The Ames Stereo Pipeline:

NASA's Open Source Automated Stereogrammetry Software

Version 2.6.2

Intelligent Robotics Group
NASA Ames Research Center
stereo-pipeline-owner@lists.nasa.gov

June 17, 2019

Credits

The Ames Stereo Pipeline (ASP) was developed by the Intelligent Robotics Group (IRG), in the Intelligent Systems Division at the National Aeronautics and Space Administration (NASA) Ames Research Center in Moffett Field, CA. It builds on over ten years of IRG experience developing surface reconstruction tools for terrestrial robotic field tests and planetary exploration.

Project Lead

- Dr. Ross Beyer (NASA/SETI Institute)

Development Team

- Oleg Alexandrov (NASA/Stinger-Ghaffarian Technologies)
- Scott McMichael (NASA/Stinger-Ghaffarian Technologies)

Former Developers

- Zachary Moratto (NASA/Stinger-Ghaffarian Technologies)
- Michael J. Broxton (NASA/Carnegie Mellon University)
- Dr. Ara Nefian (NASA/Carnegie Mellon University)
- Matthew Hancher (NASA)
- Mike Lundy (NASA/Stinger-Ghaffarian Technologies)
- Vinh To (NASA/Stinger-Ghaffarian Technologies)

Contributing Developer & Former IRG Terrain Reconstruction Lead

- Dr. Laurence Edwards (NASA)

A number of student interns have made significant contributions to this project over the years: Kyle Husmann (California Polytechnic State University), Sasha Aravkin (Washington State University), Aleksandr Segal (Stanford), Patrick Mihelich (Stanford University), Melissa Bunte (Arizona State University), Matthew Faulkner (Massachusetts Institute of Technology), Todd Templeton (UC Berkeley), Morgon Kanter (Bard College), Kerri Cahoy (Stanford University), and Ian Saxton (UC San Diego).

The open source Stereo Pipeline leverages stereo image processing work, past and present, led by Michael J. Broxton (NASA/CMU), Dr. Laurence Edwards (NASA), Eric Zbinden (formerly NASA/QSS Inc.), Dr. Michael Sims (NASA), and others in the Intelligent Systems Division at NASA Ames Research Center. It has benefited substantially from the contributions of Dr. Keith Nishihara (formerly NASA/Stanford), Randy Sargent (NASA/Carnegie Mellon University), Dr. Judd Bowman (formerly NASA/QSS Inc.), Clay Kunz (formerly NASA/QSS Inc.), and Dr. Matthew Deans (NASA).

Acknowledgments

The initial adaptation of Ames' stereo surface reconstruction tools to orbital imagers was a result of a NASA funded, industry led project to develop automated Digital elevation model (DEM) generation techniques for the Mars Global Surveyor (MGS) mission. Our work with that project's Principal Investigator, Dr. Michael Malin of Malin Space Science Systems (MSSS), and Co-Investigator, Dr. Laurence Edwards of NASA Ames, inspired the idea of making stereo surface reconstruction technology available and accessible to a broader community. We thank Dr. Malin and Dr. Edwards for providing the initial impetus that in no small way made this open source stereo pipeline possible, and we thank Dr. Michael Caplinger, Joe Fahle and others at MSSS for their help and technical assistance.

We'd also like to thank our friends and collaborators Dr. Randolph Kirk, Dr. Brent Archinal, Trent Hare, and Mark Rosiek of the United States Geological Survey's (USGS's) Astrogeology Science Center in Flagstaff, AZ, for their encouragement and willingness to share their experience and expertise by answering many of our technical questions. We also thank them for their ongoing support and efforts to help us evaluate our work. Thanks also to the USGS Integrated Software for Imagers and Spectrometers (ISIS) team, especially Jeff Anderson and Kris Becker, for their help in integrating stereo pipeline with the USGS ISIS software package.

Thanks go also to Dr. Mark Robinson, Jacob Danton, Ernest Bowman-Cisneros, Dr. Sam Laurence, and Melissa Bunte at Arizona State University for their help in adapting the Ames Stereo Pipeline to lunar data sets including the Apollo Metric Camera.

We'd also like to thank David Shean, Dr. Ben Smith, and Dr. Ian Joughin of the Applied Physics Laboratory at the University of Washington for providing design direction for adapting Ames Stereo Pipeline to Earth sciences.

Finally, we thank Dr. Ara Nefian, and Dr. Laurence Edwards for their contributions to this software, and Dr. Terry Fong (IRG Group Lead) for his management and support of the open source and public software release process.

Portions of this software were developed with support from the following NASA Science Mission Directorate (SMD) and Exploration Systems Mission Directorate (ESMD) funding sources: the Mars Technology Program, the Mars Critical Data Products Initiative, the Mars Reconnaissance Orbiter mission, the Applied Information Systems Research program grant #06-AISRP06-0142, the Lunar Advanced Science and Exploration Research (LASER) program grants #07-LASER07-0148 and #11-LASER11-0112, the ESMD Lunar Mapping and Modeling Program (LMMP), and the SMD Cryosphere Program.

Any opinions, findings, and conclusions or recommendations expressed in this documentation are those of the authors and do not necessarily reflect the views of the National Aeronautics and Space Administration.

Contents

Chapter 1

Introduction

The NASA Ames Stereo Pipeline (ASP) is a suite of free and open source automated geodesy and stereogrammetry tools designed for processing stereo images captured from satellites (around Earth and other planets), robotic rovers, aerial cameras, and historical images, with and without accurate camera pose information. It produces cartographic products, including digital elevation models (DEMs), ortho-projected images, 3D models, and bundle-adjusted networks of cameras. ASP's data products are suitable for science analysis, mission planning, and public outreach.

1.1 Background

The Intelligent Robotics Group (IRG) at the NASA Ames Research Center has been developing 3D surface reconstruction and visualization capabilities for planetary exploration for more than a decade. First demonstrated during the Mars Pathfinder Mission, the IRG has delivered tools providing these capabilities to the science operations teams of the Mars Polar Lander (MPL) mission, the Mars Exploration Rover (MER) mission, the Mars Reconnaissance Orbiter (MRO) mission, and most recently the Lunar Reconnaissance Orbiter (LRO) mission. A critical component technology enabling this work is the Ames Stereo Pipeline (ASP). The Stereo Pipeline generates high quality, dense, texture-mapped 3D surface models from stereo image pairs. In addition, ASP provides tools to perform many other cartography tasks including map projection, point cloud and DEM registration, automatic registration of cameras, data format conversion, and data visualization.

Although initially developed for ground control and scientific visualization applications, the Stereo Pipeline has evolved to address orbital stereogrammetry and cartographic applications. In particular, long-range mission planning requires detailed knowledge of planetary topography, and high resolution topography is often derived from stereo pairs captured from orbit. Orbital mapping satellites are sent as precursors to planetary bodies in advance of landers and rovers. They return a wealth of images and other data that helps mission planners and scientists identify areas worthy of more detailed study. Topographic information often plays a central role in this planning and analysis process.

Our recent development of the Stereo Pipeline coincides with a period of time when NASA orbital mapping missions are returning orders of magnitude more data than ever before. Data volumes from the Mars and Lunar Reconnaissance Orbiter missions now measure in the tens of terabytes. There is growing consensus that existing processing techniques, which are still extremely human intensive and expensive, are no longer adequate to address the data processing needs of NASA and the Planetary Science community. To pick an example of particular relevance, the High Resolution Imaging Science Experiment (HiRISE) instrument has captured a few thousand stereo pairs. Of these, only about two hundred stereo pairs have been processed to date; mostly on human-operated, high-end photogrammetric workstations. It is clear that much more value

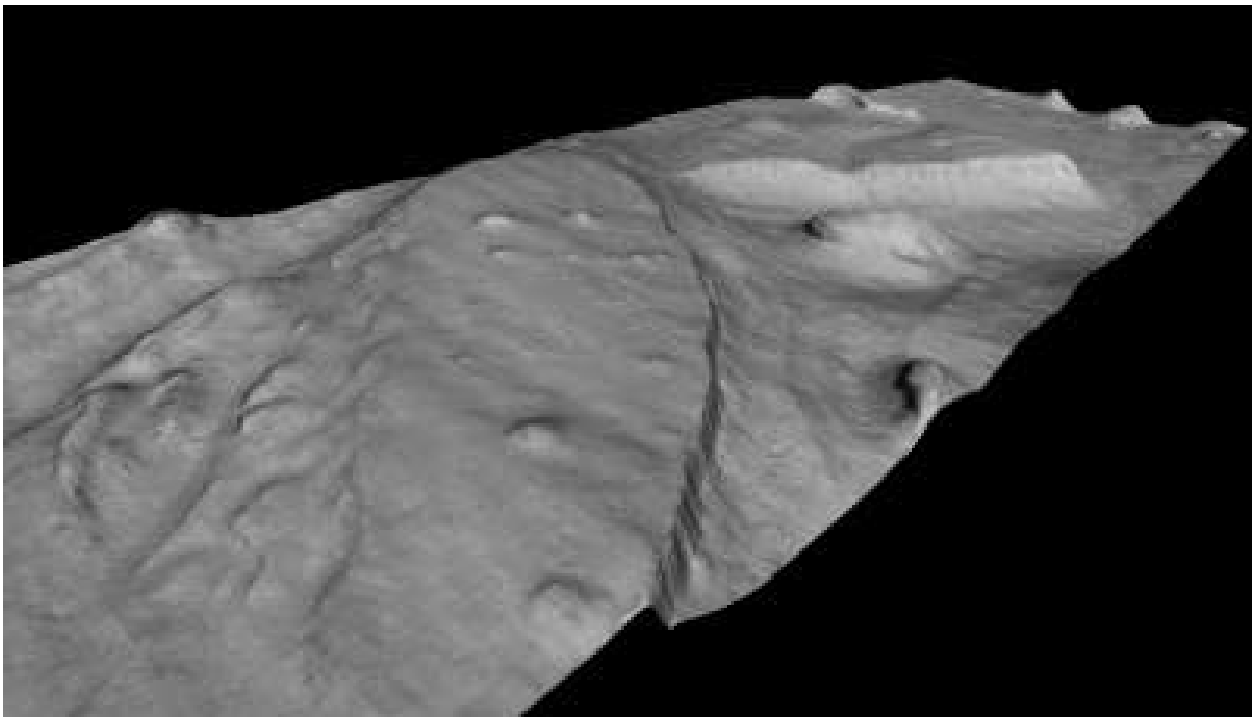


Figure 1.1: This 3D model was generated from a Mars Orbiter Camera (MOC) image pair M01/00115 and E02/01461 (34.66N, 141.29E). The complete stereo reconstruction process takes approximately thirty minutes on a 3.0 GHz workstation for input images of this size (1024 × 8064 pixels). This model, shown here without vertical exaggeration, is roughly 2 km wide in the cross-track dimension.

could be extracted from this valuable raw data if a more streamlined, efficient process could be developed.

The Stereo Pipeline was designed to address this very need. By applying recent advances in computer vision, we have created an *automated* process that is capable of generating high quality Digital elevation models (DEMs) with minimal human intervention. Users of the Stereo Pipeline can expect to spend some time picking a handful of settings when they first start processing a new type of image, but once this is done, the Stereo Pipeline can be used to process tens, hundreds, or even thousands of stereo pairs without further adjustment. With the release of this software, we hope to encourage the adoption of this tool chain at institutions that run and support these remote sensing missions. Over time, we hope to see this tool incorporated into ground data processing systems alongside other automated image processing pipelines. As this tool continues to mature, we believe that it will be capable of producing digital elevation models of exceptional quality without any human intervention.

1.2 Human vs. Computer: When to Choose Automation?

When is it appropriate to choose automated stereo mapping over the use of a conventional, human-operated photogrammetric workstation? This is a philosophical question with an answer that is likely to evolve over the coming years as automated data processing technologies become more robust and widely adopted. For now, our opinion is that you should *always* rely on human-guided, manual data processing techniques for producing mission critical data products for missions where human lives or considerable capital resources are at risk. In particular, maps for landing site analysis and precision landing absolutely require the benefit of an expert human operator to eliminate obvious errors in the DEMs, and also to guarantee that the proper procedures have been followed to correct satellite telemetry errors so that the data have the best possible geodetic control.

When it comes to using DEMs for scientific analysis, both techniques have their merits. Human-guided stereo reconstruction produces DEMs of unparalleled quality that benefit from the intuition and experience of an expert. The process of building and validating these DEMs is well-established and accepted in the scientific community.

However, only a limited number of DEMs can be processed to this level of quality. For the rest, automated stereo processing can be used to produce DEMs at a fraction of the cost. The results are not necessarily less accurate than those produced by the human operator, but they will not benefit from the same level of scrutiny and quality control. As such, users of these DEMs must be able to identify potential issues, and be on the lookout for errors that may result from the improper use of these tools.

We recommend that all users of the Stereo Pipeline take the time to thoroughly read this documentation and build an understanding of how stereo reconstruction and bundle adjustment can be best used together to produce high quality results. You are welcome to contact us if you have any questions (section 1.4).

1.3 Software Foundations

1.3.1 NASA Vision Workbench

The Stereo Pipeline is built upon the Vision Workbench software which is a general purpose image processing and computer vision library also developed by the IRG. Some of the tools discussed in this document are actually Vision Workbench programs, and any distribution of the Stereo Pipeline requires the Vision Workbench. This distinction is important only if compiling this software.

1.3.2 The USGS Integrated Software for Imagers and Spectrometers

For processing non-terrestrial NASA satellite images, Stereo Pipeline must be installed alongside a copy of United States Geological Survey (USGS) Integrated Software for Imagers and Spectrometers (ISIS). ISIS is however not required for processing Digital Globe images of Earth.

ISIS is widely used in the planetary science community for processing raw spacecraft images into high level data products of scientific interest such as map-projected and mosaicked images [6, 40, 150]. We chose ISIS because (1) it is widely adopted by the planetary science community, (2) it contains the authoritative collection of geometric camera models for planetary remote sensing instruments, and (3) it is open source software that is easy to leverage.

By installing the Stereo Pipeline, you will be adding an advanced stereo image processing capability that can be used in your existing ISIS workflow. The Stereo Pipeline supports the ISIS "cube" (`.cub`) file format, and can make use of the ISIS camera models and ancillary information (i.e. SPICE kernels) for imagers on many NASA spacecraft. The use of this single standardized set of camera models ensures consistency between products generated in the Stereo Pipeline and those generated by ISIS. Also by leveraging ISIS camera models, the Stereo Pipeline can process stereo pairs captured by just about any NASA mission.

1.4 Getting Help and Reporting Bugs

All bugs, feature requests, and general discussion should be posted on the ASP support forum:

https://groups.google.com/forum/#!forum/ames-stereo-pipeline-support

To contact the developers and project manager directly, send an email to:

stereo-pipeline-owner@lists.nasa.gov

When you submit a bug report, it may be helpful to attach the logs output by `stereo` and other tools (section 2.3.2).

1.5 Typographical Conventions

Names of programs that are meant to be run on the command line are written in a constant-width font, like the `stereo` program, as are options to those programs.

An indented line of constant-width text can be typed into your terminal, these lines will either begin with a '`>`' to denote a regular shell, or with '`ISIS`' which denotes an ISIS-enabled shell (which means you have to set the `ISISROOT` environment variable and sourced the appropriate ISIS 3 Startup script, as detailed in the ISIS 3 instructions).

```
> ls

ISIS 3> pds2isis
```

Italicized constant-width text denotes an option or argument that a user will need to supply. For example, '`stereo E0201461.map.cub M0100115.map.cub out`' is specific, but '`stereo` *`left-image right-image`* `out`' indicates that *`left-image`* and *`right-image`* are not the names of specific files, but dummy parameters which need to be replaced with actual file names.

Square brackets denote optional options or values to a command, and items separated by a vertical bar are either aliases for each other, or different, specific options. Default arguments are prefixed by an equals sign within parentheses, and line continuation with a backslash:

```
point2dem [--help|-h] [-r moon|mars] [-s float(=0)] \
          [-o output-filename] pointcloud-PC.tif
```

The above indicates a run of the `point2dem` program. The only argument that it requires is a point cloud file, which is produced by the `stereo` program and ends in `-PC.tif`, although its prefix could be anything (hence the italics for that part). Everything else is in square brackets indicating that they are optional.

Here, `--help` and `-h` refer to the same thing. Similarly, the argument to the `-r` option must be either `moon` or `mars`. The `-s` option takes a floating point value as its argument, and has a default value of zero. The `-o` option takes a filename that will be used as the output DEM.

Although there are two lines of constant-width text, the backslash at the end of the first line indicates that the command continues on the second line. You can either type everything into one long line on your own terminal, or use the backslash character and a return to continue typing on a second line in your terminal.

1.6 Referencing the Ames Stereo Pipeline in Your Work

In general, please use this reference for the Ames Stereo Pipeline:

Beyer, Ross A., Oleg Alexandrov, and Scott McMichael. 2018. The Ames Stereo Pipeline: NASA's open source software for deriving and processing terrain data. *Earth and Space Science*, **5**. https://doi.org/10.1029/2018EA000409.

If you are using ASP for application to Earth images, or need a reference which details the quality of output, then we suggest also referencing:

Shean, D. E., O. Alexandrov, Z. Moratto, B. E. Smith, I. R. Joughin, C. C. Porter, Morin, P. J. 2016. An automated, open-source pipeline for mass production of digital elevation models (DEMs) from very high-resolution commercial stereo satellite imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*. **116**.

