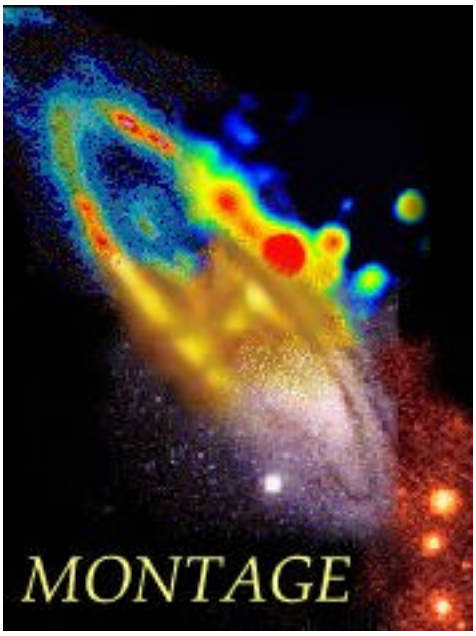


Software Design Specification for

MONTAGE

An Astronomical Image Mosaic Service for the National Virtual Observatory

Version 1.1 (November 18, 2002): 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
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.

Montage will run operationally on the *TeraGrid*, (http://www.npaci.edu/teragrid/). Users will submit requests to Montage through existing astronomical portals, and visualize and interact with the delivered mosaics through these same portals.

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.

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.
- Interface specifications ("Application Programmers Interface")
- Low level design, including error handling strategies and algorithms implemented by the system

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 then provide updates thereafter for subsequent releases. We anticipate that the design updates are incremental and will largely accommodate customers' requests for functionality.

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

SDS Version	Release Date	Montage Version (Release Date)	SDS Contents
1.0	6/30/2002	1.0.0 (2/28/2003)	Design considerations Overall architecture 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
2.0	8/30/2003	2.0.0 (2/28/2004)	Updates to 1.x as needed
3.0	8/30/2004	3.0.0 (1/10/2005)	Updates to 2.x as needed

1.3 Supporting Materials

Montage will be developed according to the style guidelines in http://www.construx.com/survivalguide/ ->Coding Standard

2 Design Considerations

2.1 Drivers and Constraints

The drivers and constraints governing the design of Montage are made clear in the requirements document [1] and in the Software Engineering Plan [2]. 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 code, correction for background radiation with parameters derived from correction algorithms supplied by data providers, and for background corrections with 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 2.300	HEASARC
WCSTools	Image projection	Version 3.0.5	SAO
Coord	Coordinate transformation	Version 1.2	IRSA

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 that can be parsed by any computer.

Montage will be constructed to operate entirely from command-line arguments, using the ancillary files describe 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 of course be tested and run operationally on the TeraGrid. We will formally test Montage on IRIX only as resources permit.

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

3 High-Level Architecture and Computational Algorithms

3.1 High Level Design of Montage

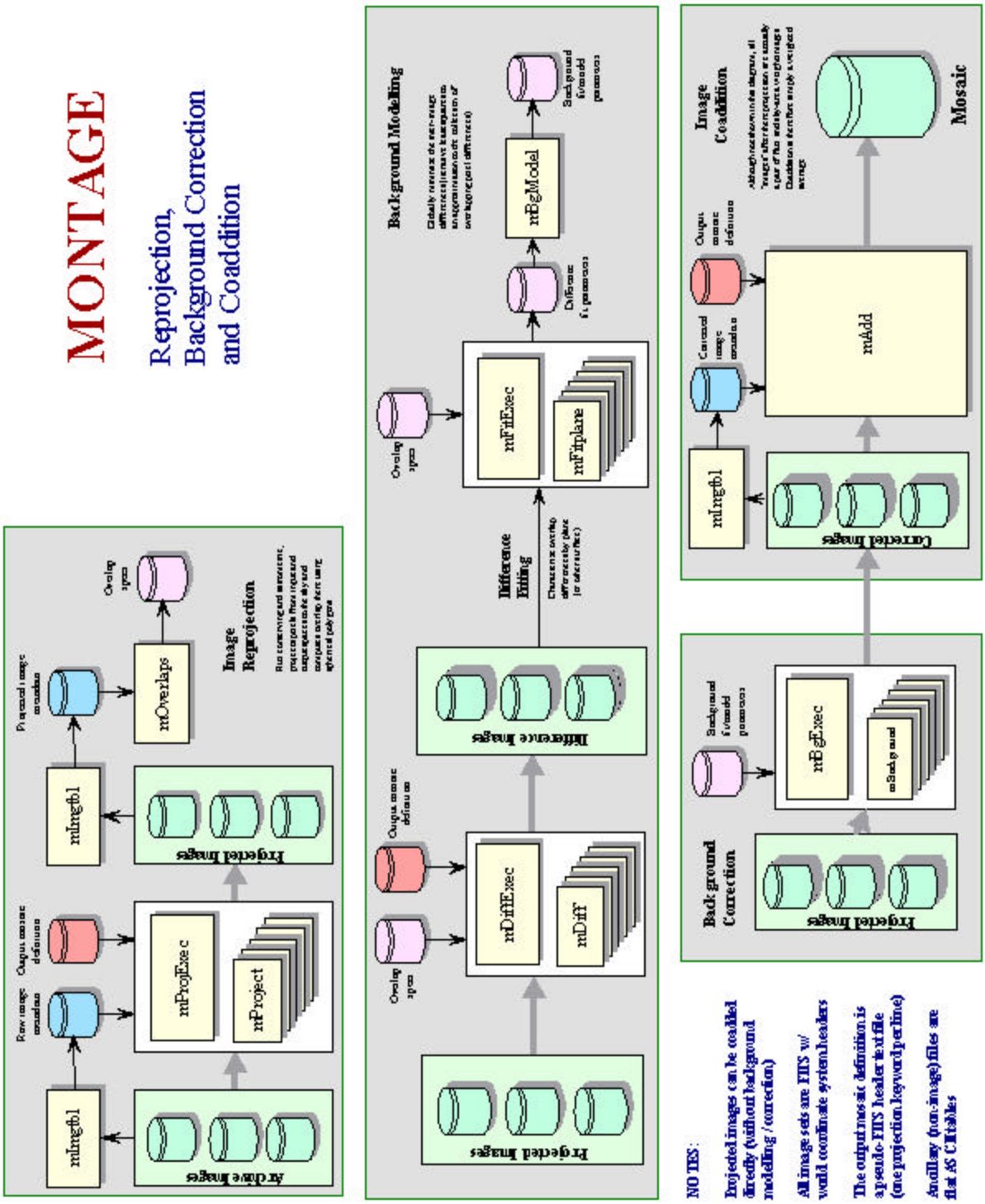
Processing a request for an image mosaic consists of three main steps

- 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 level;
- 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 minimize 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.

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.



The 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 the 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 the former is that it allows variations in the fitting algorithms to deal with the special cases and, for small regions, will probably be more sensitive to local variations than a global fit. The advantage of the latter is that it provides a uniform view of the sky and a tested "best fit" that can be certified as such by the project.

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.

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. Also in astronomy, it is 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.