In addition to using the references above, in order to help you better cite the specific version of ASP that you are using in a work, as of ASP version 2.6.0, we have started using Zenodo to create digital object identifiers (DOIs) for each ASP release. For example, the DOI for version 2.6.2 is 10.5281/zenodo.3247734, and you can cite it like this:

Beyer, Ross A., Oleg Alexandrov, and Scott McMichael. 2019. NeoGeographyToolkit/StereoPipeline: Ames Stereo Pipeline version 2.6.2. *Zenodo*. DOI: 10.5281/zenodo.3247734.

Of course, every new release of ASP will have its own unique DOI, and this link should always point to the latest DOI for ASP.

If you publish a paper using ASP, please let us know. We'll cite your work in this document, in Appendix E.

1.7 Warnings to Users of the Ames Stereo Pipeline

Ames Stereo Pipeline is a **research** product. There may be bugs or incomplete features. We reserve the ability to change the API and command line options of the tools we provide. Although we hope you will find this release helpful, you use it at your own risk. Please check each release's **NEWS** file to see a summary of our recent changes.

While we are confident that the algorithms used by this software are robust, the Ames Stereo Pipeline has a lot of adjustable parameters, and even experienced operators can produce poor results. We *strongly recommend* that if you have any concerns about the products that you (or others) create with this software, please just get in contact with us. We can help you figure out either how to make the product better, or help you accurately describe the limitations of the data or the data products, so that you can use it to confidently make new and wonderful discoveries.

Part I

Getting Started

Chapter 2

Installation

2.1 Binary Installation

This is the recommended method. Only the Stereo Pipeline binaries are required. ISIS is required only for users who wish to process NASA non-terrestrial imagery. A full ISIS installation is not required for operation of Stereo Pipeline programs (only the ISIS data directory is needed), but is required for certain preprocessing steps before Stereo Pipeline programs are run for planetary data. If you only want to process terrestrial Digital Globe imagery, skip to section 2.1.2.

Stereo Pipeline Tarball

The main Stereo Pipeline page is http://irg.arc.nasa.gov/ngt/stereo. Download the option that matches the platform you wish to use. The recommended, but optional, ISIS version is listed next to the name.

USGS ISIS

If you are working with non-terrestrial imagery, you will need to install ISIS so that you can perform preprocessing such as radiometric calibration and ephemeris attachment. The ISIS installation guide is at http://isis.astrogeology.usgs.gov/documents/InstallGuide. You must use their binaries as-is; if you need to recompile, you can follow the *Source Installation* guide for the Stereo Pipeline in Section 2.2. Note also that the USGS provides only the current version of ISIS and the previous version (denoted with a '`_OLD`' suffix) via their `rsync` service. If the current version is newer than the version of ISIS that the Stereo Pipeline is compiled against, be assured that we're working on rolling out a new version. However, since Stereo Pipeline has its own self-contained version of ISIS's libraries built internally, you should be able to use a newer version of ISIS with the now dated version of ASP. This is assuming no major changes have taken place in the data formats or camera models by USGS. At the very least, you should be able to rsync the previous version of ISIS if a break is found. To do so, view the listing of modules that is provided via the '`rsync isisdist.astrogeology.usgs.gov::`' command. You should see several modules listed with the '`_OLD`' suffix. Select the one that is appropriate for your system, and `rsync` according to the instructions.

In closing, running the Stereo Pipeline executables only requires that you have downloaded the ISIS secondary data and have appropriately set the `ISIS3DATA` environment variable. This is normally performed for the user by ISIS startup script, `$ISISROOT/scripts/isis3Startup.sh`.

2.1.1 Quick Start for ISIS Users

Fetch Stereo Pipeline

Download the Stereo Pipeline from http://irg.arc.nasa.gov/ngt/stereo.

Fetch ISIS Binaries
As detailed at http://isis.astrogeology.usgs.gov/documents/InstallGuide.

Fetch ISIS Data
As detailed at http://isis.astrogeology.usgs.gov/documents/InstallGuide.

Untar Stereo Pipeline
`tar xzvf StereoPipeline-`*`VERSION-ARCH-OS`*`.tar.gz`

Add Stereo Pipeline to Path (optional)
bash: `export PATH="`*`/path/to/StereoPipeline`*`/bin:${PATH}"`
csh: `setenv PATH "`*`/path/to/StereoPipeline`*`/bin:${PATH}"`

Set Up ISIS
bash:
```
export ISISROOT=/path/to/isisroot
source $ISISROOT/scripts/isis3Startup.sh
```
csh:
```
setenv ISISROOT /path/to/isisroot
source $ISISROOT/scripts/isis3Startup.csh
```

Try It Out
See Chapter 3 for an example.

2.1.2 Quick Start for Digital Globe Users

Fetch Stereo Pipeline
Download the Stereo Pipeline from http://irg.arc.nasa.gov/ngt/stereo.

Untar Stereo Pipeline
`tar xvfz StereoPipeline-`*`VERSION-ARCH-OS`*`.tar.gz`

Try It Out
Processing Earth imagery is described in the data processing tutorial in chapter 4.

2.1.3 Quick Start for Aerial and Historical Imagery

Fetch the software as before. Processing imagery without accurate camera pose information is described in chapter 9.

2.1.4 Common Errors

Here are some errors you might see, and what it could mean. Treat these as templates for problems. In practice, the error messages might be slightly different.

```
**I/O ERROR** Unable to open [$ISIS3DATA/Some/Path/Here].
Stereo step 0: Preprocessing failed
```

You need to set up your ISIS environment or manually set the correct location for `ISIS3DATA`.

```
point2mesh stereo-output-PC.tif stereo-output-L.tif
[...]
99%  Vertices:   [************************************************************] Complete!
       > size: 82212 vertices
Drawing Triangle Strips
Attaching Texture Data
zsh: bus error  point2mesh stereo-output-PC.tif stereo-output-L.tif
```

The source of this problem is an old version of OpenSceneGraph in your library path. Check your `LD_LIBRARY_PATH` (for Linux), `DYLD_LIBRARY_PATH` (for OSX), or your `DYLD_FALLBACK_LIBRARY_PATH` (for OSX) to see if you have an old version listed, and remove it from the path if that is the case. It is not necessary to remove the old versions from your computer, you just need to remove the reference to them from your library path.

```
bash: stereo: command not found
```

You need to add the `bin` directory of your deployed Stereo Pipeline installation to the environmental variable `PATH`.

2.2 Installation from Source

This method is for advanced users. You will need to fetch the Stereo Pipeline source code from GitHub at https://github.com/NeoGeographyToolkit/StereoPipeline and then follow the instructions specified in `INSTALLGUIDE`.

2.3 Settings Optimization

Finally, the last thing to be done for Stereo Pipeline is to setup up Vision Workbench's render and logging settings. This step is optional, but for best performance some thought should be applied here.

Vision Workbench is a multi-threaded image processing library used by Stereo Pipeline. The settings by which Vision Workbench processes is configurable by having a `.vwrc` file hidden in your home directory. Below is an example.

```
# This is an example VW configuration file. Save this file to ~/.vwrc
# to adjust the VW log settings, even if the program is already running.

# General settings
[general]
default_num_threads = 16
write_pool_size = 40
system_cache_size = 1024000000 # ~ 1 GB

# The following integers are associated with the log levels throughout the
# Vision Workbench.  Use these in the log rules below.
#
#    ErrorMessage = 0
#    WarningMessage = 10
#    InfoMessage = 20
#    DebugMessage = 30
#    VerboseDebugMessage = 40
#    EveryMessage = 100
#
# You can create a new log file or adjust the settings
# for the console log:
#   logfile <filename>
#       - or -
#   logfile console

# Once you have created a logfile (or selected the console), you can
# add log rules using the following syntax. (Note that you can use
# wildcard characters '*' to catch all log_levels for a given
# log_namespace, or vice versa.)

# <log_level> <log_namespace>

# Below are examples of using the log settings.

# Turn on various logging levels for several subsystems, with the
# output going to the console (standard output).
[logfile console]
# Turn on error and warning messages for the thread subsystem.
10 = thread
# Turn on error, warning, and info messages for the asp subsystem.
20 = asp
# Turn on error, warning, info, and debug messages for the stereo subsystem.
30 = stereo
# Turn on every single message for the cache subsystem (this will be
# extremely verbose and is not recommended).
# 100 = cache
# Turn off all progress bars to the console (not recommended).
# 0 = *.progress

# Turn on logging of error and warning messages to a file for the
# stereo subsystem. Warning: This file will be always appended to, so
# it should be deleted periodically.
# [logfile /tmp/vw_log.txt]
# 10 = stereo
```

There are a lot of possible options that can be implemented in the above example. Let's cover the most important options and the concerns the user should have when selecting a value.

2.3.1 Performance Settings

default_num_threads (default=2)
This sets the maximum number of threads that can be used for rendering. When stereo's `subpixel_rfne` is running you'll probably notice 10 threads are running when you have `default_num_threads` set to 8. This is not an error, you are seeing 8 threads being used for rendering, 1 thread for holding `main()`'s execution, and finally 1 optional thread acting as the interface to the file driver.

It is usually best to set this parameter equal to the number of processors on your system. Be sure to include the number of logical processors in your arithmetic if your system supports hyper-threading. Adding more threads for rasterization increases the memory demands of Stereo Pipeline. If your system is memory limited, it might be best to lower the `default_num_threads` option.

write_pool_size (default=21)
The `write_pool_size` option represents the max waiting pool size of tiles waiting to be written to disk. Most file formats do not allow tiles to be written arbitrarily out of order. Most however will let rows of tiles to be written out of order, while tiles inside a row must be written in order. Because of the previous constraint, after a tile is rasterized it might spend some time waiting in the 'write pool' before it can be written to disk. If the 'write pool' fills up, only the next tile in order can be rasterized. That makes Stereo Pipeline perform like it is only using a single processor.

Increasing the `write_pool_size` makes Stereo Pipeline more able to use all processing cores in the system. Having this value too large can mean excessive use of memory as it must keep more portions of the image around in memory while they wait to be written. This number should be larger than the number of threads, perhaps by about 20.

system_cache_size (default=805306368)
Accessing a file from the hard drive can be very slow. It is especially bad if an application needs to make multiple passes over an input file. To increase performance, Vision Workbench will usually leave an input file stored in memory for quick access. This file storage is known as the 'system cache' and its max size is dictated by `system_cache_size`. The default value is 768 MB.

Setting this value too high can cause your application to crash. It is usually recommend to keep this value around 1/4 of the maximum available memory on the system. The units of this property is in bytes.

The recommendations for these values are based on use of the block matching algorithm in ASP. When using memory intensive algorithms such as SGM you may wish to lower some of these values (such as the cache size) to leave more memory available for the algorithm to use.

2.3.2 Logging Settings

The messages displayed in the console by Stereo Pipeline are grouped into several namespaces, and by level of verbosity. An example of customizing Stereo Pipeline's output is given in the `.vwrc` file shown above.

Several of the tools in Stereo Pipeline, including `stereo`, automatically append the information displayed in the console to a log file in the current output directory. These logs contain in addition some data about your system and settings, which may be helpful in resolving problems with the tools.

It is also possible to specify a global log file to which all tools will append to, as illustrated in `.vwrc`.

Chapter 3

Tutorial: Processing Mars Orbiter Camera Imagery

3.1 Quick Start

The Stereo Pipeline package contains GUI and command-line programs that convert a stereo pair in the ISIS `.cub` format into a 3D "point cloud" image (its format is described in section C). This is an intermediate format that can be passed along to one of several programs that convert a point cloud into a mesh for 3D viewing, a gridded digital elevation model (DEM) for GIS purposes, or a LAS/LAZ point cloud.

There are a number of ways to fine-tune parameters and analyze the results, but ultimately this software suite takes images and builds models in a mostly automatic way. To create a point cloud file, you simply pass two image files to the `stereo` command:

```
ISIS 3> stereo left_input_image.cub right_input_image.cub stereo-output
```

Alternatively, the `stereo_gui` frontend can be invoked, with the same options, as described in section A.2. This tool makes it possible to select small clips on which to run `stereo`.

The string `stereo-output` is an arbitrary output prefix, it is used when generating names for `stereo` output files. For example, it can be set to `results/output`, in which case all output files will be in the `results` directory and start with the prefix `output`. See section 5.1 for a more detailed discussion.

You can then make a visualizable mesh or a DEM file with the following commands (the *`stereo-output`*`-PC.tif` and *`stereo-output`*`-L.tif` files are created by the `stereo` program above):

```
ISIS 3> point2mesh stereo-output-PC.tif stereo-output-L.tif
ISIS 3> point2dem  stereo-output-PC.tif
```

More details are provided in section 5.2.

3.2 Preparing the Data

The data set that is used in the tutorial and examples below is a pair of Mars Orbital Camera (MOC) [86, 85] images whose Planetary Data System (PDS) Product IDs are M01/00115 and E02/01461. This data can be downloaded from the PDS directly, or they can be found in the `examples/MOC` directory of your Stereo Pipeline distribution.

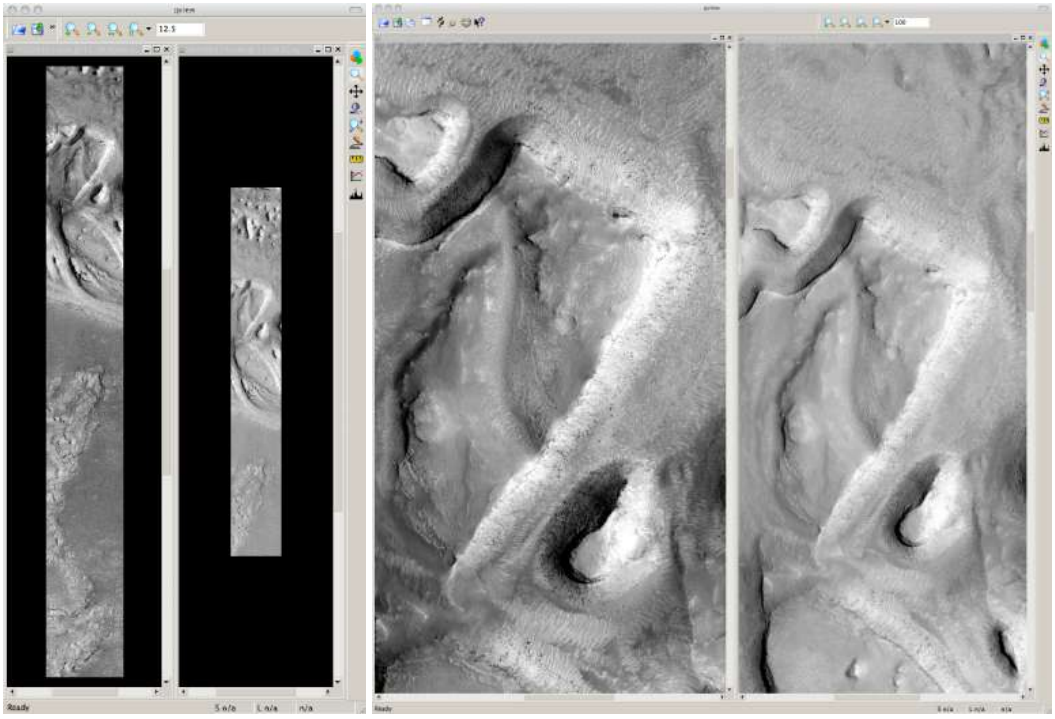


Figure 3.1: This figure shows `E0201461.cub` and `M0100115.cub` open in ISIS's qview program. The view on the left shows their full extents at the same zoom level, showing how they have different ground scales. The view on the right shows both images zoomed in on the same feature.

3.2.1 Loading and Calibrating Images using ISIS

These raw PDS images (`M0100115.imq` and `E0201461.imq`) need to be imported into the ISIS environment and radiometrically calibrated. You will need to be in an ISIS environment (have set the `ISISROOT` environment variable and sourced the appropriate ISIS 3 startup script, as detailed in the ISIS 3 instructions; we will denote this state with the '`ISIS 3>`' prompt). Then you can use the `mocproc` program, as follows:

```
ISIS 3> mocproc from=M0100115.imq to=M0100115.cub Mapping=NO
ISIS 3> mocproc from=E0201461.imq to=E0201461.cub Mapping=NO
```

There are also `Ingestion` and `Calibration` parameters whose defaults are '`YES`' which will bring the image into the ISIS format and perform radiometric calibration. By setting the `Mapping` parameter to '`NO`', the resultant file will be an ISIS cube file that is calibrated, but not map-projected. Note that while we have not explicitly run `spiceinit`, the Ingestion portion of `mocproc` quietly ran `spiceinit` for you (you'll find the record of it in the ISIS Session Log, usually written out to a file named `print.prt`). Refer to Figure 3.1 to see the results at this stage of processing.

Datasets for other type of cameras or other planets can be pre-processed similarly, using the ISIS tools specific to them.

3.2.2 Aligning Images

Once the .cub files are obtained, it is possible to run stereo right away, as

```
ISIS 3> stereo E0201461.cub M0100115.cub    \
          --alignment-method affineepipolar \
          -s stereo.default.example results/output
```

In this case, the first thing `stereo` does is to internally align (or rectify the images), which helps with finding stereo matches. Here we have used `affineepipolar` alignment. Another option is to use `homography` alignment, as described in section 5.1.2.

Alternatively, the images can be aligned externally, by map-projecting them in ISIS. External alignment can sometimes give better results than the simple internal alignment described earlier, especially if the images are taken from very different perspectives, or if the curvature of the planet/body being imaged is non-negligible.