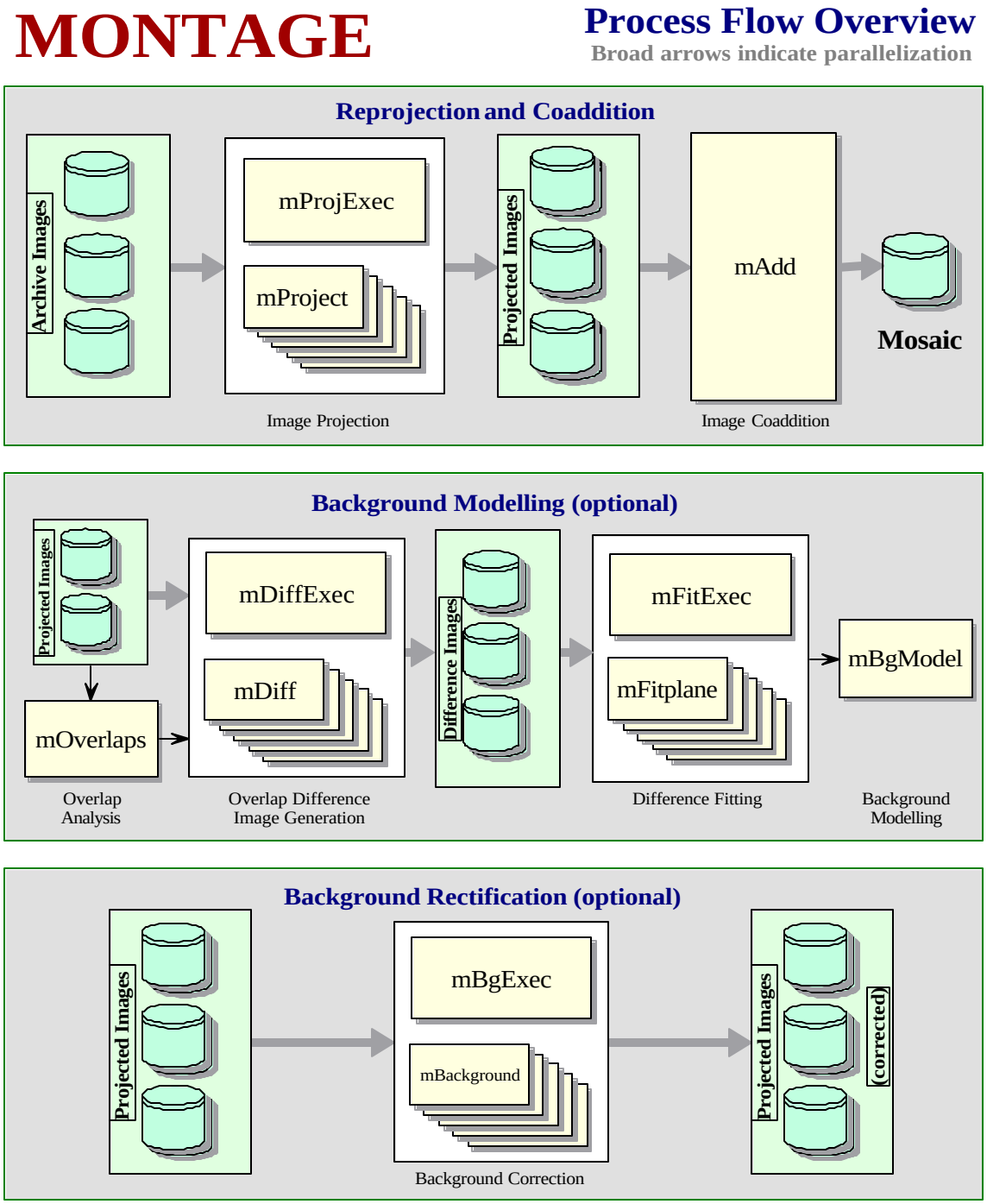


Figure 2: Design Components of Montage - Process Flow Overview.

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.

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 [3] 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 [4], 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 affect 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 [5]. 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 in Montage from its inception.

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 discrepencies 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 depend on a 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 it 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 it 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).

Such images are in practice very flat (with only slight slopes due to the image projection) since cumulative affect is that each output pixel is covered but 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.

When combining multiple overlapping images, these area images provide a natural weighting function; the output pixel value being the area-weighted average of the images being combined.

3.2.4 Parallelization

The first released code (to which this document applies) is intended to run on a single processor. Nevertheless, we can make some remarks on how the design supports parallelization. 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 cross-talk 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 process is unlikely to ever be a primary time sink so this will probably not be necessary.

4. Detailed Design of Montage

4.1. Interface Specifications

For Montage internal testing and error diagnosis, all programs can take a "-d(ebug) level" argument where 'level' is an integer denoting the detail of debugging output generated. For some programs this is just an on/off flag, for others it can have a value as high as 4. This debugging output will most likely be unavailable in the released code.

mImgtbl

Description

Extracts the FITS header geometry information from a set of files and creates an ASCII image metadata table from it used by several of the other programs.

Syntax: `mImgtbl <directory> <images.tbl>`

Input	**Datatype**	**Description**
`<directory>`	char	directory to be searched for FITS files
Output		**Description**
`<images.tbl>`	char	Column-delimited ASCII table file containing FITS file information on the geometry of each image on the sky.

mProject

Description

Reprojects a single image to the scale defined in a FITS –style header template file. Actually produces a pair of images: the reprojected image and an "area" image consisting of the fraction input pixel sky area that went into each output pixel. The "drizzle" algorithm is implemented. The algorithm proceeds by mapping pixel corners (as adjusted by drizzle, if called) from the input pixel space to the output pixel space, calculating overlap area with each output pixel, and accumulating an appropriate fraction of the input flux into the output image pixels. In addition, the appropriate fraction of the input pixel area is accumulated into the area image pixels. Projection of points from input pixel space to output pixel space is calculated in two steps: first map from input pixel space to sky coordinates; second map from sky coordinates to output pixel space.

Syntax: `mProject <in.fits> <out.fits> <template.hdr> [-drizzle <factor>]`

Input	Datatype	Description
`<in.fits>`	`char`	`<in.fits>` is the FITS image to be re-projected
`[-drizzle<factor>]`	Real*4	Invokes the STScI "drizzle" algorithm
Output		**Description**
<out.fits>	char	Output reprojected FITS file
`<template.hdr>`	char	Column delimited ASCII file containing the lines to be used to create the output FITS header.

mProjectExec

Description

A simple executive that runs mProject for each image in an image metadata table.

Syntax

`mProjExec <images.tbl> <template.hdr> <directory> <stats.tbl>`

Input	Datatype	Description
`<images.tbl>`	char	ASCII table created by `mImgtbl`
`<template.hdr>`	char	Column delimited ASCII file containing the lines to be used to create the output FITS header.
Output		**Description**
`<directory>`	char	Location where the reprojected images will be stored
`<stats.tbl>`	char	ASCII runtime log of the processing of the individual images

mAdd

Description
Coadd the reprojected images using the same FITS header template and working from the same mImgtbl list. Both pixel values and pixel areas are accumulated

Syntax
```
mAdd <images.tbl> <template.hdr><mosaic.fits>
```

Input	Datatype	Description
<images.tbl>	char	ASCII table created by mImgtbl
<template.hdr>	char	Column delimited ASCII file containing the lines to be used to create the output FITS header.
Output		**Description**
<mosaic.fits>	char	Final co-added mosaic in FITS format

mOverlaps

Description
Analyze an image metadata table to determine a list of overlapping images. Each image is compared with every other image to determine all overlapping image pairs. A pair of images are deemed to overlap if any pixel around the perimeter of one image falls within the boundary of the other image.

Syntax
```
mOverlaps <images.tbl> <diffs.tbl>
```

Input	Datatype	Description
<images.tbl>	char	ASCII table created by mImgtbl
Output		**Description**
<diffs.tbl>	char	Column delimited ASCII file that contains a summary of the pairs of images that overlap.

mDiff

Description

Analyze an image metadata table to determine a list of overlapping images. Each image is compared with every other image to determine all overlapping image pairs. A pair of images are deemed to overlap if any pixel around the perimeter of one image falls within the boundary of the other image.

Syntax

```
mDiff <in1.fits> <in2.fits> <template.hdr> <out.fits>
```

Input	Datatype	Description
<in2.fits>	Char	FITS format file to be differenced (locations specified by the <directory> output from mDiffExec)
<in2.fits>	Char	FITS format file to be differenced ((locations specified by the <directory> output from mDiffExec)
<template.hdr>	Char	Column delimited ASCII file containing the lines to be used to create the output FITS header.
Output		**Description**
<out.fits>	char	FTS format difference file

mDiffExec

Description

A simple executive that runs mDiff on each image pair identified by mOverlaps

Syntax

```
mDiffExec <diffs.tbl> <template.hdr> <directory>
```

Input	Datatype	Description
<diffs.tbl>	Char	Table created by mOverlaps
<template.hdr>	Char	
Output		**Description**
<template.hdr>	Char	Column delimited ASCII file containing the lines to be used to create the output FITS header.

mFitPlane

Description

Applies a least squares planar fit, excluding outlier pixels, to an image. Used on the difference images generated by mDiff.

Usage: `mFitplane <in.fits>`

Input: `<in.fits> are` the FITS images to be fit.
Output: Parameters describing plane are written as "stdout," where they are read by mDiffExec.

Syntax

`mFitPlane <in.fits>`

Input	Datatype	Description
`<in.fits>`	char	FITS image to which least squares fit is to be applied

Output		Description
Parameters describing plane	char	Parameters are written as "stdout," where they are read by mDiffExec

mFitExec

Description

A simple executive that runs mFitplane on all of the overlapping image pairs identified by mOverlaps. Creates a table of image-to-image difference parameters.

Syntax

`mFitExec <diffs.tbl> <fits.tbl>`

Input	Datatype	Description
`<diffs.tbl>`	char	Column delimited ASCII table created by `mOverlaps`

Output		Description
`<fits.tbl>`	char	Column-delimited ASCII file that contains the background fits for all images.

mBgModel

Description

Modeling/fitting program which uses the image-to-image difference parameter table to iteratively determine a set of corrections to apply to each image to achieve a "best" global fit. The algorithm proceeds by determining the neighbors of each image, determining (in a least squares sense) the best correction plane to match the image's background with its neighbors, and iterating for a fixed number of iterations to achieve convergence on a global solution.

Syntax

`mBgModel <images.tbl> <fits.tbl> <corrections.tbl>`

Input	Datatype	Description
`<images.tbl>`	Char	ASCII table created by `mImgtbl`
`<fits.tbl>`	Char	Column-delimited ASCII file that contains the background fits for all images.

Output		Description
`<corrections.tbl>`	Char	Column delimited ASCII file that contains the corrections to be applied to the original (projected) images.

mBackground

Description

Remove a background plane from a FITS image. The background correction applied to the image is specified as Ax+By+C, where (x,y) is the pixel coordinate using the image center as the origin, and (A,B,C) are the background plane parameters specified as linear coefficients.

Syntax

`mBackground <in.fits> <A> <B> <C> <xcenter> <ycenter> <out.fits>`

Input	Datatype	Description
`<in.fits>`	Char	FITS image to be background corrected
`<A>`	Real*4	Plane fit parameters, as stored in the `<corrections.tbl>` table by `mBgModel`.
`<B>`	Real*4	Plane fit parameters, as stored in the `<corrections.tbl>` table by `mBgModel`
`<C>`	Real*4	Plane fit parameters, as stored in the `<corrections.tbl>` table by `mBgModel`
`<xcenter>`	Real*4	Plane fit parameters, as stored in the `<corrections.tbl>` table by `mBgModel`
`<ycenter>`	Real*4	Plane fit parameters, as stored in the `<corrections.tbl>` table by `mBgModel`

Output		Description
`<out.fits>`	char	Output corrected FITS image

mBgExec

Description
A simple executive that runs `mBackground` on all the images in the metadata table

Syntax
```
mBgExec <images.tbl> <corrections.tbl> <directory>
```

Input	Datatype	Description
`<images.tbl>`		ASCII table created by `mImgtbl`
`<corrections.tbl>`		Column delimited ASCII file that contains the corrections to be applied to the original (projected) images.

Output	Description
`<directory>`	Location where the background corrected images will be stored.

mBgModel

Description
Determines through iterative least squares fitting the parameters of the background model

Syntax
```
mBgModel images.tbl fits.tbl corrections.tbl [-iteration niter]
```

Input	Datatype	Description
`<images.tbl>`	Char	ASCII table created by `mImgtbl`
`<corrections.tbl>`	Char	Column delimited ASCII file that contains the corrections to be applied to the original (projected) images.
[-iteration niter]	Int	The number of interations to run on the background model (defaults to 25). There is no convergence criterion on this algorithm yet but the algorithm is fast and usually converges quickly.

Output		Description
`<fits.tbl>`	char	This table contains the set of plane parameters fit to the difference image generated from the diffs.tbl list.

4.2. Definitions of Montage File Formats

4.2.1. ASCII Table formats & the images.tbl file

The modules described above read and generate column delimited flat ASCII table files, whose format is obvious from this description. One of these files, 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 use 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

Column	Definition	FITS standard?
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
crpix2	The pixel scale (in degrees on the sky per pixel) at the reference location for dimensions 1 and 2	Y
cdelt1, cdelt2	The pixel scale (in degrees on the sky per pixel) cdelt2 = at the reference location	Y
crota2	The rotation angle from the "up" direction to the celestial pole	Y
fname	The path to the original FITS file	N

4.2.2. The Template.hdr file

Description:
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. Often generated by hand but can be created by `mMakeHdr` analyzing an images.tbl file.

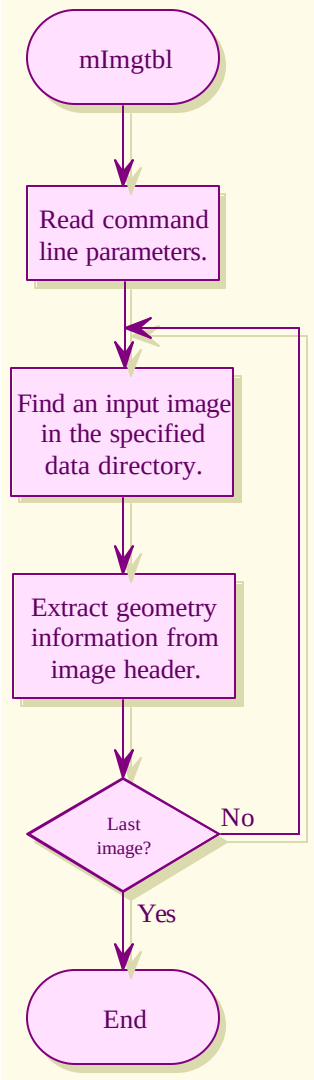
FITS headers consist of a variable of number of 80-character card images at the beginning of the file concatenated together with no punctuation.