We will now describe how to do this alignment, but we also provide the `cam2map4stereo.py` program (page 179) which performs this work automatically for you. (Also note that ASP has its own internal way of map-projecting images, which we believe is preferable. That approach is described in section 5.1.7.)

The ISIS `cam2map` program will map-project these images:

```
ISIS 3> cam2map from=M0100115.cub to=M0100115.map.cub
ISIS 3> cam2map from=E0201461.cub to=E0201461.map.cub map=M0100115.map.cub matchmap=true
```

Notice the order in which the images were run through `cam2map`. The first projection with `M0100115.cub` produced a map-projected image centered on the center of that image. The projection of `E0201461.cub` used the `map=` parameter to indicate that `cam2map` should use the same map projection parameters as those of `M0100115.map.cub` (including center of projection, map extents, map scale, etc.) in creating the projected image. By map-projecting the image with the worse resolution first, and then matching to that, we ensure two things: (1) that the second image is summed or scaled down instead of being magnified up, and (2) that we are minimizing the file sizes to make processing in the Stereo Pipeline more efficient.

Technically, the same end result could be achieved by using the `mocproc` program alone, and using its `map= M0100115.map.cub` option for the run of `mocproc` on `E0201461.cub` (it behaves identically to `cam2map`). However, this would not allow for determining which of the two images had the worse resolution and extracting their minimum intersecting bounding box (see below). Furthermore, if you choose to conduct bundle adjustment (see Chapter 8, page 61) as a pre-processing step, you would do so between `mocproc` (as run above) and `cam2map`.

The above procedure is in the case of two images which cover similar real estate on the ground. If you have a pair of images where one image has a footprint on the ground that is much larger than the other, only the area that is common to both (the intersection of their areas) should be kept to perform correlation (since non-overlapping regions don't contribute to the stereo solution). If the image with the larger footprint size also happens to be the image with the better resolution (i.e. the image run through `cam2map` second with the `map=` parameter), then the above `cam2map` procedure with `matchmap=true` will take care of it just fine. Otherwise you'll need to figure out the latitude and longitude boundaries of the intersection boundary (with the ISIS `camrange` program). Then use that smaller boundary as the arguments to the `MINLAT`, `MAXLAT`, `MINLON`, and `MAXLON` parameters of the first run of `cam2map`. So in the above example, after `mocproc` with `Mapping= NO` you'd do this:

```
ISIS 3> camrange from=M0100115.cub
         ... lots of camrange output omitted ...
Group = UniversalGroundRange
  LatitudeType       = Planetocentric
  LongitudeDirection = PositiveEast
  LongitudeDomain    = 360
  MinimumLatitude    = 34.079818835324
  MaximumLatitude    = 34.436797628116
  MinimumLongitude   = 141.50666207418
```

```
  MaximumLongitude   = 141.62534719278
End_Group
         ... more output of camrange omitted ...

ISIS 3> camrange from=E0201461.cub
         ... lots of camrange output omitted ...
Group = UniversalGroundRange
  LatitudeType       = Planetocentric
  LongitudeDirection = PositiveEast
  LongitudeDomain    = 360
  MinimumLatitude    = 34.103893080982
  MaximumLatitude    = 34.547719435156
  MinimumLongitude   = 141.48853937384
  MaximumLongitude   = 141.62919740048
End_Group
         ... more output of camrange omitted ...
```

Now compare the boundaries of the two above and determine the intersection to use as the boundaries for `cam2map`:

```
ISIS 3> cam2map from=M0100115.cub to=M0100115.map.cub DEFAULTRANGE=CAMERA \
                MINLAT=34.10 MAXLAT=34.44 MINLON=141.50 MAXLON=141.63
ISIS 3> cam2map from=E0201461.cub to=E0201461.map.cub map=M0100115.map.cub matchmap=true
```

You only have to do the boundaries explicitly for the first run of `cam2map`, because the second one uses the `map=` parameter to mimic the map-projection of the first. These two images are not radically different in spatial coverage, so this is not really necessary for these images, it is just an example.

Again, unless you are doing something complicated, using the `cam2map4stereo.py` program (page 179) will take care of all these steps for you.

At this stage we can run the stereo program with map-projected images:

```
ISIS 3> stereo E0201461.map.cub M0100115.map.cub --alignment-method none \
          -s stereo.default.example results/output
```

Here we have used `alignment-method none` since `cam2map4stereo.py` brought the two images into the same perspective and using the same resolution. If you invoke `cam2map` independently on the two images, without `matchmap=true`, their resolutions may differ, and using an alignment method rather than `none` to correct for that is still necessary.

Now you may skip to chapter 5 which will discuss the `stereo` program in more detail and the other tools in ASP.

Chapter 4

Tutorial: Processing Earth Digital Globe Imagery

In this chapter we will focus on how to process Earth imagery, or more specifically Digital Globe data. This is different from our previous chapter in that at no point will we be using ISIS utilities. This is because ISIS only supports NASA instruments, while most Earth imagery comes from commercial providers.

In addition to Digital Globe's satellites, ASP supports any Earth imagery that uses the RPC camera model format. How to process such data is described in section 11.12, although following this tutorial may still be insightful even if your data is not from Digital Globe.

Digital Globe provides imagery from Quick Bird and the three World View satellites. These are the hardest images to process with Ames Stereo Pipeline because they are exceedingly large, much larger than HiRISE imagery (the GUI interface can be used to run stereo on just a portion of the images). There is also a wide range of terrain challenges and atmospheric effects that can confuse ASP. Trees are particularly difficult for us since their texture is nearly nadir and perpendicular to our line of sight. It is important to know that the driving force behind our support for Digital Globe imagery is to create models of ice and bare rock. That is the type of imagery that we have tested with and have focused on. If we can make models of wooded or urban areas, that is a bonus, but we can't provide any advice for how to perform or improve the results if you choose to use ASP in that way.

ASP can only process Level 1B satellite imagery, and cannot process Digital Globe's aerial images.

The camera information for Digital Globe images is contained in an XML file for each image. In addition to the exact linear camera model, the XML file also has its RPC approximation. In this chapter we will focus only on processing data using the linear camera model. For more detail on RPC camera models we refer as before to section 11.12.

Our implementation of the linear camera model only models the geometry of the imaging hardware itself and velocity aberration. We do not currently model refraction due to light bending in Earth's atmosphere. It is our understanding that this could represent misplacement of points up to a meter for some imagery. However this is still smaller error than the error from measurement of the spacecraft's position and orientation. The latter can be corrected using bundle adjustment, ideally used with ground control points (section A.4). Alternatively, the `pc_align` tool discussed in section 5.2.5 can be used to align the terrain obtained from ASP to an accurate set of ground measurements.

In the next two sections we will show how to process unmodified and map-projected variants of World View imagery. The imagery we are using is from the free stereo pair labeled "System-Ready (1B) Stereo, 50cm" which captures the city of Stockholm, found on Digital Globe's website [44]. These images represent a non-ideal problem for us since this is an urban location, but at least you should be able to download this

imagery yourself and follow along.

4.1 Processing Raw

After you have downloaded the example stereo imagery of Stockholm, you will find a directory titled

```
056082198020_01_P001_PAN
```

It has a lot of files and many of them contain redundant information just displayed in different formats. We are interested only in the TIF or NTF imagery and the similarly named XML files.

Some Worldview folders will contain multiple image files. This is because Digital Globe breaks down a single observation into multiple files for what we assume are size reasons. These files have a pattern string of "_R[N]C1-", where N increments for every subframe of the full observation. The tool named `dg_mosaic` can be used to mosaic (and optionally reduce the resolution of) such a set of sub-observations into a single image file and create an appropriate camera file

```
> dg_mosaic 12FEB16101327*TIF --output-prefix 12FEB16101327 --reduce-percent 50
```

and analogously for the second set. See section A.11 for more details. The `stereo` program can use either the original or the mosaicked images. This sample data only contains two image files so we do not need to use the `dg_mosaic` tool.

Since we are ingesting these images raw, it is strongly recommended that you use affine epipolar alignment to reduce the search range. The `stereo` command and a rendering of the results are shown below.

```
> stereo -t dg --subpixel-mode 1 --alignment-method affineepipolar \
         12FEB16101327.r50.tif  12FEB16101426.r50.tif         \
         12FEB16101327.r50.xml  12FEB16101426.r50.xml  dg/out
```

Alternatively, the `stereo_gui` frontend can be invoked, with the same options, as described in section A.2.

How to create a DEM and visualize the results of stereo is desribed in section 5.2.

Above, we have used `subpixel-mode 1` which is less accurate but reasonably fast. More details about how to set this and other `stereo` parameters can be found in section 5.1.2.

It is important to note that we could have performed stereo using the approximate RPC model instead of the exact linear camera model (both models are in the same XML file), by switching the session in the `stereo` command above from `-t dg` to `-t rpc`. The RPC model is somewhat less accurate, so the results will not be the same, in our experiments we've seen differences in the 3D terrains using the two approaches of 5 meters or more.

4.2 Processing Map-Projected Imagery

ASP computes the highest quality 3D terrain if used with images map-projected onto a low-resolution DEM that is used as an initial guess. This process is described in section 5.1.7.

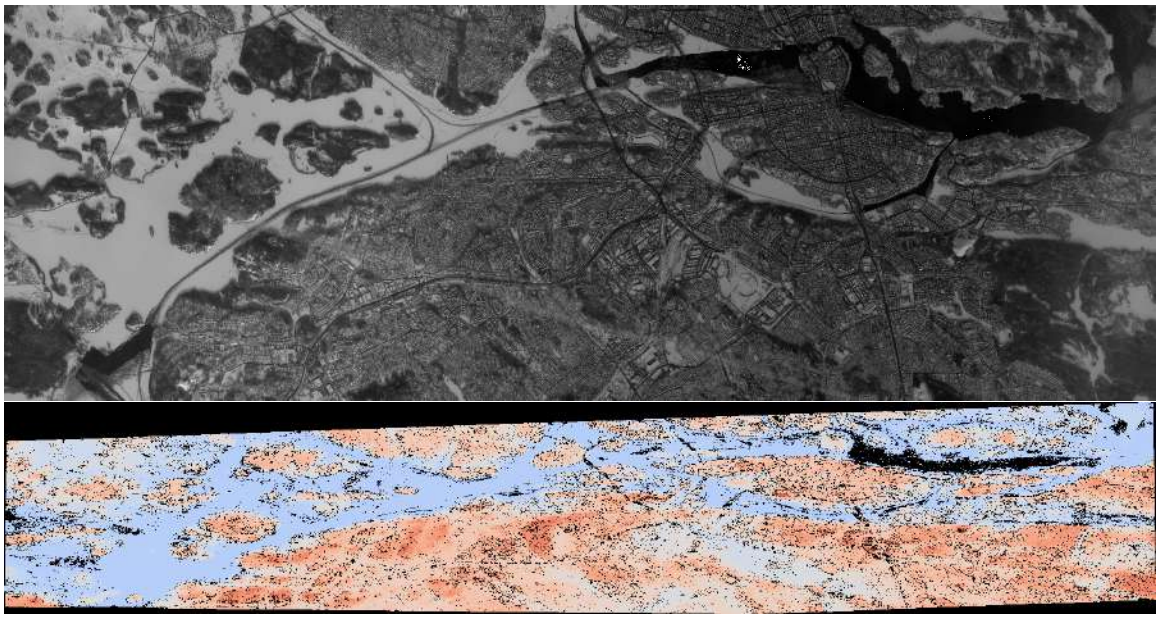


Figure 4.1: Example WorldView image section and colorized height map.

4.3 Handling CCD Boundary Artifacts

Digital Globe World View images [43] may exhibit slight subpixel artifacts which manifest themselves as discontinuities in the 3D terrain obtained using ASP. We provide a tool named `wv_correct`, that can largely correct such artifacts for World View-1 and World View-2 images for most TDI. It can be invoked as follows:

```
> wv_correct image_in.ntf image.xml image_out.tif
```

The corrected images can be used just as the originals, and the camera models do not change. When working with such imagery, we recommend that CCD artifact correction happen first, on original unprojected imagery. Afterward images can be mosaicked with `dg_mosaic`, map-projected, and the resulting data used to run stereo and create terrain models.

This tool is described in section A.24, and an example of using it is in Figure 4.2.

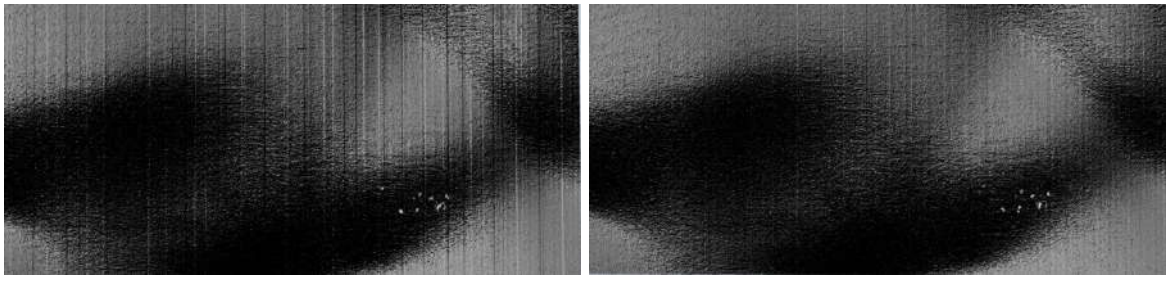


Figure 4.2: Example of a hill-shaded terrain obtained using stereo without (left) and with (right) CCD boundary artifact corrections applied using `wv_correct`.

4.4 Managing Camera Jitter

In this section we will talk about the second largest source of inaccuracies in Digital Globe imagery, after CCD artifacts, namely jitter, and how to correct it.

It is important to note that jitter correction is highly experimental, and while it usually works, it may not be production-ready.

The order in which these corrections need to be handled is the following. First, CCD artifacts are corrected. Then, optionally, images are mosaicked with `dg_mosaic` and map-projected. And jitter should be handled last, during stereo. An exception is made for WV03 images, for which CCD artifacts do not appear to have a significant effect.

Camera jitter has its origin in the fact that the measured position and orientation of the image-acquiring line sensor as specified in a camera XML file is usually not perfectly accurate, the sensor in fact wiggles slightly from where it is assumed to be as it travels through space and appends rows of pixels to the image. This results in slight errors in the final DEM created using stereo. Those are most clearly seen in the intersection error map output by invoking `point2dem --errorimage`.

ASP provides support for correcting this jitter, at least its lower-frequency component. During stereo, right before the triangulation step, so after the left-to-right image disparity is computed, it can solve for adjustments to apply to the satellite position and orientation. Those adjustments are placed along-track (hence at several lines in the image) with interpolation between them. This is quite analogous to what `bundle_adjust` is doing, except that the latter uses just one adjustment for each image.

This process can be triggered by invoking `stereo` with `--image-lines-per-piecewise-adjustment arg`. A recommended value here is 1000, though it is suggested to try several values. A smaller value of `arg` will result in more adjustments being used (each adjustment being responsible for fewer image lines), hence providing finer-grained control, though making this number too small may result in over-fitting and instability. A smaller value here will also require overall more interest point matches (as computed from the disparity), which is set via `--num-matches-for-piecewise-adjustment`.

Jitter correction is more effective if `stereo` is preceded by bundle adjustment, with the adjusted cameras then being passed to `stereo` via `--bundle-adjust-prefix`.

If it appears that the adjustments show some instability at the starting and ending lines due to not enough matches being present (as deduced from examining the intersection error image), the locations of the first and last adjustment (and everything in between) may be brought closer to each other, by modifying `--piecewise-adjustment-percentiles`. Its values are by default 5 and 95, and could be set for example to 10 and 90. For very tall images, it may be desirable to use instead values closer to 0 and 100.

Section B.5 has the full list of parameters used in jitter correction.

In order for jitter correction to be successful, the disparity map (`*-F.tif`) should be of good quality. If that is not the case, it is suggested to redo stereo, and use, for example, map-projected images, and in the case of terrain lacking large scale features, the value `corr-seed-mode 3` (section 4.5).

An illustration of jitter correction is given in figure 4.3.

4.5 Dealing with Terrain Lacking Large-Scale Features

Stereo Pipeline's approach to performing correlation is a two-step pyramid algorithm, in which low-resolution versions of the input images are created, the disparity map (*output_prefix*`-D_sub.tif`) is found, and then this disparity map is refined using increasingly higher-resolution versions of the input images (section 7.2).

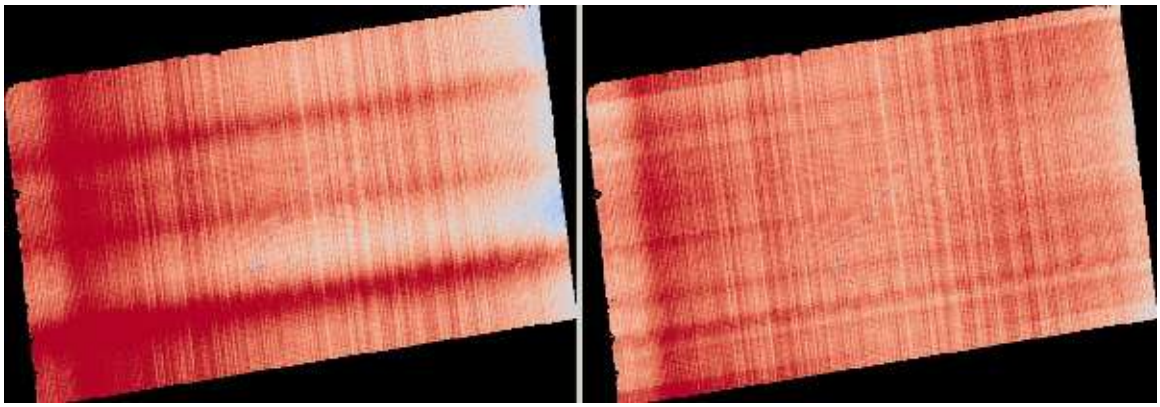


Figure 4.3: Example of a colorized intersection error map before (left) and after jitter correction.