The template.hdr files used by Montage differs from this only in that the card images are one to a line (left justified and newline delimited) and the lines can be any length less than 80 characters. Other than that, the information content is identical; 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

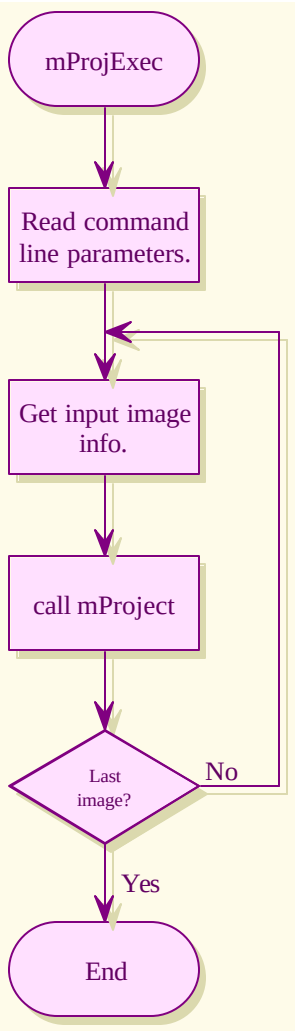
```
SIMPLE  =                    T /
BITPIX  =                  -64 /
NAXIS   =                    2 /
NAXIS1  =                 3000 /
NAXIS2  =                 3000 /
CDELT1  =         -3.333333E-4 /
CDELT2  =          3.333333E-4 /
CRPIX1  =               1500.5 /
CRPIX2  =               1500.5 /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

4.3. Design of Montage Modules: Flow Charts

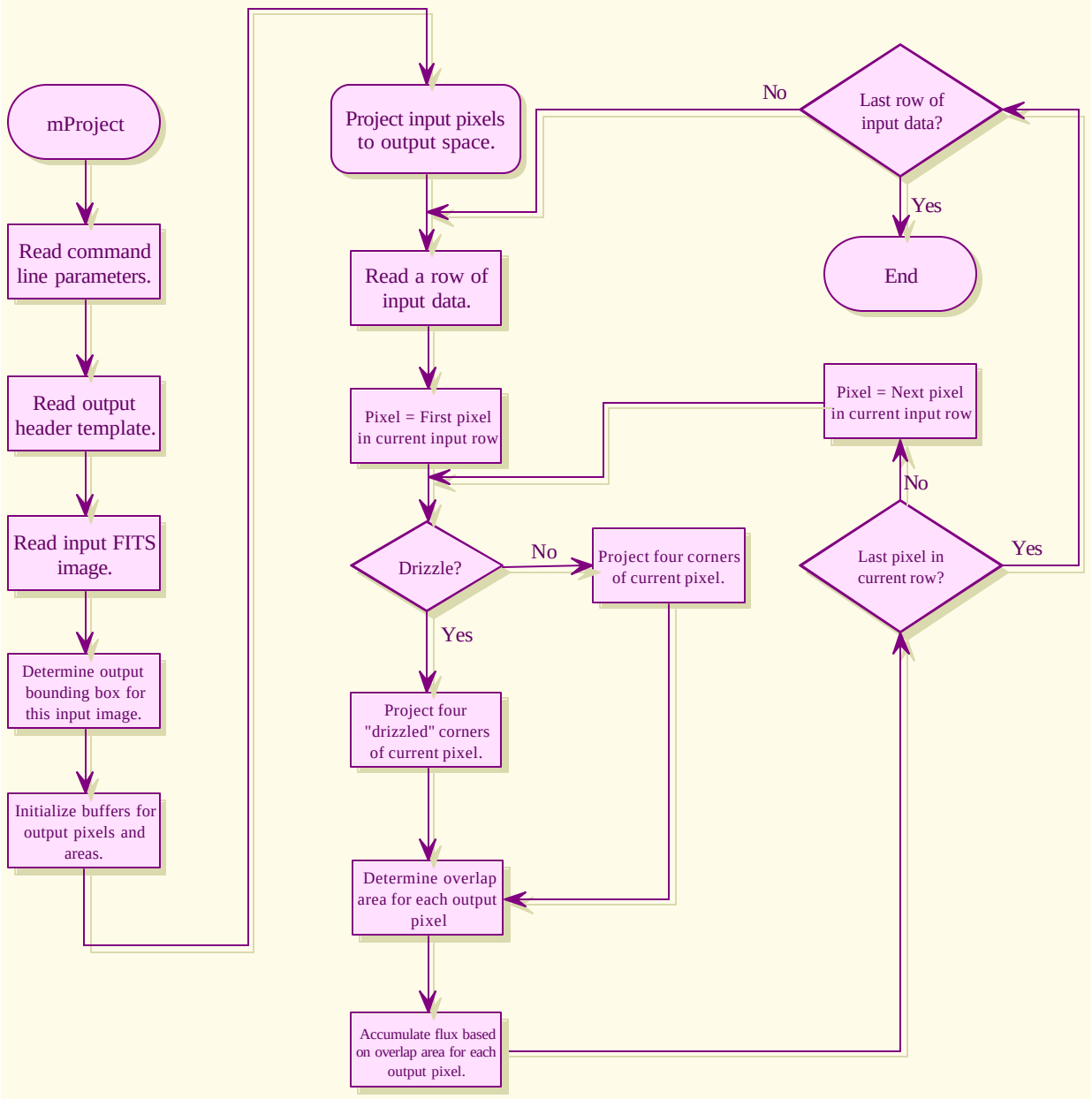
mImgtbl



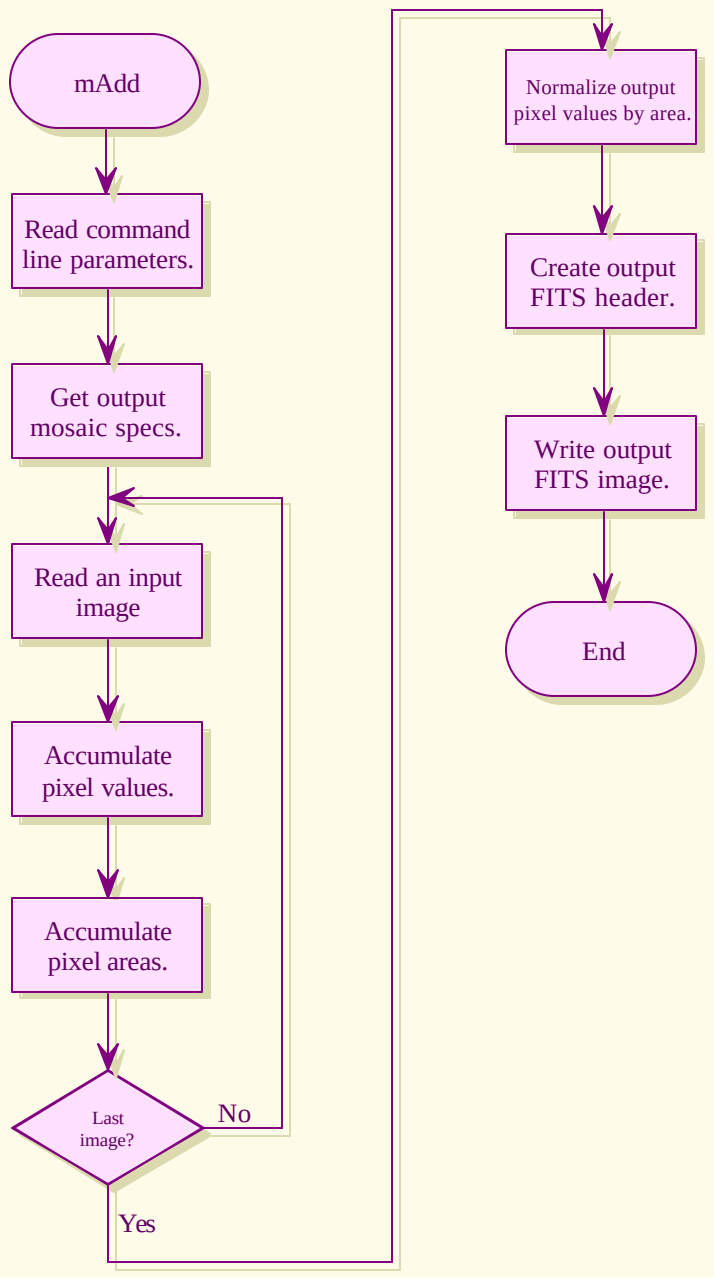
mProjectExec



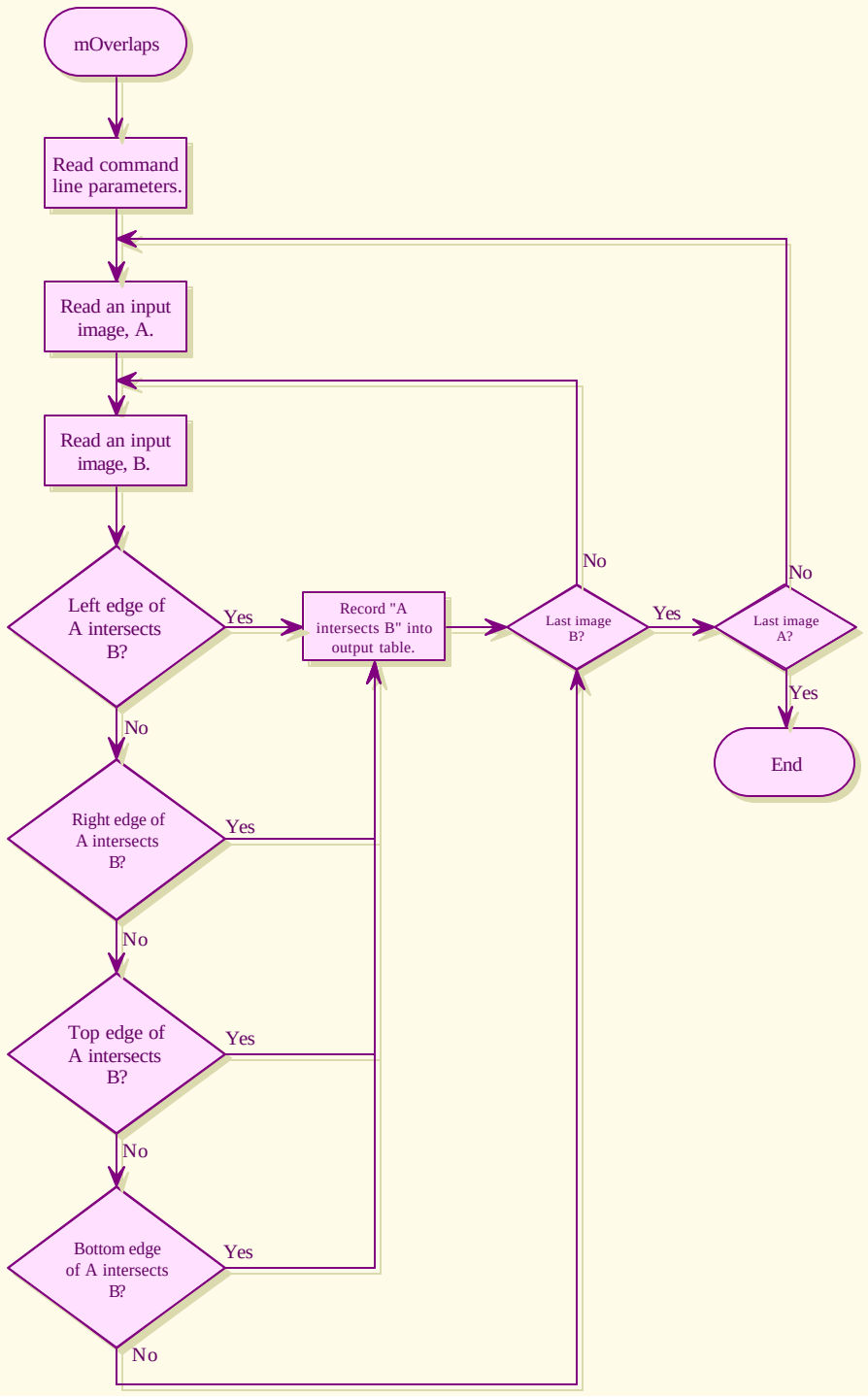
mProject


mProject
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.
Pixel = Next pixel in current input row
No
Last pixel in current row?
Yes
No
Last row of input data?
Yes
End

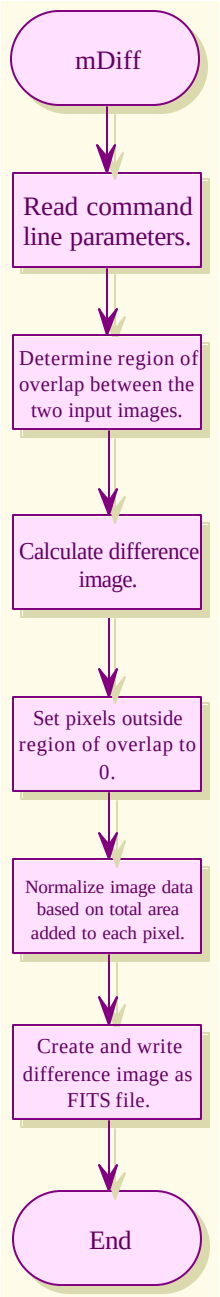
mAdd


mAdd
Read command line parameters.
Get output mosaic specs.
Read an input image
Accumulate pixel values.
Accumulate pixel areas.
Last image?
No
Yes
Normalize output pixel values by area.
Create output FITS header.
Write output FITS image.
End

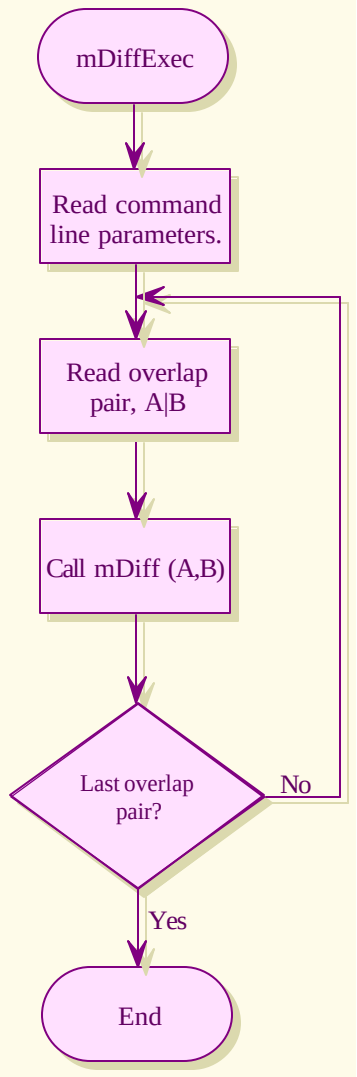
mOverlaps


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

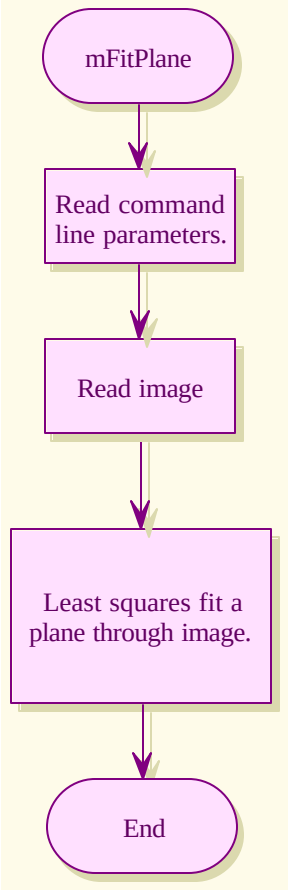
mDiff



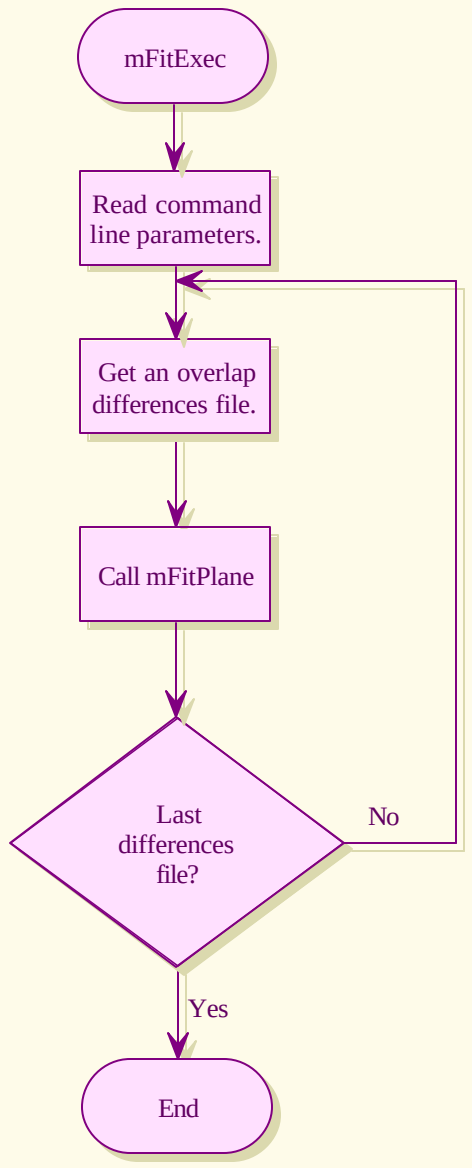
mDiffExec



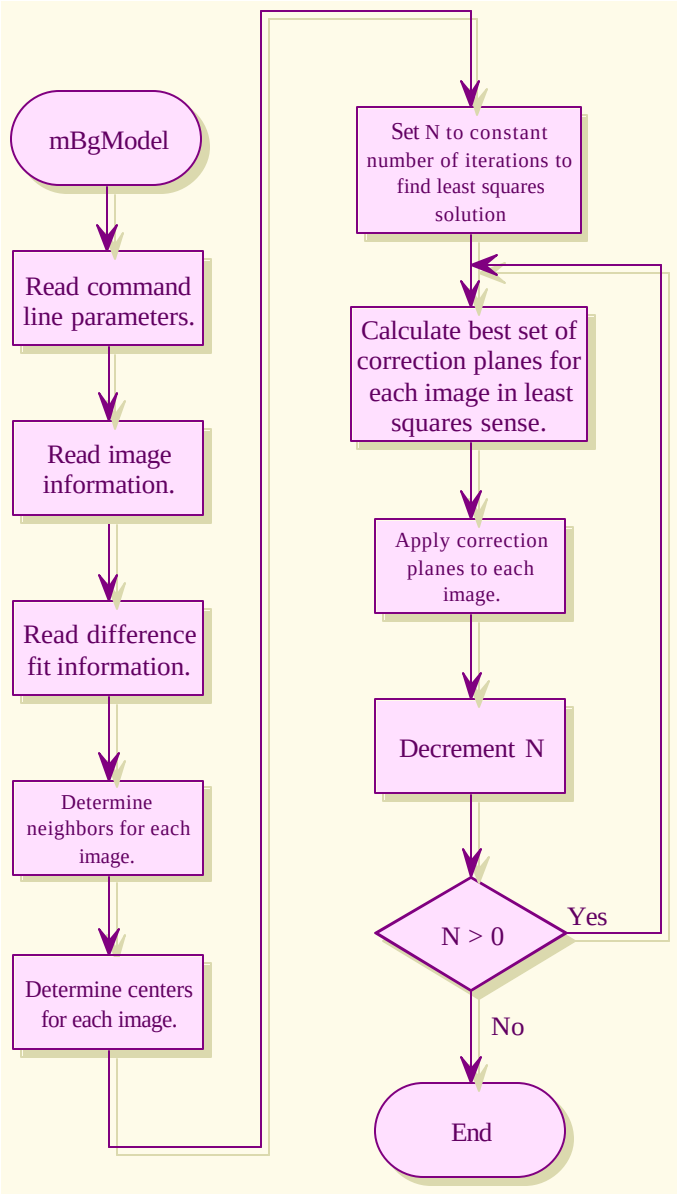
mFitPlane



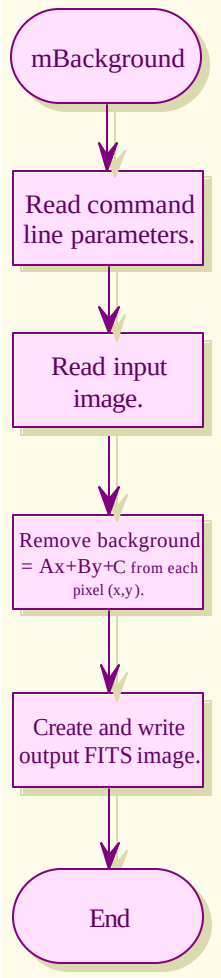
mFitExec



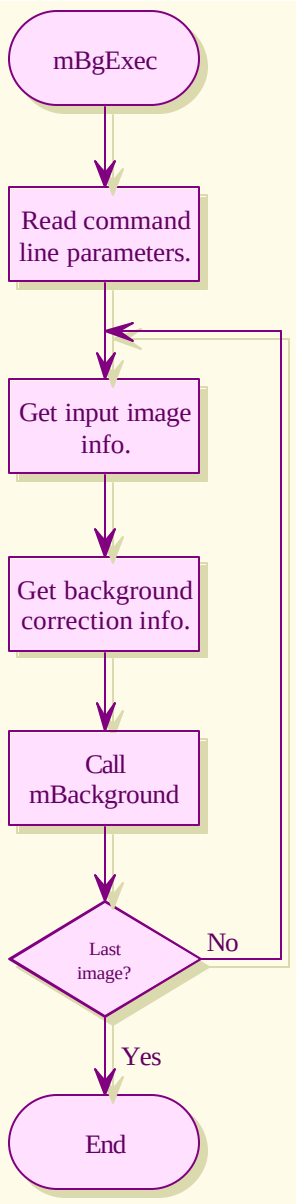
mBgModel



mBackground



mBgExec


mBgExec
Read command line parameters.
Get input image info.
Get background correction info.
Call mBackground
Last image?
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 [6]. The Appendix contains a complete list of successful and error return codes module-by-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 3 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 [7]. 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.

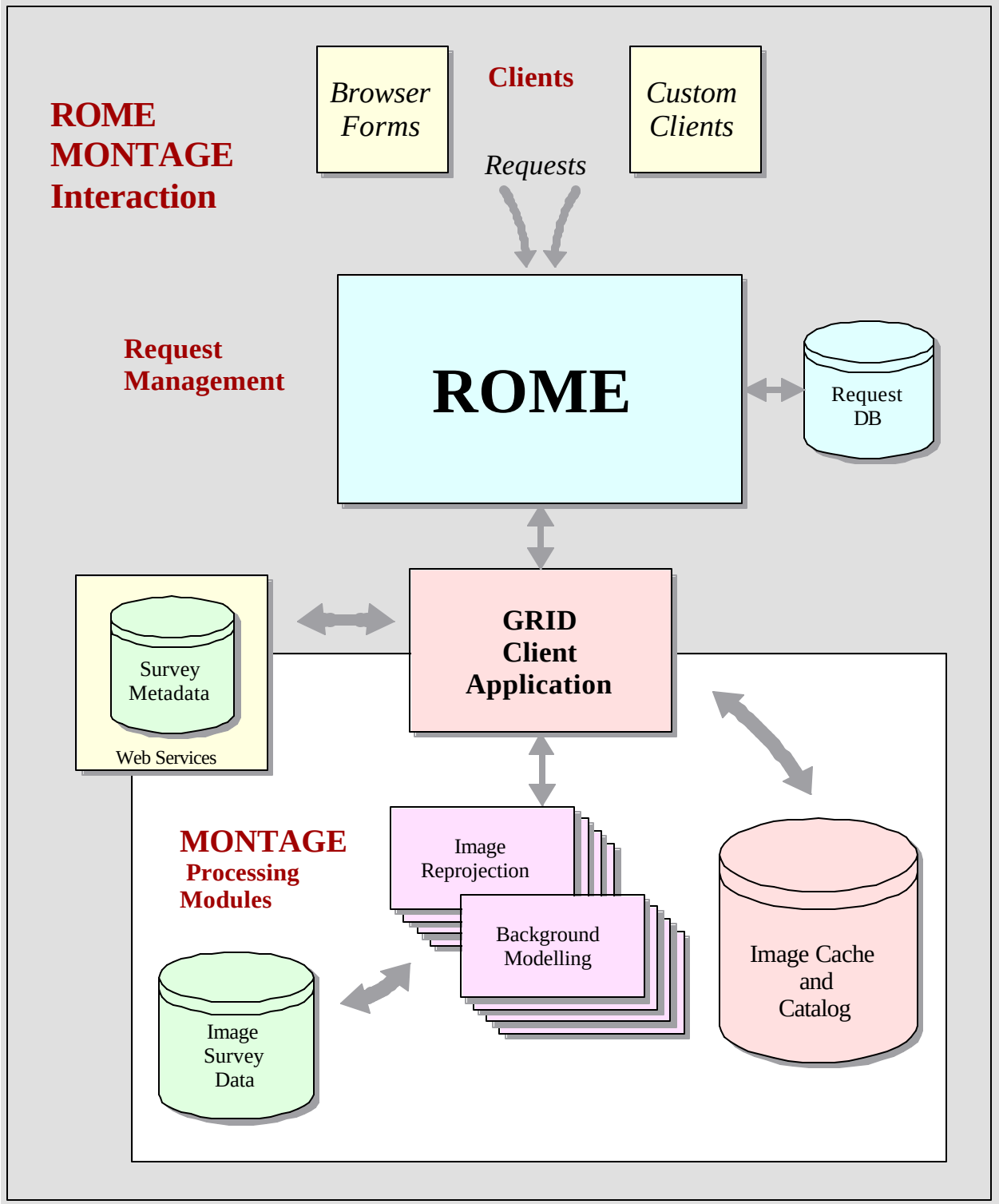


Figure 3: Montage Integrated in the NVO

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 [8]. 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) [9]. 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 [5]. 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 [2].

Use Case I - Science Analysis

The SIRTF First Look ancillary VLA image is a 2x2 degree radio image of a field that will be observed by SIRTF. As a field uncluttered by galactic radiation in SIRTF's 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) aboard the Space Infra Red Telescope Facility (http://sirtf.caltech.edu/SSC/MIPS/mips_intro.html) 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 Space Infra Red Telescope Facility (SIRTF) Infra Red Array Camera (IRAC) (http://sirtf.caltech.edu/SSC/IRAC/SSC_B4.html) 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] "Software Requirements Specification for Montage". Version 1.0 (May 31, 2002); http://montage.ipac.caltech.edu/projectdocs/Requirements.doc

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

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

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

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

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

[7] "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.

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