This approach usually works quite well for rocky terrain but may fail for snowy landscapes, whose only features may be small-scale grooves or ridges sculpted by wind (so-called *zastrugi*) that disappear at low resolution.

Stereo Pipeline handles such terrains by using a tool named `sparse_disp` to create *`output_prefix`*`-D_sub.tif` at full resolution, yet only at a sparse set of pixels for reasons of speed. This low-resolution disparity is then refined as earlier using a pyramid approach.

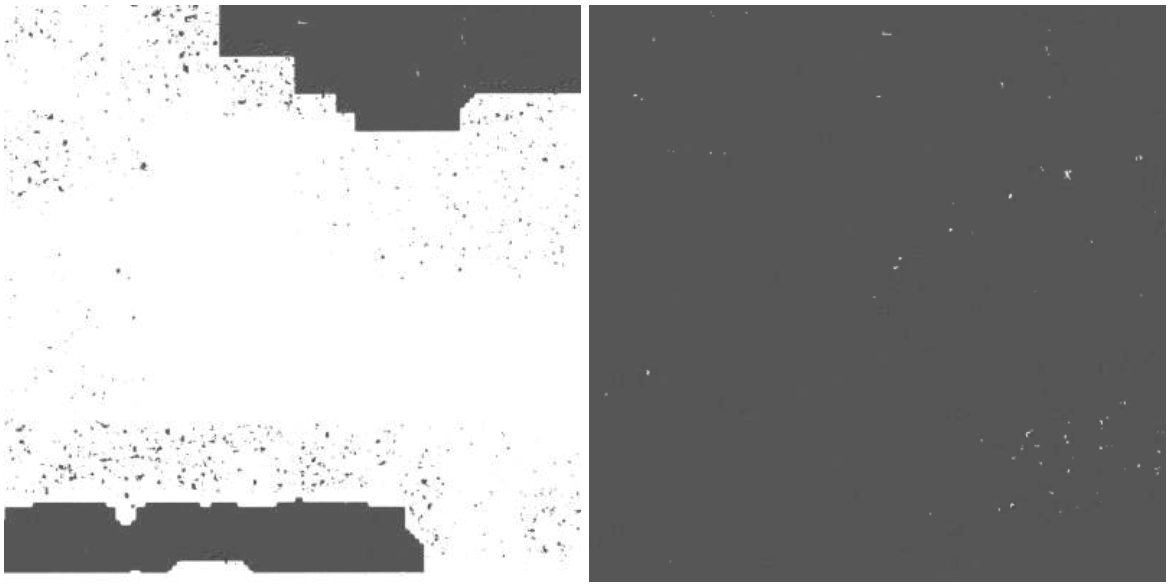


Figure 4.4: Example of a difficult terrain obtained without (left) and with (right) `sparse_disp`. (In these DEMs there is very little elevation change, hence the flat appearance.)

This mode can be invoked by passing to `stereo` the option `--corr-seed-mode 3`. Also, during pyramid correlation it is suggested to use somewhat fewer levels than the default `--corr-max-levels 5`, to again not subsample the images too much and lose the features.

Here is an example:

```
> stereo -t dg --corr-seed-mode 3 --corr-max-levels 2     \
         left_mapped.tif right_mapped.tif                 \
         12FEB12053305-P1BS_R2C1-052783824050_01_P001.XML \
```

```
12FEB12053341-P1BS_R2C1-052783824050_01_P001.XML \
dg/dg srtm_53_07.tif
```

If `sparse_disp` is not working well for your images you may be able to improve its results by experimenting with the set of `sparse_disp` options which can be passed into `stereo` through the `--sparse-disp-options` parameter. `sparse_disp` has so far only been tested with `affineepipolar` image alignment so you may not get good results with other alignment methods.

Since `sparse_disp` is written in Python it depends on a variety of binary Python modules. These modules cannot be distributed with Stereo Pipeline as they depend on the version of Python installed on your system. One way to get these Python modules is to install them yourself. We recommend the Conda Python management system (https://conda.io/docs/index.html) as an easy way to install these dependencies.

`sparse_disp` has been tested on Ubuntu 16.04 using Conda with following additional packages installed:

```
scipy=1.0.0
numpy=1.14.0
simplekml=1.3.0
pyfftw=0.10.4
proj4=4.9.3
gdal=2.2.2
geos=3.6.2
blas=1.0
```

Note that the `simplekml` and `pyfftw` packages needed the argument `-c conda-forge` added to their `conda install` command.

Another way to get these dependencies is to use an installation script provided by ASP. It will download and compile the dependencies of this tool for your platform. The script and instructions are at

https://github.com/NeoGeographyToolkit/BinaryBuilder/tree/master/build_python_modules

After building the `sparse_disp` dependencies, per the instructions, the path to the Python modules must be set, for example as:

```
export ASP_PYTHON_MODULES_PATH=<path to python modules>
```

(once the script from the above location will finish, it will print the value of this variable that must then be set).

This path does not need to be set if you are relying on your own Python installation such as from Conda.

4.6 Processing Multi-Spectral Images

In addition to panchromatic (grayscale) imagery, the Digital Globe satellites also produce lower-resolution multi-spectral (multi-band) images. Stereo Pipeline is designed to process single-band images only. If invoked on multi-spectral data, it will quietly process the first band and ignore the rest. To use one of the other bands it can be singled out by invoking `dg_mosaic` (section 4.1) with the `--band <num>` option. We have evaluated ASP with Digital Globe's multi-spectral images, but support for it is still experimental. We recommend using the panchromatic imagery whenever possible.

Chapter 5

The Next Steps

This chapter will discuss in more detail ASP's stereo process and other tools available to either pre-process the input images/cameras or to manipulate `stereo`'s outputs, both in the context of planetary ISIS data and for Earth imagery. This includes how to (a) customize `stereo`'s settings (b) use `point2dem` to create 3D terrain models, (c) visualize the results, (d) align the obtained point clouds to another data source, (e) perform 3D terrain adjustments in respect to a geoid, etc.

5.1 Stereo Pipeline in More Detail

5.1.1 Stereo Algorithms

The default stereo algorithm in ASP is block-matching, with various values for subpixel refinement, as seen below. The latest version of ASP includes the SGM and MGM algorithms, which overall can perform better, but are more experimental. For details about how to invoke these algorithms, see section 7.2.4.

5.1.2 Setting Options in the `stereo.default` File

The `stereo` program requires a `stereo.default` file that contains settings that affect the stereo reconstruction process. Its contents can be altered for your needs; details are found in appendix B on page 213. You may find it useful to save multiple versions of the `stereo.default` file for various processing needs. If you do this, be sure to specify the desired settings file by invoking `stereo` with the `-s` option. If this option is not given, the `stereo` program will search for a file named `stereo.default` in the current working directory. If `stereo` does not find `stereo.default` in the current working directory and no file was given with the `-s` option, `stereo` will assume default settings and continue.

An example `stereo.default` file is available in the `examples/` directory of ASP. The actual file has a lot of comments to show you what options and values are possible. Here's a trimmed version of the important values in that file.

```
alignment-method affineepipolar
cost-mode 2
corr-kernel 21 21
subpixel-mode 1
subpixel-kernel 21 21
```

All these options can be overridden from the command line, as described in section 5.1.5.

Alignment Method

The most important line in `stereo.default` is the first one, specifying the alignment method. For raw images, alignment is always necessary, as the left and right images are from different perspectives. Several alignment methods are supported, including `affineepipolar` and `homography` (see section B.1 for details).

Alternatively, stereo can be performed with map-projected images (section 5.1.7). In effect we take a smooth low-resolution terrain and map both the left and right raw images onto that terrain. This automatically brings both images into the same perspective, and as such, for map-projected images the alignment method is always set to `none`.

Correlation Parameters

The second and third lines in `stereo.default` define what correlation metric *(normalized cross correlation)* we'll be using and how big the template or kernel size should be *(21 pixels square)*. A pixel in the left image will be matched to a pixel in the right image by comparing the windows of this size centered at them.

Making the kernel sizes smaller, such as 15×15, or even 11×11, may improve results on more complex features, such as steep cliffs, at the expense of perhaps introducing more false matches or noise.

Subpixel Refinement Parameters

A highly critical parameter in ASP is the value of `subpixel-mode`, on the fourth line. When set to 1, `stereo` performs parabola subpixel refinement, which is very fast but not very accurate. When set to 2, it produces very accurate results, but it is about an order of magnitude slower. When set to 3, the accuracy and speed will be somewhere in between the other methods.

The fifth line sets the kernel size to use during subpixel refinement *(also 21 pixels square)*.

Search Range Determination

Using these settings alone, ASP will attempt to work out the minimum and maximum disparity it will search for automatically. However if you wish to, you can explicitly set the extent of the search range by adding the option:

```
corr-search -80 -2 20 2
```

More details about this option and the inner workings of stereo correlation can be found in chapter 7.

5.1.3 Performing Stereo Correlation

As already mentioned, the `stereo` program can be invoked for ISIS images as

```
ISIS 3> stereo left_image.cub right_image.cub \
          -s stereo.default results/output
```

For Digital Globe imagery the cameras need to be specified separately:

```
> stereo left.tif right.tif left.xml right.xml \
    -s stereo.default results/output
```

Figure 5.1: These are the four viewable `.tif` files created by the `stereo` program. On the left are the two aligned, pre-processed images: (`results/output-L.tif` and `results/output-R.tif`). The next two are mask images (`results/output-lMask.tif` and `results/output-rMask.tif`), which indicate which pixels in the aligned images are good to use in stereo correlation. The image on the right is the "Good Pixel map", (`results/output-GoodPixelMap.tif`), which indicates (in gray) which were successfully matched with the correlator, and (in red) those that were not matched.

As stated in section 3.1, the string `results/output` is arbitrary, and in this case we will simply make all outputs go to the `results` directory.

When `stereo` finishes, it will have produced a point cloud image. Section 5.2 describes how to convert it to a digital elevation model (DEM) or other formats.

The `stereo` command can also take multiple input images, performing multi-view stereo (section 5.1.8).

5.1.4 Running the GUI Frontend

The `stereo_gui` program is a GUI frontend to `stereo`. It is invoked with the same options as `stereo`. It displays the input images, and makes it possible to zoom in and select smaller regions to run stereo on. The GUI is described in section A.2.

5.1.5 Specifying Settings on the Command Line

All the settings given via the `stereo.default` file can be over-ridden from the command line. Just add a double hyphen (`--`) in front the option's name and then fill out the option just as you would in the configuration file. For options in the `stereo.default` file that take multiple numbers, they must be separated by spaces (like '`corr-kernel 25 25`') on the command line. Here is an example in which we override the search range and subpixel mode from the command line.

```
ISIS 3> stereo E0201461.map.cub M0100115.map.cub  \
          -s stereo.map --corr-search -70 -4 40 4 \
          --subpixel-mode 0 results/output
```

5.1.6 Stereo on Multiple Machines

If the input images are really large it may desirable to distribute the work over several computing nodes. ASP provides a tool named `parallel_stereo` for that purpose. Its usage is described in section A.3.

5.1.7 Running Stereo with Map-projected Images

The way stereo correlation works is by matching a neighborhood of each pixel in the left image to a similar neighborhood in the right image. This matching process can fail or become unreliable if the two images are too different, which can happen for example if the perspectives of the two cameras are very different or the underlying terrain has steep portions. This will result in ASP producing terrains with noise or missing data.

ASP can mitigate this by *map-projecting* the left and right images onto some pre-existing low-resolution smooth terrain model without holes, and using the output images to do stereo. In effect, this makes the images much more similar and more likely for stereo correlation to succeed.

In this mode, ASP does not create a terrain model from scratch, but rather uses an existing terrain model as an initial guess, and improves on it.

For Earth, an existing terrain model can be, for example, NASA SRTM, GMTED2010, USGS's NED data, or NGA's DTED data. There exist pre-made terrain models for other planets as well, for example the Moon LRO LOLA global DEM and the Mars MGS MOLA DEM.

Alternatively, a low-resolution smooth DEM can be obtained by running ASP itself as described in previous sections. In such a run, subpixel mode may be set to parabola (`subpixel-mode 1`) for speed. To make it sufficiently coarse and smooth, the resolution can be set to about 40 times coarser than either the default `point2dem` resolution or the resolution of the input images. If the resulting DEM turns out to be noisy or have holes, one could change in `point2dem` the search radius factor, use hole-filling, invoke more aggressive outlier removal, and erode pixels at the boundary (those tend to be less reliable). Alternatively, holes can be filled with `dem_mosaic`.

The tool used for map-projecting the images is called `mapproject` (section A.12). It is very important to specify correctly the output resolution (the `--tr` option for `mapproject`) when creating map-projected images. For example, if the input images are about 1 meter/pixel, the same number should be used in `mapproject` (if the desired projection is in degrees, this value should be converted to degrees). If the output resolution is not correct, there will be artifacts in the stereo results.

Some experimentation on a small area may be necessary to obtain the best results. Once images are map-projected, they can be cropped to a small shared region using `gdal_translate -projwin` and then stereo with these clips can be invoked.

Example for ISIS images

In this example we illustrate how to run stereo with map-projected images for ISIS data. We start with LRO NAC Lunar images M1121224102LE and M1121209902LE from ASU's LRO NAC web site, http://lroc.sese.asu.edu/. We convert them to ISIS cubes using the ISIS program `lronac2isis`, then we use the ISIS tools `spiceinit`, `lronaccal`, and `lrnonacecho` to update the SPICE kernels and to do radiometric and echo correction. We name the two obtained .cub files `left.cub` and `right.cub`.

Here we decided to run ASP to create the low-resolution DEM needed for map-projection, rather than get them from an external source. For speed, we process just a small portion of the images:

```
parallel_stereo left.cub right.cub            \
  --left-image-crop-win 1984 11602 4000 5000  \
  --right-image-crop-win 3111 11027 4000 5000 \
  --job-size-w 1024 --job-size-h 1024         \
  --subpixel-mode 1                           \
  run_nomap/run
```

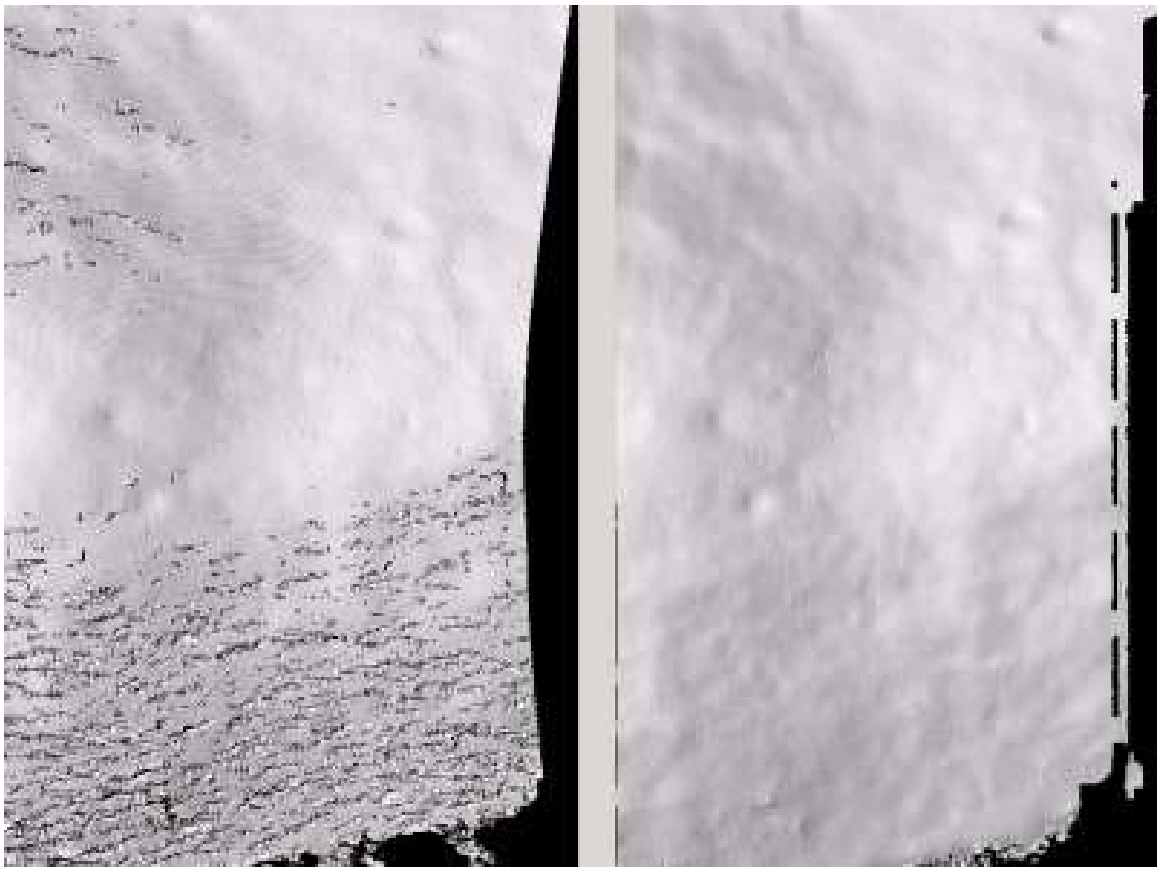


Figure 5.2: A DEM obtained using plain stereo (left) and stereo with map-projected images (right). Their quality will be comparable for relatively flat terrain and the second will be much better for rugged terrain. The right image has some artifacts, but those are limited to areas close to the boundary.

(the crop windows can be determined using `stereo_gui`). The input images have resolution of about 1 meter, or 3.3×10^{-5} degrees on the Moon. We create the low-resolution DEM using a resolution 40 times as coarse, so we use a grid size of 0.0013 degrees (we use degrees since the default `point2dem` projection invoked here is `longlat`).

```
point2dem --search-radius-factor 5 --tr 0.0013 run_nomap/run-PC.tif
```

As mentioned earlier, some tweaks to the parameters used by `point2dem` may be necessary for this low-resolution DEM to be smooth enough and with no holes.

Note that we used `--search-radius-factor 5` to expand the DEM a bit, to counteract future erosion in stereo due to the correlation kernel size.

If this terrain is close to the poles, say within 25 degrees of latitude, it is advised to use a stereographic projection, centered either at the nearest pole, or close to the center of the current DEM. Its center's longitude and latitude can be found with `gdalinfo -stats`, which can then be passed to `point2dem` such as

```
point2dem --stereographic --proj-lon <lon_ctr> --proj-lat <lat_ctr> ...
```

By calling `gdalinfo -proj4`, the PROJ.4 string of the obtained DEM can be found, which can be used in mapprojection later, and with the resolution switched to meters from degrees (see section 5.1.7 for more details).

Next, we map-project the images onto this DEM, using the original resolution of 3.3×10^{-5} degrees.

```
mapproject --tr 0.000033 run_nomap/run-DEM.tif left.cub left_proj.tif \
  --t_projwin 3.6175120 25.5669989 3.6653695 25.4952127
mapproject --tr 0.000033 run_nomap/run-DEM.tif right.cub right_proj.tif \
  --t_projwin 3.6175120 25.5669989 3.6653695 25.4952127
```

Notice that we restricted the area of computation using `--t_projwin` to again make the process faster.

Next, we do stereo with these map-projected images.

```
parallel_stereo --job-size-w 1024 --job-size-h 1024 \
  --subpixel-mode 3                                 \
  left_proj.tif right_proj.tif left.cub right.cub   \
  run_map/run run_nomap/run-DEM.tif
```

Notice that even though we use map-projected images, we still specified the original images as the third and fourth arguments. That because we need the camera information from those files. The fifth argument is the output prefix, while the sixth is the low-resolution DEM we used for map-projection. We have used here `--subpixel-mode 3` as this will be the final point cloud and we want the increased accuracy.

Lastly, we create a DEM at 1 meter resolution:

```
point2dem --nodata-value -32768 --tr 0.000033 run_map/run-PC.tif
```

Note here that we could have used a coarser resolution for the final DEM, such as 4 meters/pixel, since we won't see detail at the level of 1 meter in this DEM, as the stereo process is lossy. This is explained in more detail in section A.6.2.

In figure 5.2 we show the effect of using map-projected images on accuracy of the final DEM.

It is important to note that we could have map-projected the images using the ISIS tool `cam2map`, as described in section 3.2.2. The current approach could be preferable since it allows us to choose the DEM to map-project onto, and it is much faster, since ASP's `mapproject` uses multiple processes, while `cam2map` is restricted to one process and one thread.

Example for Digital Globe Images

In this section we will describe how to run stereo with map-projected images for Digital Globe cameras for Earth. The same process can be used with very minor modifications for any satellite imagery that uses the the RPC camera model. All that is needed is to replace the stereo session when invoking `stereo` below with `rpcmaprpc` from `dgmaprpc`.

Unlike the previous section, here we will use an external DEM to map-project onto, rather than creating our own. We will use a variant of NASA SRTM data with no holes. Other choices have been mentioned earlier.

It is important to note that ASP expects the input low-resolution DEM to be in reference to a datum ellipsoid, such as WGS84 or NAD83. If the DEM is in respect to either the EGM96 or NAVD88 geoids, the ASP tool `dem_geoid` can be used to convert the DEM to WGS84 or NAD83 (section A.10). (The same tool can be used to convert back the final output ASP DEM to be in reference to a geoid, if desired.)

Not applying this conversion might not properly negate the parallax seen between the two images, though it will not corrupt the triangulation results. In other words, sometimes one may be able to ignore the vertical datums on the input but we do not recommend doing that. Also, you should note that the geoheader attached to those types of files usually does not describe the vertical datum they used. That can only be understood by careful reading of your provider's documents.

In this example we use as an input low-resolution DEM the file `srtm_53_07.tif`, a 90 meter resolution tile from the CGIAR-CSI modification of the original NASA SRTM product [39]. The NASA SRTM square for this example spot in India is N26E080.

Below are the commands for map-projecting the input and then running through stereo. You can use any projection you like as long as it preserves detail in the imagery. Note that the last parameter in the stereo call is the input low-resolution DEM. The dataset is the same as the one used in section 4.1.

For best quality results, the resolution used for mapprojection should be very similar to the documented ground sample distance (GSD) for your camera.

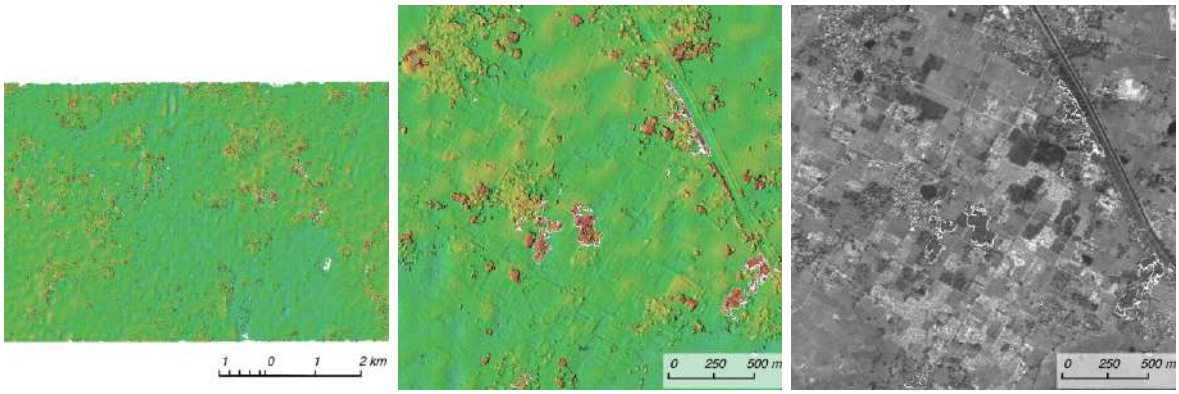


Figure 5.3: Example colorized height map and ortho image output.

Commands

```
mapproject -t rpc --t_srs "+proj=eqc +units=m +datum=WGS84" \
  --tr 0.5 srtm_53_07.tif                                   \
  12FEB12053305-P1BS_R2C1-052783824050_01_P001.TIF          \
  12FEB12053305-P1BS_R2C1-052783824050_01_P001.XML          \
  left_mapped.tif
mapproject -t rpc --t_srs "+proj=eqc +units=m +datum=WGS84" \
  --tr 0.5 srtm_53_07.tif                                   \
  12FEB12053341-P1BS_R2C1-052783824050_01_P001.TIF          \
  12FEB12053341-P1BS_R2C1-052783824050_01_P001.XML          \
  right_mapped.tif
stereo -t dgmaprpc --subpixel-mode 1 --alignment-method none  \
       left_mapped.tif right_mapped.tif                       \
       12FEB12053305-P1BS_R2C1-052783824050_01_P001.XML \
       12FEB12053341-P1BS_R2C1-052783824050_01_P001.XML \
       dg/dg srtm_53_07.tif
```

If the `--t_srs` option is not specified, it will be read from the low-resolution input DEM.

The complete list of options for `mapproject` is described in section A.12.

In the `stereo` command, we have used `subpixel-mode 1` which is less accurate but reasonably fast. We have also used `alignment-method none`, since the images are map-projected onto the same terrain with the same resolution, thus no additional alignment is necessary. More details about how to set these and other `stereo` parameters can be found in section 5.1.2.

It is important to note here that any Digital Globe camera file has two models in it, the exact linescan model (which we name `DG`), and its `RPC` approximation. Above, we have used the approximate RPC model for map-projection, since map-projection is just a pre-processing step to make the images more similar to each other, this step will be undone during stereo triangulation, and hence using the RPC model is good enough, while being much faster than the exact `DG` model. At the stereo stage, we see above that we invoked the `dgmaprpc` session, which suggests that we have used the RPC model during map-projection, but we would like to use the accurate DG model when performing actual triangulation from the cameras to the ground.

RPC and Pinhole Camera Models

Map-projected images can also be used with RPC and Pinhole camera models. The `mapproject` command needs to be invoked with `-t rpc` and `-t pinhole` respectively. As earlier, when invoking `stereo` the the first two arguments should be the map-projected images, followed by the camera models, output prefix, and the name of the DEM used for map-projection. The session name passed to `stereo` should be `rpcmaprpc` and `pinholemappinhole` respectively.

5.1.8 Multi-View Stereo

ASP supports multi-view stereo at the triangulation stage. This mode is somewhat experimental, and not used widely. We have obtained higher quality results by doing pairwise stereo and merging the result, as described later on in thi section.

In the multiview scenario, the first image is set as reference, disparities are computed from it to all the other images, and then joint triangulation is performed [136]. A single point cloud is generated with one 3D point for each pixel in the first image. The inputs to multi-view stereo and its output point cloud are handled in the same way as for two-view stereo (e.g., inputs can be map-projected, the output can be converted to a DEM, etc.).

It is suggested that images be bundle-adjusted (section section 8.2) before running multi-view stereo.

Example (for ISIS with three images):

```
stereo file1.cub file2.cub file3.cub results/run
```

Example (for Digital Globe data with three map-projected images):

```
stereo file1.tif file2.tif file3.tif file1.xml file2.xml file3.xml \
  results/run input-DEM.tif
```

The `parallel_stereo` tool can also be used with multiple images (section A.3).

For a sequence of images, multi-view stereo can be run several times with each image as a reference, and the obtained point clouds combined into a single DEM using `point2dem` (section A.6).

The ray intersection error, the fourth band in the point cloud file, is computed as twice the mean of distances from the optimally computed intersection point to the individual rays. For two rays, this agrees with the

intersection error for two-view stereo which is defined as the minimal distance between rays. For multi-view stereo this error is much less amenable to interpretation than for two-view stereo, since the number of valid rays corresponding to a given feature can vary across the image, which results in discontinuities in the intersection error.

Other ways of combining multiple images

As an alternative to multi-view stereo, point clouds can be generated from multiple stereo pairs, and then a single DEM can be created with `point2dem` (section 5.2.2). Or, multiple DEMs can be created, then combined into a single DEM with `dem_mosaic` (section A.8).

In both of these approaches, the point clouds could be registered to a trusted dataset using `pc_align` before creating a combined terrain model (section 5.2.5).

The advantage of creating separate DEMs and then merging them (after alignment) with `dem_mosaic`, compared to multiview triangulation, is that this approach will not create visible seams, while likely it will still increase the accuracy compared to the individual input DEMs.

5.1.9 Diagnosing Problems

Once invoked, `stereo` proceeds through several stages that are detailed on page 142. Intermediate and final output files are generated as it goes. See Appendix C, page 223 for a comprehensive listing. Many of these files are useful for diagnosing and debugging problems. For example, as Figure 5.1 shows, a quick look at some of the TIFF files in the `results/` directory provides some insight into the process.

Perhaps the most accessible file for assessing the quality of your results is the good pixel image, (`results/output-GoodPixelMap.tif`). If this file shows mostly good, gray pixels in the overlap area (the area that is white in both the `results/output-lMask.tif` and `results/output-rMask.tif` files), then your results are just fine. If the good pixel image shows lots of failed data, signified by red pixels in the overlap area, then you need to go back and tune your `stereo.default` file until your results improve. This might be a good time to make a copy of `stereo.default` as you tune the parameters to improve the results.

Whenever `stereo`, `point2dem`, and other executables are run, they create log files in given tool's results directory, containing a copy of the configuration file, the command that was run, your system settings, and tool's console output. This will help track what was performed so that others in the future can recreate your work.

Another handy debugging tool is the `disparitydebug` program, which allows you to generate viewable versions of the intermediate results from the stereo correlation algorithm. `disparitydebug` converts information in the disparity image files into two TIFF images that contain horizontal and vertical components of the disparity (i.e. matching offsets for each pixel in the horizontal and vertical directions). There are actually three flavors of disparity map: the `-D.tif`, the `-RD.tif`, and `-F.tif`. You can run `disparitydebug` on any of them. Each shows the disparity map at the different stages of processing.

```
> disparitydebug results/output-F.tif
```

If the output H and V files from `disparitydebug` look good, then the point cloud image is most likely ready for post-processing. You can proceed to make a mesh or a DEM by processing `results/output-PC.tif` using the `point2mesh` or `point2dem` tools, respectively.

Figure 5.4 shows the outputs of `disparitydebug`.

If the input images are map-projected (georeferenced) and the alignment method is `none`, all images output by stereo are georeferenced as well, such as GoodPixelMap, D_sub, disparity, etc. As such, all these data can be overlayed in `stereo_gui`. `disparitydebug` also preserves any georeference.

5.1.10 Dealing with Long Run-times

If `stereo_corr` takes unreasonably long, it may have encountered a portion of the image where, due to noise (such as clouds, shadows, etc.) the determined search range is much larger than what it should be. The option `--corr-timeout` *integer* can be used to limit how long each 1024×1024 pixel tile can take. A good value here could be 300 (seconds) or more if your terrain is expected to have large height variations.

5.2 Visualizing and Manipulating the Results

When `stereo` finishes, it will have produced a point cloud image. At this point, many kinds of data products can be built from the `results/output-PC.tif` point cloud file.

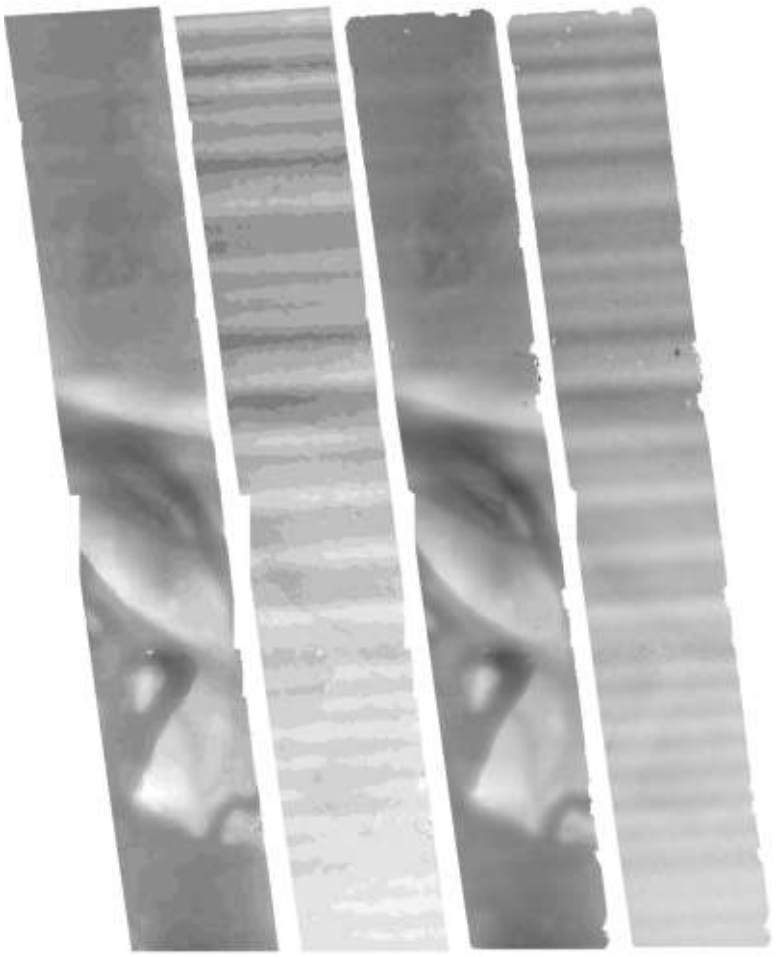


Figure 5.4: Disparity images produced using the `disparitydebug` tool. The two images on the left are the `results/output-D-H.tif` and `results/output-D-V.tif` files, which are normalized horizontal and vertical disparity components produced by the disparity map initialization phase. The two images on the right are `results/output-F-H.tif` and `results/output-F-V.tif`, which are the final filtered, sub-pixel-refined disparity maps that are fed into the Triangulation phase to build the point cloud image. Since these MOC images were acquired by rolling the spacecraft across-track, most of the disparity that represents topography is present in the horizontal disparity map. The vertical disparity map shows disparity due to "washboarding," which is not from topography but from spacecraft movement. Note however that the horizontal and vertical disparity images are normalized independently. Although both have the same range of gray values from white to black, they represent significantly different absolute ranges of disparity.