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

Acronyms

2MASS	Two Micron All Sky Survey
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
CACR	Center for Advanced Computing Research
CCD	Charge Coupled Device
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
HEASARC	High Energy Astrophysics Science ARChive
HDU	Header Data Unit
IDE	Interactive Development Environment
IPAC	Infrared Processing and Analysis Center
IPG	Information Power Grid
IRSA	InfraRed Science Archive
IRAC	InfraRed Array Camera
JPL	Jet Propulsion Laboratory
MIPS	Multiband Infrared Photometer for SIRTF
NVO	National Virtual Observatory
OASIS	On-Line Archive Science Information Services
SAO	Smithsonian Astrophysical Observatory
SDSC	San Diego Supercomputer Center
SDSS	Sloan Digital Sky Survey
SIRTF	Space Infrared Telescope Facility
SRB	Storage Resource Broker
STScI	Space Telescope Science Institute
TBD	To Be Decided
VLA	Very Large Array
WCS	World Coordinate System
XML	eXtensible Markup Language

Appendix: Montage Return Codes

A1. Montage Successful Completion Codes

All Montage program return a success message upon normal completion. In addition to the "OK" status, most also return useful information (in the case of mFitPlane data that needs to written to a file as later background modelling input).

The return 'types' are indicated by “C printf() syntax” (e.g. %d for an integer).

Table A1: Montage Successful Completion Codes

Module	Completion Code [1]
mAdd.c	[struct stat="OK", time=%d]
mBackground.c	[struct stat="OK"]
mBgExec.c	[struct stat="OK", count=%d, failed=%d]
mBgModel.c	[struct stat="OK"]
mDiff.c	[struct stat="OK"]
mDiffExec.c	[struct stat="OK", count=%d, failed=%d]
mFitExec.c	[struct stat="OK", count=%d, failed=%d]
mFitplane.c	[struct stat="OK", a=%-g, b=%-g, c=%-g, xmin=%-g, xmax=%-g, ymin=%-g, ymax=%-g, xcenter=%-g, ycenter=%-g, rms=%-g][2]