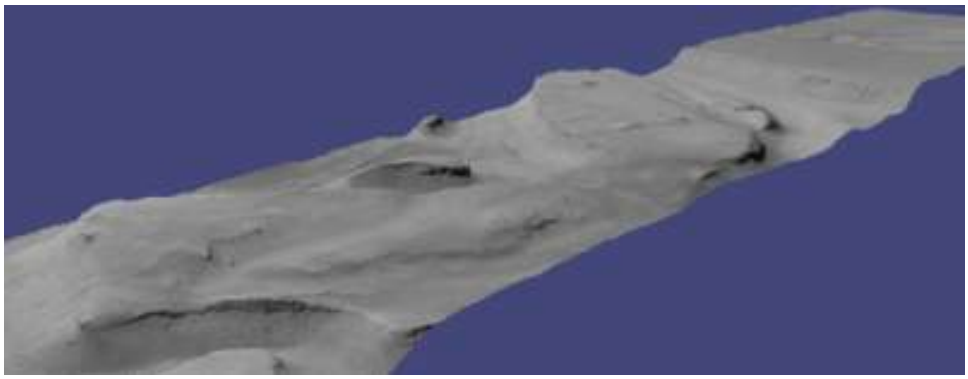


Figure 5.5: The `results/output.osgb` file displayed in the OSG Viewer.

5.2.1 Building a 3D Mesh Model

If you wish to see the data in an interactive 3D browser, then you can generate a 3D object file using the `point2mesh` command (page 163). The resulting file is stored in Open Scene Graph binary format [27]. It can be viewed with `osgviewer` (the Open Scene Graph Viewer program, distributed with the binary version of the Stereo Pipeline). The `point2mesh` program takes the point cloud file and the left normalized image as inputs:

```
> point2mesh results/output-PC.tif results/output-L.tif
> osgviewer results/output.osgb
```

The image displayed by `osgviewer` is shown in figure 5.5.

When the `osgviewer` program starts, you may want to toggle the lighting with the 'L' key, toggle texturing with the 'T' key, and toggle wireframe mode with the 'W'. Press '?' to see a variety of other interactive options.

If you already have a DEM and an ortho image (section 5.2.2), they can be used to build a mesh as well, in the same way as done above:

```
> point2mesh results/output-DEM.tif results/output-DRG.tif
```

5.2.2 Building a Digital Elevation Model and Ortho Image

The `point2dem` program (page 158) creates a Digital Elevation Model (DEM) from the point cloud file.

```
>  point2dem results/output-PC.tif
```

The resulting TIFF file is map-projected and will contain georeferencing information stored as GeoTIFF tags.

The tool will infer the datum and projection from the input images, if present. You can explicitly specify a coordinate system (e.g., mercator, sinusoidal) and a reference spheroid (i.e., calculated for the Moon, Mars, or Earth). Alternatively, the datum semi-axes can be set or a PROJ.4 string can be passed in.

```
>  point2dem -r mars results/output-PC.tif
```

The output DEM will be named `results/output-DEM.tif`. It can be imported into a variety of GIS platforms. The DEM can be transformed into a hill-shaded image for visualization (section 5.2.9). Both the DEM itself and its hill-shaded version can be examined in `stereo_gui`.

The `point2dem` program can also be used to orthoproject raw satellite imagery onto the DEM. To do this, invoke `point2dem` just as before, but add the `--orthoimage` option and specify the use of the left image file as the texture file to use for the projection:

```
> point2dem results/output-PC.tif --orthoimage results/output-L.tif
```

The texture file must always be specified after the point cloud file in this command. See figure 5.6 on the right for the output image.

To fill in any holes in the obtained orthoimage, one can invoke it with a larger value of the grid size (the `--tr` option) and/or with a variation of the options:

```
--no-dem --orthoimage-hole-fill-len 100 --search-radius-factor 2
```

The `point2dem` program is also able to accept output projection options the same way as the tools in GDAL. Well-known EPSG, IAU2000 projections, and custom PROJ.4 strings can applied with the target spatial reference set flag, `--t_srs`. If the target spatial reference flag is applied with any of the reference spheroid options, the reference spheroid option will overwrite the datum defined in the target spatial reference set. The following examples produce the same output. However, the last two results will also show correctly the name of the datum in the geoheader, not just the values of its axes.

```
point2dem --t_srs "+proj=longlat +a=3396190 +b=3376200"
   results/output-PC.tif

point2dem --t_srs http://spatialreference.org/ref/iau2000/49900/ \
   results/output-PC.tif

 point2dem --t_srs 'GEOGCS["Geographic Coordinate System",
                      DATUM["D_Mars_2000",
                      SPHEROID["Mars_2000_IAU_IAG",3396190,169.894447223611]],
                      PRIMEM["Greenwich",0],
                      UNIT["degree",0.0174532925199433]]' results/output-PC.tif
```

The `point2dem` program can be used in many different ways. The complete documentation is in section A.6.

5.2.3 Orthorectification of an Image From a Different Source

If you have already obtained a DEM, using ASP or some other approach, and have an image and camera pair which you would like to overlay on top of this terrain, use the `mapproject` tool (section A.12).

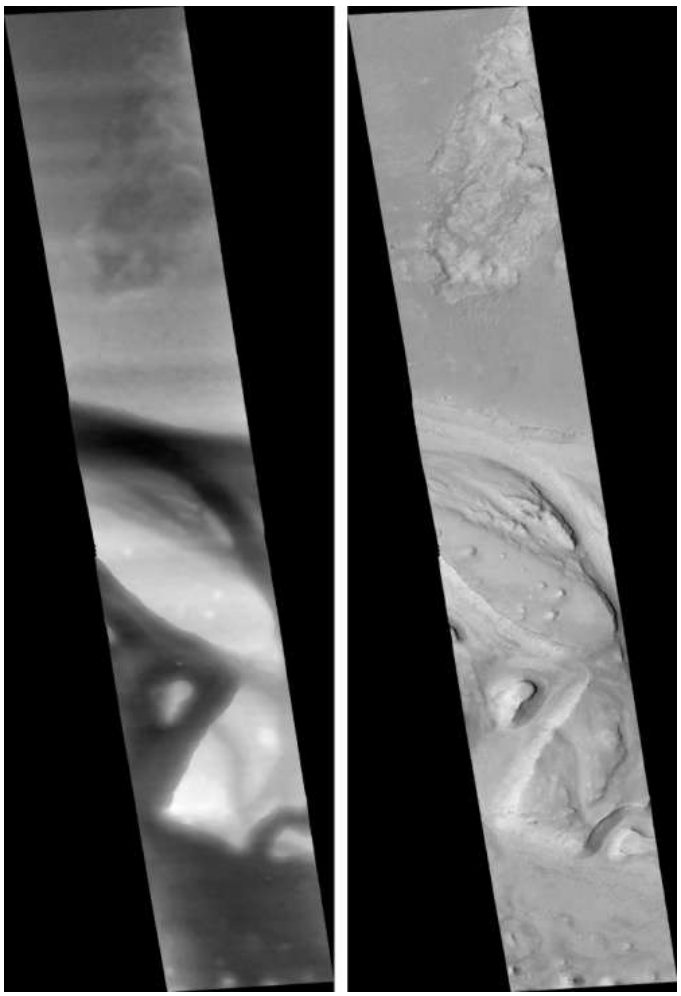


Figure 5.6: The image on the left is a normalized DEM (generated using `point2dem`'s `-n` option), which shows low terrain values as black and high terrain values as white. The image on the right is the left input image projected onto the DEM (created using the `--orthoimage` option to `point2dem`).

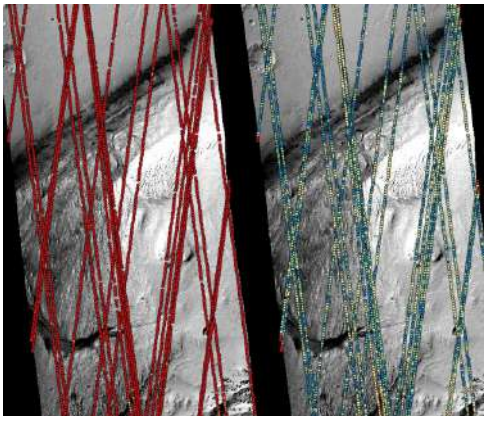


Figure 5.7: Example of using `pc_align` to align a DEM obtained using stereo from CTX images to a set of MOLA tracks. The MOLA points are colored by the offset error initially (left) and after pc align was applied (right) to the terrain model. The red dots indicate more than 100 m of error and blue less than 5 m. The `pc_align` algorithm determined that by moving the terrain model approximately 40 m south, 70 m west, and 175 m vertically, goodness of fit between MOLA and the CTX model was increased substantially.

5.2.4 Correcting Camera Positions and Orientations

The `bundle_adjust` program can be used to adjust the camera positions and orientations before running stereo. These adjustments only makes the cameras self-consistent. For the adjustments to be absolute, it is necessary to use `bundle_adjust` with ground control points. This tool is described in section A.4.

5.2.5 Alignment to Point Clouds From a Different Source

Often the 3D terrain models output by `stereo` (point clouds and DEMs) can be intrinsically quite accurate yet their actual position on the planet may be off by several meters or several kilometers, depending on the spacecraft. This can result from small errors in the position and orientation of the satellite cameras taking the pictures.

Such errors can be corrected in advance using bundle adjustment, as described in the previous section. That requires using ground control points, that may not be easy to collect. Alternatively, the images and cameras can be used as they are, and the absolute position of the output point clouds can be corrected in post-processing. For that, ASP provides a tool named `pc_align`. It aligns a 3D terrain to a much more accurately positioned (if potentially sparser) dataset. Such datasets can be made up of GPS measurements (in the case of Earth), or from laser altimetry instruments on satellites, such as ICESat/GLASS for Earth, LRO/LOLA on the Moon, and MGS/MOLA on Mars. Under the hood, `pc_align` uses the Iterative Closest Point algorithm (ICP) (both the point-to-plane and point-to-point flavors are supported, and with point-to-point ICP it is also possible to solve for a scale change).

The `pc_align` tool requires another input, an a priori guess for the maximum displacement we expect to see as result of alignment, i.e., by how much the points are allowed to move when the alignment transform is applied. If not known, a large (but not unreasonably so) number can be specified. It is used to remove most of the points in the source (movable) point cloud which have no chance of having a corresponding point in the reference (fixed) point cloud.

Here is how `pc_align` can be called (the denser cloud is specified first).

```
> pc_align --max-displacement 200 --datum MOLA   \
    --save-inv-transformed-reference-points      \
    --csv-format '1:lon 2:lat 3:radius_m'        \
    stereo-PC.tif mola.csv
```

It is important to note here that there are two widely used Mars datums, and if your CSV file has, unlike above, the heights relative to a datum, the correct datum name must be specified via `--datum`. Section A.6.1 talks in more detail about the Mars datums.

Figure 5.7 shows an example of using `pc_align`. The complete documentation for this program is in section A.21.

5.2.6 Alignment and Orthoimages

Two related issues are discussed here. The first is that sometimes, after ASP has created a DEM, and the left and right images are map-projected to it, they are shifted in respect to each other. That is due to the errors in camera positions. To rectify it, one has to run `bundle_adjust` first, then rerun the stereo and mapprojection tools, with the adjusted cameras being passed to both via `--bundle-adjust-prefix`.

Note that this approach will create self-consistent outputs, but not necessarily aligned with pre-existing ground truth. That we deal with next.

Once an ASP-generated DEM has been aligned to known ground data using `pc_align`, it may be desired to create orthoimages that are also aligned to the ground. That can be accomplished in two ways.

The `point2dem --orthoimage` approach be used, and one can pass to it the point cloud after alignment and the `L` image before alignment (all this tool does is copy pixels from the texture image, so position errors are not a problem).

Alternatively, one can invoke the `mapproject` tool again. Yet, there is a challenge, because this tool uses the original cameras, before alignment, but will project onto the DEM after alignment, so the obtained orthoimage location on the ground will be wrong.

The solution is to invoke `bundle_adjust` on the two input images and cameras, while passing to it the transform obtained from `pc_align` via the `--initial-transform` option. This will shift the cameras to the right place, and then `mapproject` can be called with the adjusted cameras, using again the `--bundle-adjust-prefix` option. If all that is wanted is to shift the cameras, without doing any actual adjustments, the tool can be invoked with 0 iterations.

5.2.7 Creating DEMs Relative to the Geoid/Areoid

The DEMs generated using `point2dem` are in reference to a datum ellipsoid. If desired, the `dem_geoid` program can be used to convert this DEM to be relative to a geoid/areoid on Earth/Mars respectively. Example usage:

```
> dem_geoid results/output-DEM.tif
```

5.2.8 Converting to the LAS Format

If it is desired to use the `stereo` generated point cloud outside of ASP, it can be converted to the LAS file format, which is a public file format for the interchange of 3-dimensional point cloud data. The tool `point2las` can be used for that purpose (section A.20). Example usage:

```
> point2las --compressed -r Earth results/output-PC.tif
```

5.2.9 Generating Color Hillshade Maps

Once you have generated a DEM file, you can use the `colormap` and `hillshade` tools to create colorized and/or shaded relief images.

To create a colorized version of the DEM, you need only specify the DEM file to use. The colormap is applied to the full range of the DEM, which is computed automatically. Alternatively you can specify your own min and max range for the color map.

```
> colormap results/output-DEM.tif -o hrad-colorized.tif
```

To create a hillshade of the DEM, specify the DEM file to use. You can control the azimuth and elevation of the light source using the `-a` and `-e` options.

```
> hillshade results/output-DEM.tif -o hrad-shaded.tif -e 25 -a 300
```

To create a colorized version of the shaded relief file, specify the DEM and the shaded relief file that should be used:

```
> colormap results/output-DEM.tif -s hrad-shaded.tif -o hrad-color-shaded.tif
```

See figure 5.8 showing the images obtained with these commands.

The complete documentation for `colormap` is in section A.29, and for `hillshade` in section A.30.

5.2.10 Building Overlays for Moon and Mars Mode in Google Earth

Sometimes it may be convenient to see how the DEMs and orthoimages generated by ASP look on top of existing imagery in Google Earth. ASP provides a tool named `image2qtree` for that purpose. It creates multi-resolution image tiles and a metadata tree in KML format that can be loaded into Google Earth from your local hard drive or streamed from a remote server over the Internet.

The `image2qtree` program can only be used on 8-bit image files with georeferencing information (e.g. grayscale or RGB GeoTIFF images). In this example, it can be used to process

`results/output-DEM-normalized.tif`, `results/output-DRG.tif`, `hrad-shaded.tif`, `hrad-colorized.tif`, and `hrad-shaded-colorized.tif`.

These images were generated respectively by using `point2dem` with the `-n` option creating a normalized DEM, the `--orthoimage` option to `point2dem` which projects the left image onto the DEM, and the images created earlier with `colormap`.

Here's an example of how to invoke this program.

```
> image2qtree hrad-shaded-colorized.tif -m kml --draw-order 100
```

Figure 5.9 shows the obtained KML files in Google Earth.

The complete documentation is in section A.31.

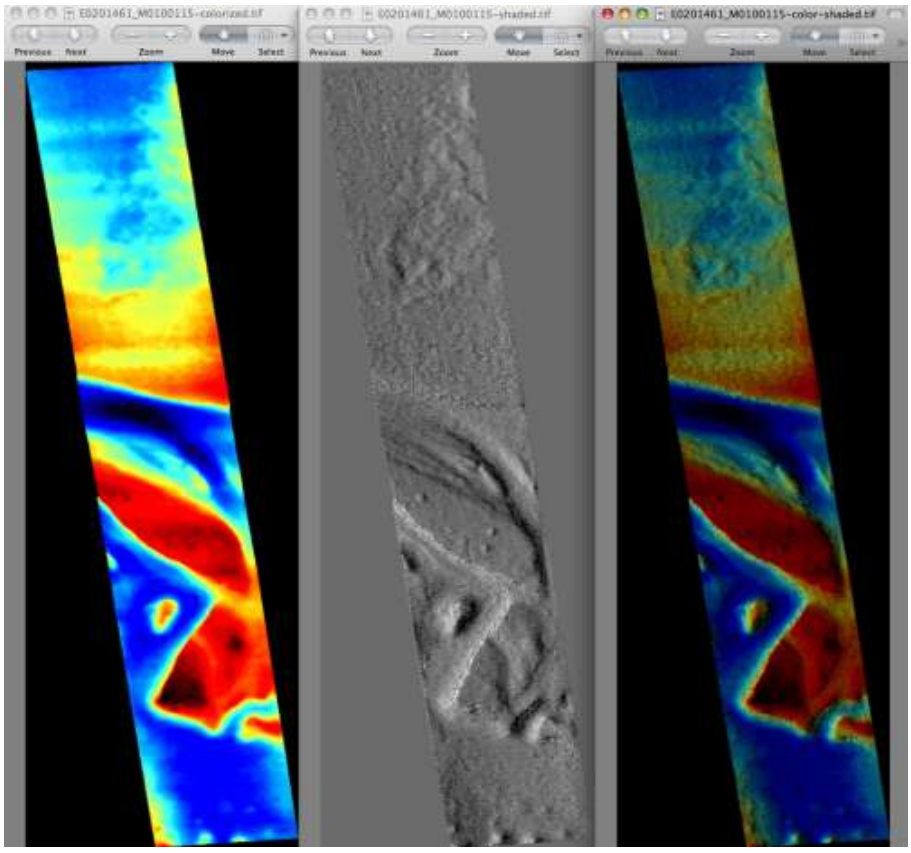


Figure 5.8: The colorized DEM, the shaded relief image, and the colorized hillshade.

5.2.11 Using DERT to Visualize Terrain Models

The open source Desktop Exploration of Remote Terrain (DERT) software tool can be used to explore large digital terrain models, like those created by the Ames Stereo Pipeline. For more information, visit https://github.com/nasa/DERT.

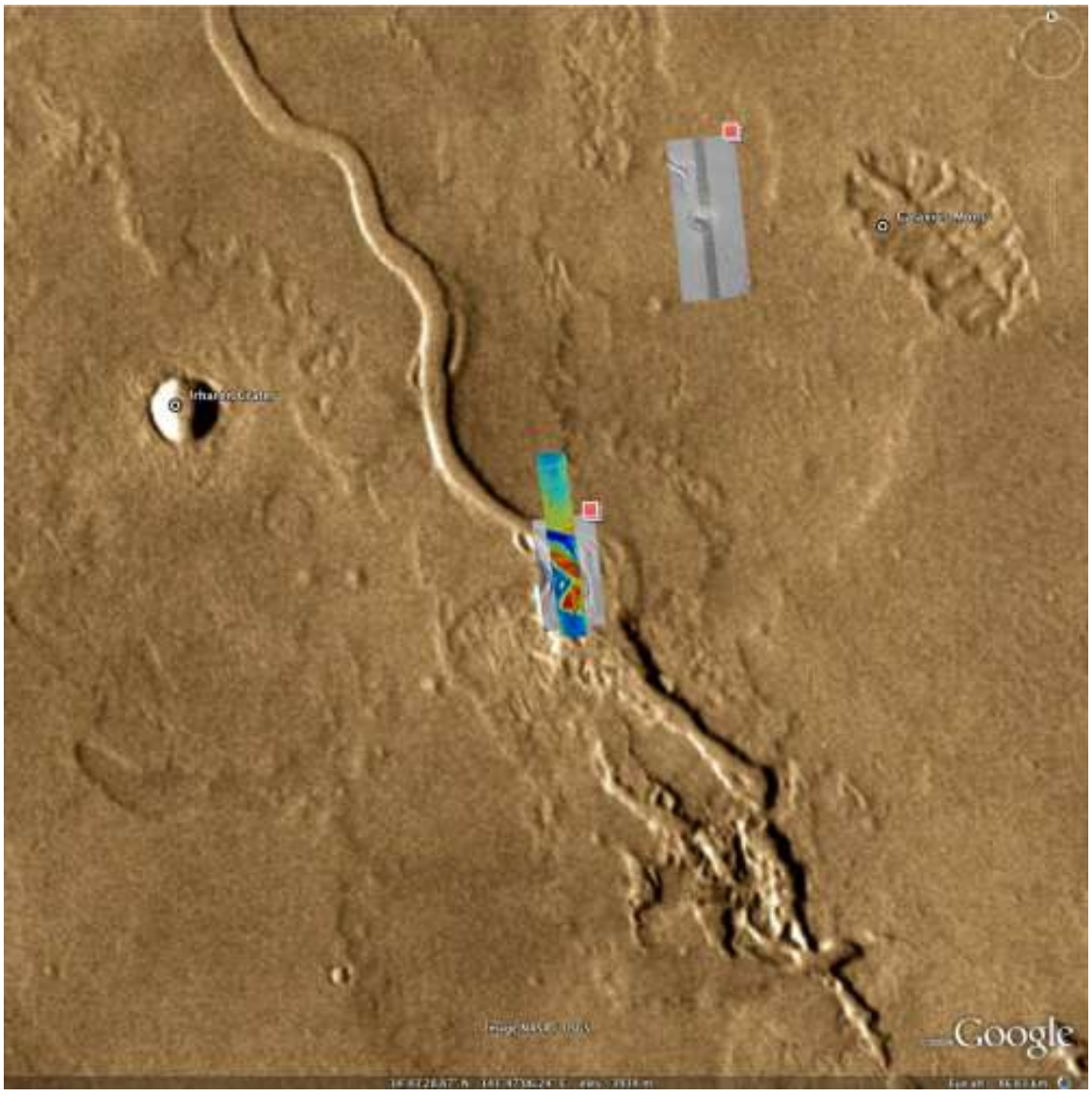


Figure 5.9: The colorized hillshade DEM as a KML overlay.

Chapter 6

Tips and Tricks

Here we summarize, in one place, some insights in how to get the most from ASP, particularly the highest quality results in the smallest amount of time.

- Ask for help or if you have questions. We're always glad to share what we know, implement suggestions, and fix issues (section 1.4).
- Use the GUI (section A.2) to get comfortable with ASP on a small region and to tune parameters (section A.2). A solution specific to ISIS imagery is to crop your stereo pair (using the ISIS `crop` command) to a small region of interest.
- The highest quality results with ASP can be obtained with map-projected images (section 5.1.7).
- Run stereo on multiple machines (section A.3).
- Improve the quality of the inputs to get better outputs. Bundle-adjustment can be used to find out the camera positions more accurately (section 8.2). CCD artifact correction can be used to remove artifacts from WorldView images (section 4.3). Jitter correction can be used for Digital Globe imagery (section 4.4).
- Align the output point cloud to some known absolute reference with `pc_align` (section 5.2.5).
- Remove noise from the output point cloud. During stereo triangulation, points that are further or closer than given distances from planet center or left camera center can be removed as outliers (section B.5). During DEM generation (section A.6), points with large triangulation error can be removed using `--remove-outliers-params`. Spikes can be removed with `--median-filter-params`. Points close to the boundary, that tend to be less accurate, can be eroded (`--erode-length`).
- During stereo filtering, islands can be removed with `--erode-max-size`.
- Remove noise from the low-resolution disparity (D_sub) that can greatly slow down a run using `--rm-quantile-percentile` and `--rm-quantile-multiple`. Some care is needed with these to not remove too much information.
- Fill holes in output orthoimages for nicer display (also in DEMs), during DEM and orthoimage generation with `point2dem` (section A.6). Holes in an existing DEM can also be filled using `dem_mosaic` (section A.8).
- To get good results if the images lack large-scale features (such as for ice plains) use a different way to get the low-resolution disparity (section 4.5).

- If a run takes unreasonably long, decreasing the timeout parameter may be in order (section 5.1.10).
- Manually set the search range if the automated approach fails (section 7.2.2).
- To increase speed, the image pair can be subsampled. For ISIS imagery, the ISIS `reduce` command can be used, while for Digital Globe data one can invoke the `dg_mosaic` tool (section A.11, though note that this tool may introduce aliasing). With subsampling, you are trading resolution for speed, so this probably only makes sense for debugging or "previewing" 3D terrain. That said, subsampling will tend to increase the signal to noise ratio, so it may also be helpful for obtaining 3D terrain out of noisy, low quality images.
- Photometric calibration (using the ISIS tools) can be used to improve the input images and hence get higher quality stereo results.
- If your images have missing or inaccurate camera pose information, and they were acquired with frame (pinhole cameras), such data can be solved for using structure-from-motion and bundle adjustment (chapter 9).
- Shape-from-shading (chapter 10) can be used to further increase the level of detail of a DEM obtained from stereo, though this is a computationally expensive process and its results are not easy to validate.

We'll be happy to add here more suggestions from community's accumulated wisdom on using ASP.

Part II

The Stereo Pipeline in Depth

Chapter 7

Stereo Correlation

In this chapter we will dive much deeper into understanding the core algorithms in the Stereo Pipeline. We start with an overview of the five stages of stereo reconstruction. Then we move into an in-depth discussion and exposition of the various correlation algorithms.

The goal of this chapter is to build an intuition for the stereo correlation process. This will help users to identify unusual results in their DEMs and hopefully eliminate them by tuning various parameters in the `stereo.default` file (appendix B). For scientists and engineers who are using DEMs produced with the Stereo Pipeline, this chapter may help to answer the question, "What is the Stereo Pipeline doing to the raw data to produce this DEM?"

A related question that is commonly asked is, "How accurate is a DEM produced by the Stereo Pipeline?" This chapter does not yet address matters of accuracy and error, however we have several efforts underway to quantify the accuracy of Stereo Pipeline-derived DEMs, and will be publishing more information about that shortly. Stay tuned.

The entire stereo correlation process, from raw input images to a point cloud or DEM, can be viewed as a multistage pipeline as depicted in Figure 7.1, and detailed in the following sections.

7.1 Pre-Processing

The first optional (but recommended) step in the process is least squares Bundle Adjustment, which is described in detail in Chapter 8.

Next, the left and right images are roughly aligned using one of the four methods: (1) a homography transform of the right image based on automated tie-point measurements, (2) an affine epipolar transform of both the left and right images (also based on tie-point measurements as earlier), the effect of which is equivalent to rotating the original cameras which took the pictures, (3) a 3D rotation that achieves epipolar rectification *(only implemented for Pinhole sessions for missions like MER or K10 – see sections 11.5 and 11.6)* or (4) map-projection of both the left and right images using the ISIS `cam2map` command or through the more general `mapproject` tool that works for any cameras supported by ASP (see section 5.1.7 for the latter). The first three options can be applied automatically by the Stereo Pipeline when the `alignment-method` variable in the `stereo.default` file is set to `affineepipolar`, `homography`, or `epipolar`, respectively.

The latter option, running `cam2map`, `cam2map4stereo.py`, or `mapproject` must be carried out by the user prior to invoking the `stereo` command. Map-projecting the images using ISIS eliminates any unusual distortion in the image due to the unusual camera acquisition modes (e.g. pitching "ROTO" maneuvers during image acquisition for MOC, or highly elliptical orbits and changing line exposure times for the High

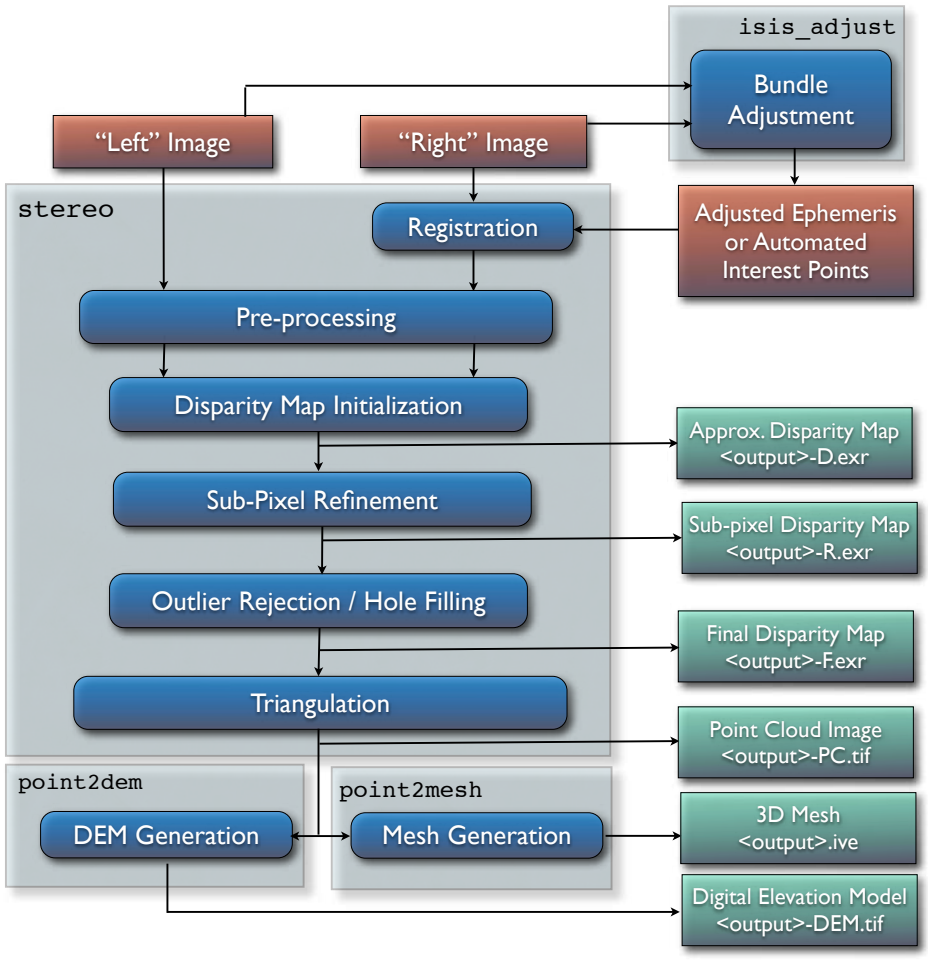


Figure 7.1: Flow of data through the Stereo Pipeline.

Resolution Stereo Camera, HRSC). It also eliminates some of the perspective differences in the image pair that are due to large terrain features by taking the existing low-resolution terrain model into account (e.g., the Mars Orbiter Laser Altimeter, MOLA; Lunar Orbiter Laser Altimeter, LOLA; National Elevation Dataset, NED; or Unified Lunar Coordinate Network, ULCN, 2005 models).

In essence, map-projecting the images results in a pair of very closely matched images that are as close to ideal as possible given existing information. This leaves only small perspective differences in the images, which are exactly the features that the stereo correlation process is designed to detect.

For this reason, we recommend map-projection for pre-alignment of most stereo pairs. Its only cost is longer triangulation times as more math must be applied to work back through the transforms applied to the images. In either case, the pre-alignment step is essential for performance because it ensures that the disparity search space is bounded to a known area. In both cases, the effects of pre-alignment are taken into account later in the process during triangulation, so you do not need to worry that pre-alignment will compromise the geometric integrity of your DEM.

In some cases the pre-processing step may also normalize the pixel values in the left and right images to bring them into the same dynamic range. Various options in the `stereo.default` file affect whether or how normalization is carried out, including `individually-normalize` and `force-use-entire-range`. Although the defaults work in most cases, the use of these normalization steps can vary from data set to data set, so we recommend you refer to the examples in Chapter 11 to see if these are necessary in your use case.

Finally, pre-processing can perform some filtering of the input images (as determined by `prefilter-mode`) to reduce noise and extract edges in the images. When active, these filters apply a kernel with a sigma of `prefilter-kernel-width` pixels that can improve results for noisy images (`prefilter-mode` must be chosen carefully in conjunction with `cost-mode`, see Appendix B). The pre-processing modes that extract image edges are useful for stereo pairs that do not have the same lighting conditions, contrast, and absolute brightness [114]. We recommend that you use the defaults for these parameters to start with, and then experiment only if your results are sub-optimal.

7.2 Disparity Map Initialization

Correlation is the process at the heart of the Stereo Pipeline. It is a collection of algorithms that compute correspondences between pixels in the left image and pixels in the right image. The map of these correspondences is called a *disparity map*. You can think of a disparity map as an image whose pixel locations correspond to the pixel (u, v) in the left image, and whose pixel values contain the horizontal and vertical offsets (d_u, d_v) to the matching pixel in the right image, which is $(u + d_u, v + d_v)$.

The correlation process attempts to find a match for every pixel in the left image. The only pixels skipped are those marked invalid in the mask images. For large images (e.g. from HiRISE, Lunar Reconnaissance Orbiter Camera, LROC, or WorldView), this is very expensive computationally, so the correlation process is split into two stages. The disparity map initialization step computes approximate correspondences using a pyramid-based search that is highly optimized for speed, but trades resolution for speed. The results of disparity map initialization are integer-valued disparity estimates. The sub-pixel refinement step takes these integer estimates as initial conditions for an iterative optimization and refines them using the algorithm discussed in the next section.

We employ several optimizations to accelerate disparity map initialization: (1) a box filter-like accumulator that reduces duplicate operations during correlation [143]; (2) a coarse-to-fine pyramid based approach where disparities are estimated using low-resolution images, and then successively refined at higher resolutions; and (3) partitioning of the disparity search space into rectangular sub-regions with similar values of disparity determined in the previous lower resolution level of the pyramid [143].

Naive correlation itself is carried out by moving a small, rectangular template window from the from left image over the specified search region of the right image, as in Figure 7.2. The "best" match is determined by applying a cost function that compares the two windows. The location at which the window evaluates to the lowest cost compared to all the other search locations is reported as the disparity value. The `cost-mode` variable allows you to choose one of three cost functions, though we recommend normalized cross correlation [94], since it is most robust to slight lighting and contrast variations between a pair of images. Try the others if you need more speed at the cost of quality.

Our implementation of pyramid correlation is a little unique in that it is actually split into two levels of pyramid searching. There is a *`output_prefix`*`-D_sub.tif` disparity image that is computed from the greatly reduced input images `*-L_sub.tif` and *`output_prefix`*`-R_sub.tif`. Those "sub" images have their size chosen so that their area is around 2.25 megapixels, a size that is easily viewed on the screen unlike the raw source imagery. The low-resolution disparity image then defines the per thread search range of the higher resolution disparity, *`output_prefix`*`-D.tif`.

This solution is imperfect but comes from our model of multi-threaded processing. ASP processes individual tiles of the output disparity in parallel. The smaller the tiles, the easier it is to distribute evenly among the CPU cores. The size of the tile unfortunately limits the max number of pyramid levels we can process. We've struck a balance where every 1024 by 1024 pixel area is processed individually in a tile. This practice allows only 5 levels of pyramid processing. With the addition of the second tier of pyramid searching with *`output_prefix`*`-D_sub.tif`, we are allowed to process beyond that limitation.