mImgtbl.c	[struct stat="OK", count=%d]
mMakeHdr.c	[struct stat="OK", count=%d]
mOverlaps.c	[struct stat="OK", count=%d]
mProjExec.c	[struct stat="OK", count=%d, failed=%d]

[1] Key to fields in return messages:

time	The execution time in seconds
count	The number of images (or overlap area) processed or identified.
failed	When processing image lists using Exec programs gives the number that failed for whatever reason (the reason is given by the return message of the exec-ed program).
noverlap	For mProjExec; the number of input images that did not overlap the region of interest.

[2] The return parameters for mFitplane represent the plane fit to the data:

plane value = a*(x-xcenter) + b*(y-ycenter) + c

where x and y are measured relative to xcenter, ycenter. The parameters xmin, xmax, ymin, ymax give the range of pixels with data values since images usually have some 'blank' (NaN) values. The data RMS after of the fit is given by the parameter rms.

A2. Montage Error Return Codes

Whenever a processing error occurs, all Montage services return error codes to stdout immediately before exiting . These errors trap conditions from incorrect input arguments to usage reminders to diagnostics on input data formats to I/O errors due to file permissions or disk space.

Table A2 gives a complete list of all error codes, module by module. Most messages are self explanatory. As with all such structured messages, all the text is returned on one line, though we have added new lines to improve readability where necessary.

All error returns contain a "msg" parameter; some include a FITs library error code (integer); and messages referring to files, often include the file name.

Table A2: Complete listing of Montage error codes module-by-module

mAdd

```
[struct stat="ERROR", msg="Usage: mAdd images.tbl out.fits
                              template.hdr [-d(ebug) level]"]
[struct stat="ERROR", msg="Need columns: cntr, fname in image list"]

[struct stat="ERROR", status=%d, msg="%s"]  (FITS library status code
and message)
[struct stat="ERROR", msg="Template file not found"]
[struct stat="ERROR", msg="All pixels are blank."]
```

mBackground

```
[struct stat="ERROR", msg="Usage: mBackground in.fits out.fits A B C
                              xcenter ycenter [-d(ebug) level]"]

[struct stat="ERROR", status=%d, msg="%s"]  (FITS library status code
and message)
```

mBgExec

```
[struct stat="ERROR", msg="Usage: mBgExec images.tbl corrections.tbl
                           corrdir [-d]"]
[struct stat="ERROR", msg="Need columns: cntr,fname in image list"]

[struct stat="ERROR", msg="Need columns: id,a,b,c,xcenter,ycenter
                           in corrections file"]
```

MBgModel

```
[struct stat="ERROR", msg="Usage: mBgModel images.tbl fits.tbl
 corrections.tbl [-iteration niter] [-d(ebug) level]"]

[struct stat="ERROR", msg="Failed to open output %s"]   (output file
name)
[struct stat="ERROR", msg="Need columns: cntr nl ns crpix1 crpix2 in
image info file"]
[struct stat="ERROR", msg="Need columns: plus minus a b c xmin xmax
                           ymin ymax xcenter ycenter"]
[struct stat="ERROR" msg="Singular Matrix-1"]  (Messages from matrix
inversion routines)
[struct stat="ERROR" msg="Singular Matrix-2"]
[struct stat="ERROR" msg="Allocation failure in ivector()"]
```

mDiff

```
[struct stat="ERROR", msg="Usage: mDiff in1.fits in2.fits out.fits
                           hdr.template [-d(ebug) level]"]

[struct stat="ERROR", status=%d, msg="%s"]  (FITS library status code
and message)
[struct stat="ERROR", msg="Template file not found."]
[struct stat="ERROR", msg="All pixels are blank."]
```

mDiffExec

```
[struct stat="ERROR", msg="Usage: mDiffExec diffs.tbl template.hdr
[diffdir] [-d]"]

[struct stat="ERROR", msg="Need columns: cntr1 cntr2 plus minus diff"]
```

mFitExec

```
[struct stat="ERROR", msg="Usage: mFitExec diffs.tbl fits.tbl [diffdir]
[-d]"]

[struct stat="ERROR", msg="Can't open output file."]
struct stat="ERROR", msg="Need columns: cntr1 cntr2 plus minus diff"]
```

mFitPlane

```
[struct stat="ERROR", msg="Usage: mFitPlane in.fits [-d(ebug) level]"]
[struct stat="ERROR", status=%d, msg="%s"]  (FITS library status code
and message)
[struct stat="ERROR" msg="Singular Matrix-1"]  (Messages from matrix
inversion routines)
[struct stat="ERROR" msg="Singular Matrix-2"]
```

mImbtbl

```
[struct stat="ERROR", msg="Usage: mImgtbl directory images.tbl [-
d(ebug)]"]
[struct stat="ERROR", msg="Can't open output table."]

[struct stat="ERROR", msg="Can't open copy table."]

[struct stat="ERROR", msg="Can't open tmp (in) table."]

[struct stat="ERROR", msg="Can't open tmp (out) table."]

[struct stat="ERROR", msg="Can't open final table."]
```

mOverlaps

```
[struct stat="ERROR", msg="Usage: mOverlaps images.tbl diffs.tbl [-
d(ebug) level]"]
[struct stat="ERROR", msg="Failed to open output %s"]
[struct stat="ERROR", msg="Need columns: cntr ctype1 ctype2 nl ns
crval1 crval2 crpix1 crpix2 cdelt1 cdelt2 crota2 fname (equinox
optional)"]

[struct stat="ERROR", msg="Bad WCS for image %d"]  (record number in
images.tbl)
```

MProjExec

```
[struct stat="ERROR", msg="Usage: mProjExec images.tbl template.hdr
                           projdir stats.tbl [-d]"]
[struct stat="ERROR", msg="Can't open output file."]
[struct stat="ERROR", msg="Need column fname in input"]
```

mProject

```
[struct stat="ERROR", msg="Usage: mProject in.fits out.fits
hdr.template [-drizzle factor][-d(ebug) level][-i(nrefpix) ypix xpix]
[-o(utrefpix) ypix xpix]"]
[struct stat="ERROR", msg="No overlap"]
[struct stat="ERROR", msg="Output wcsinit() failed."]
[struct stat="ERROR", msg="Input wcsinit() failed."]
[struct stat="ERROR", status=%d, msg="%s"]  (FITS library status code
and message)
[struct stat="ERROR", msg="Template file not found."]
[struct stat="ERROR", msg="All pixels are blank."]
```