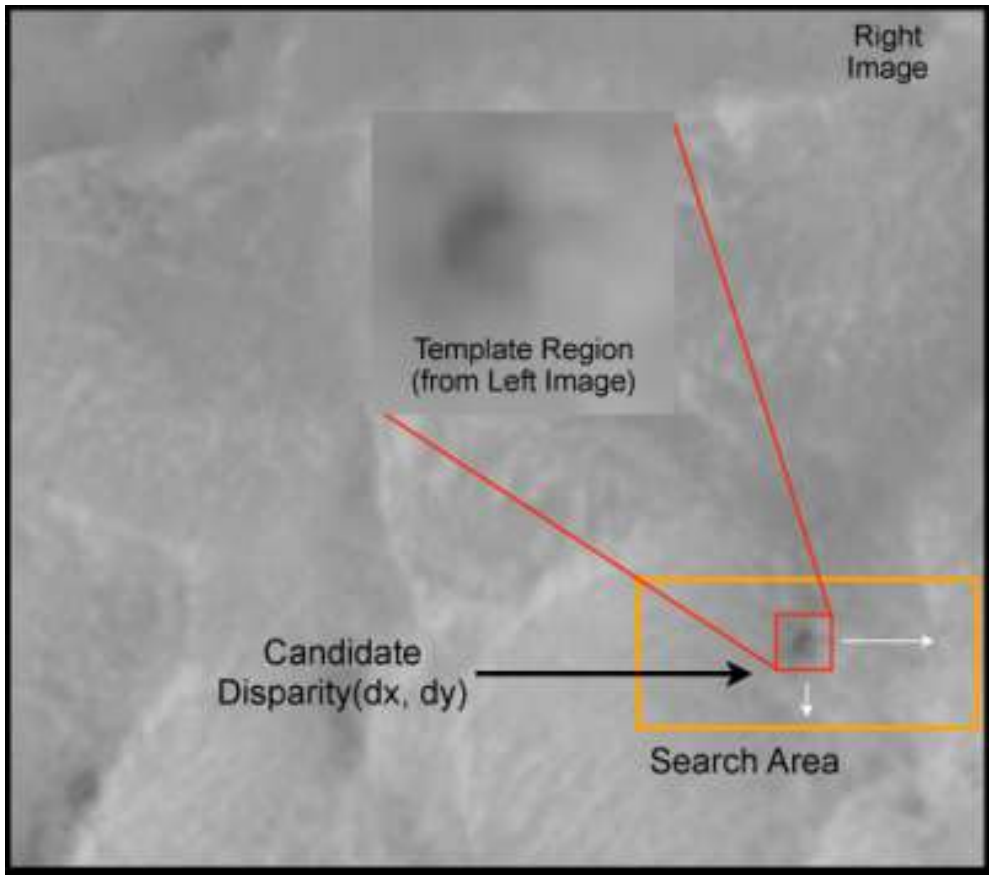


Figure 7.2: The correlation algorithm in disparity map initialization uses a sliding template window from the left image to find the best match in the right image. The size of the template window can be adjusted using the `H_KERN` and `V_KERN` parameters in the `stereo.default` file, and the search range can be adjusted using the {H,V}_CORR_{MIN/MAX} parameters.

Any large failure in the low-resolution disparity image will be detrimental to the performance of the higher resolution disparity. In the event that the low-resolution disparity is completely unhelpful, it can be skipped by adding `corr-seed-mode 0` in the `stereo.default` file and using a manual search range (section 7.2.2). This should only be considered in cases where the texture in an image is completely lost when subsampled. An example would be satellite imagery of fresh snow in the Arctic. Alternatively, *output_prefix*`-D_sub.tif` can be computed at a sparse set of pixels at full resolution, as described in section 4.5.

An alternative to computing *output_prefix*`-D.tif` from sub-sampled images (`corr-seed-mode 1`) or skipping it altogether (`corr-seed-mode 0`), is to compute it from a lower-resolution DEM of the area (`corr-seed-mode 2`). In this situation, the low-resolution DEM needs to be specified together with its estimated error. See section B.2 for more detailed information as to how to specify these options. In our experiments, if the input DEM has a resolution of 1 km, a good value for the DEM error is about 10 m, or higher if the terrain is very variable.

7.2.1 Debugging Disparity Map Initialization

Never will all pixels be successfully matched during stereo matching. Though a good chunk of the image should be correctly processed. If you see large areas where matching failed, this could be due to a variety of reasons:

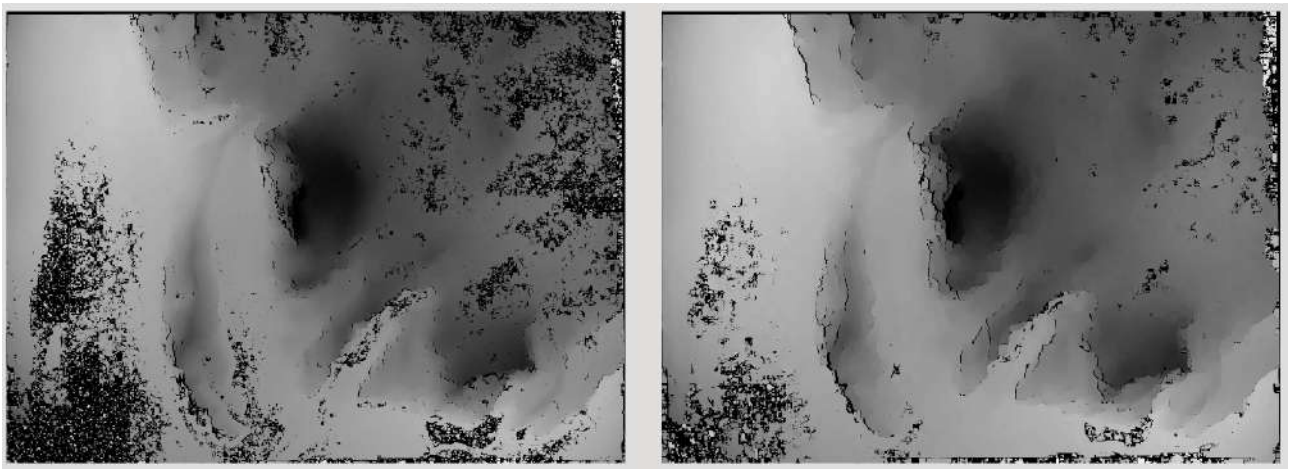


Figure 7.3: The effect of increasing the correlation kernel size from 35 (left) to 75 (right). This location is covered in snow and several regions lack texture for the correlator to use but a large kernel increases the chances of finding useful texture for a given pixel.

- In regions where the images do not overlap, there should be no valid matches in the disparity map.
- Match quality may be poor in regions of the images that have different lighting conditions, contrast, or specular properties of the surface.
- Areas that have image content with very little texture or extremely low contrast may have an insufficient signal to noise ratio, and will be rejected by the correlator.
- Areas that are highly distorted due to different image perspective, such as crater and canyon walls, may exhibit poor matching performance. This could also be due to failure of the preprocessing step in aligning the images. The correlator can not match images that are rotated differently from each other or have different scale/resolution. Mapprojection is used to at least partially rectify these issues (section 5.1.7).

Bad matches, often called "blunders" or "artifacts" are also common, and can happen for many of the same reasons listed above. The Stereo Pipeline does its best to automatically detect and eliminate these blunders, but the effectiveness of these outlier rejection strategies does vary depending on the quality of the input imagery.

When tuning up your `stereo.default` file, you will find that it is very helpful to look at the raw output of the disparity map initialization step. This can be done using the `disparitydebug` tool, which converts the *`output_prefix`*`-D.tif` file into a pair of normal images that contain the horizontal and vertical components of disparity. You can open these in a standard image viewing application and see immediately which pixels were matched successfully, and which were not. Stereo matching blunders are usually also obvious when inspecting these images. With a good intuition for the effects of various `stereo.default` parameters and a good intuition for reading the output of `disparitydebug`, it is possible to quickly identify and address most problems.

If you are seeing too many holes in your disparity images, one option that may give good results is to increase the size of the correlation kernel used by `stereo_corr` with the `-corr-kernel` option. Increasing the kernel size will increase the processing time but should help fill in regions of the image where no match was found.

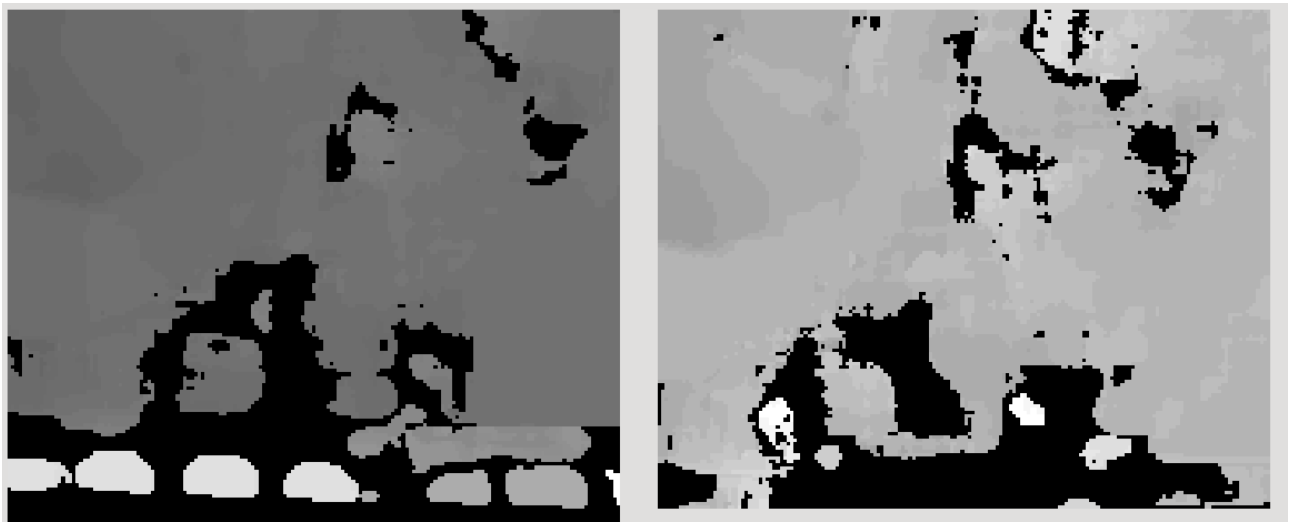


Figure 7.4: The effect of using the `rm-quantile` filtering option in `stereo_corr`. In the left image there are a series of high disparity "islands" at the bottom of the image. In the right image quantile filtering has removed those islands while leaving the rest of the image intact.

7.2.2 Search Range Determination

In some circumstances, the low-resolution disparity `D_sub.tif` may fail to get computed, or it may be inaccurate. This can happen for example if only very small features are present in the original images, and they disappear during the resampling that is necessary to obtain `D_sub.tif`. In this case, it is possible to set `corr-seed-mode` to 0, and manually set a search range to use for full-resolution correlation via the parameter `corr-search`. In `stereo.default` this parameter's entry will look like:

```
corr-search -80 -2 20 2
```

The exact values to use with this option you'll have to discover yourself. The numbers right of `corr-search` represent the horizontal minimum boundary, vertical minimum boundary, horizontal maximum boundary, and finally the horizontal maximum boundary within which we will search for the disparity during correlation.

It can be tricky to select a good search range for the `stereo.default` file. That's why the best way is to let `stereo` perform an automated guess for the search range. If you find that you can do a better estimate of the search range, take look at the intermediate disparity images using the `disparitydebug` program to figure out which search directions can be expanded or contracted. The output images will clearly show good data or bad data depending on whether the search range is correct.

The worst case scenario is to determine the search range manually. For example, for ISIS images, both images could be opened in `qview` and the coordinates of points that can be matched visually can be compared. Subtract line,sample locations in the first image from the coordinates of the same feature in the second image, and this will yield offsets that can be used in the search range. Make several of these offset measurements and use them to define a line,sample bounding box, then expand this by 50% and use it for `corr-search`. This will produce good results in most images.

Also, if you are using an alignment option, you'll instead want to make those disparity measurements against the written L.tif and R.tif files (see chapter C) instead of the original input files.

7.2.3 Local Homography

Local homography decomposes the left image into tiles, and tries to find the best homography transform from each tile to the right image before computing the correlation. This is more fine-grained than using a global homography transform.

This approach is experimental. We suggest instead the map-projection approach be used (section 5.1.7), as that one is even more fine-grained, and does not suffer from artifacts that may arise from the local homography piecewise approach.

This option can be turned on with the flag `use-local-homography`.

7.2.4 Semi-Global Matching

A new option for integer stereo correlation available in ASP is the popular semi-global matching algorithm introduced in [58]. The algorithm is not typically used for DEM generation but it has been used successfully to process HRSC images [59]. The version of the algorithm implemented by ASP has a few modifications relative to the original implementation. The most significant difference is that ASP's implementation performs a 2D disparity search, similar to what is done in the NG-fSGM algorithm [156]. Since ASP processes a wide variety of cameras with varying degrees of metadata quality, the standard assumption with SGM that the disparity search can be performed only along a one-dimensional epipolar line does not hold. The other major change is that ASP uses a multi- resolution hierarchical search combined with a compressed memory scheme similar to what is used in the SGM algorithm [131]. With these two features the SGM algorithm can be used for unrectified, larger images. ASP also supports a mode using the MGM algorithm [33], referred to in some places in the documentation as Smooth SGM. This algorithm reduces the amount of high frequency artifacts in textureless regions at the cost of a longer run time. ASP also offers the option of a hybrid SGM/MGM mode where MGM is used only for the final resolution level which obtains results somewhere between the pure SGM and MGM options.

The greatest advantage of the SGM algorithm over the normal ASP correlation algorithm is an improved ability to find disparity matches in areas of repetitive or low texture. SGM can also discern finer resolution features than the standard correlation algorithm since it tends to use much smaller matching kernels. Along with these advantages come several disadvantages. First, SGM is computationally expensive and requires a lot of memory. Second, in some situations it can produce noticeable artifacts at tile boundaries. Third, it can sometimes produce inaccurate results in textureless regions. With careful parameter selection and usage these disadvantages can be mitigated.

In order to use SGM, pass in `stereo-algorithm`. Use 1 to use SGM or 2 to use MGM. To process large images you must use the `parallel_stereo` program instead of the `stereo` program. `parallel_stereo` replaces the refinement stage with a new seam blending stage to suppress artifacts along tile borders. Without this step SGM can produce artifacts along tile borders. The `stereo` program can be used as long as the `corr-tile-size` command is set large enough to fit the entire image into a single processing tile. When running SGM, a single ASP process will handle only one tile at a time but it will use multiple threads per tile, as opposed to normal stereo where each tile uses its own thread. MGM is currently limited to using 8 simultaneous threads but SGM does not have a limit. When running `parallel_stereo` use the following options:

- Specify the `sgm-collar-size` option or leave it at the default value. Increasing this value decreases the chances of seeing artifacts along tile borders but increases processing time and memory usage.
- Set the `corr-tile-size` option to determine the tile size, keeping in mind that larger tile sizes produce better results but consume more memory. The collar size you selected will enlarge the processed tile size.

- Set the `processes` option keeping in mind memory constraints as discussed earlier. Each process will run one simultaneous SGM instance and consume memory.
- The `corr-memory-limit-mb` parameter limits the number of megabytes of memory that can be used by SGM/MGM. This limit is per-process. To be safe, make sure that you have more RAM available than the value of this parameter multiplied by the number of processes.
- `job-size-w` and `job-size-h` are set equal to `corr-tile-size`. Do not override them!

By setting these parameters in the manner described, each process will generate a single SGM tile which will then be blended in the new blend step. Each process can use multiple threads with `threads-singleprocess` without affecting the stereo results.

When SGM or MGM is specified, certain stereo parameters have their default values replaced with values that will work with SGM. You can still manually specify these options.

- Cost Mode (default 4). Mean absolute distance (MAD) (`cost-mode <= 2`) usually does not work well. The census transform mode (`cost-mode 3`) [159] tends to perform better overall but can produce artifacts on featureless terrain. The ternary census transform mode (`cost-mode 4`) [60] is a modification of the census transform that is more stable on low contrast terrain but may be less accurate elsewhere.
- Kernel size. SGM kernels must always be symmetric. The SGM algorithm works with much smaller kernel sizes than the regular integer correlator so the default large kernel is not recommended. The MAD cost mode can be used with any odd kernel size (including size 1) but the census cost modes can only be used with kernel sizes 3, 5, 7, and 9. Size 7 is usually a good choice.
- Xcorr-Threshold. By default, this is disabled in order to nearly halve the (long) run time of the SGM algorithm. Set `xcorr-threshold` to $>= 0$ to turn it back on. If you set the `min-xcorr-level` parameter to 1 you can perform cross correlation on the smaller resolution levels without spending the time to run it on the largest resolution level.
- The median and texture filters in the `stereo_fltr` tool (defaults 3, 11, 0.13). These filters were designed specifically to clean up output from the SGM algorithm and are especially useful in suppressing image artifacts in low-texture portions of the image. A median filter size of 3 and a texture filter size of 11 are good starts but the best values will depend on your input images. The `texture-smooth-scale` parameter will have to be adjusted to taste, but a range of 0.13 to 0.15 is typical for icy images. These values are enabled by default and must be manually disabled. If your images have good texture throughout it may be best to disable these filters.
- The `prefilter-mode` setting is ignored when using SGM.
- The `subpixel-mode` If not set or set to values 7-12 SGM will perform subpixel interpolation during the stereo correlation step and will not do additional work in the stereo refinement step. This means that after dealing with the long SGM processing time you do not need to follow it up with a slow subpixel option! If desired, you can specify modes 1-4 to force those subpixel operations to be performed after the default SGM subpixel method.

Figure 7.5 shows a comparison between two stereo modes. The DEM on the left was generated using the default stereo parameters and `--subpixel-mode 3`. The DEM on the right was generated using the command:

```
stereo --stereo-algorithm 1 --threads 1 --xcorr-threshold -1 --corr-kernel 7 7 \
--corr-tile-size 6400 --cost-mode 4 --median-filter-size 3   \
--texture-smooth-size 13 --texture-smooth-scale 0.13
```

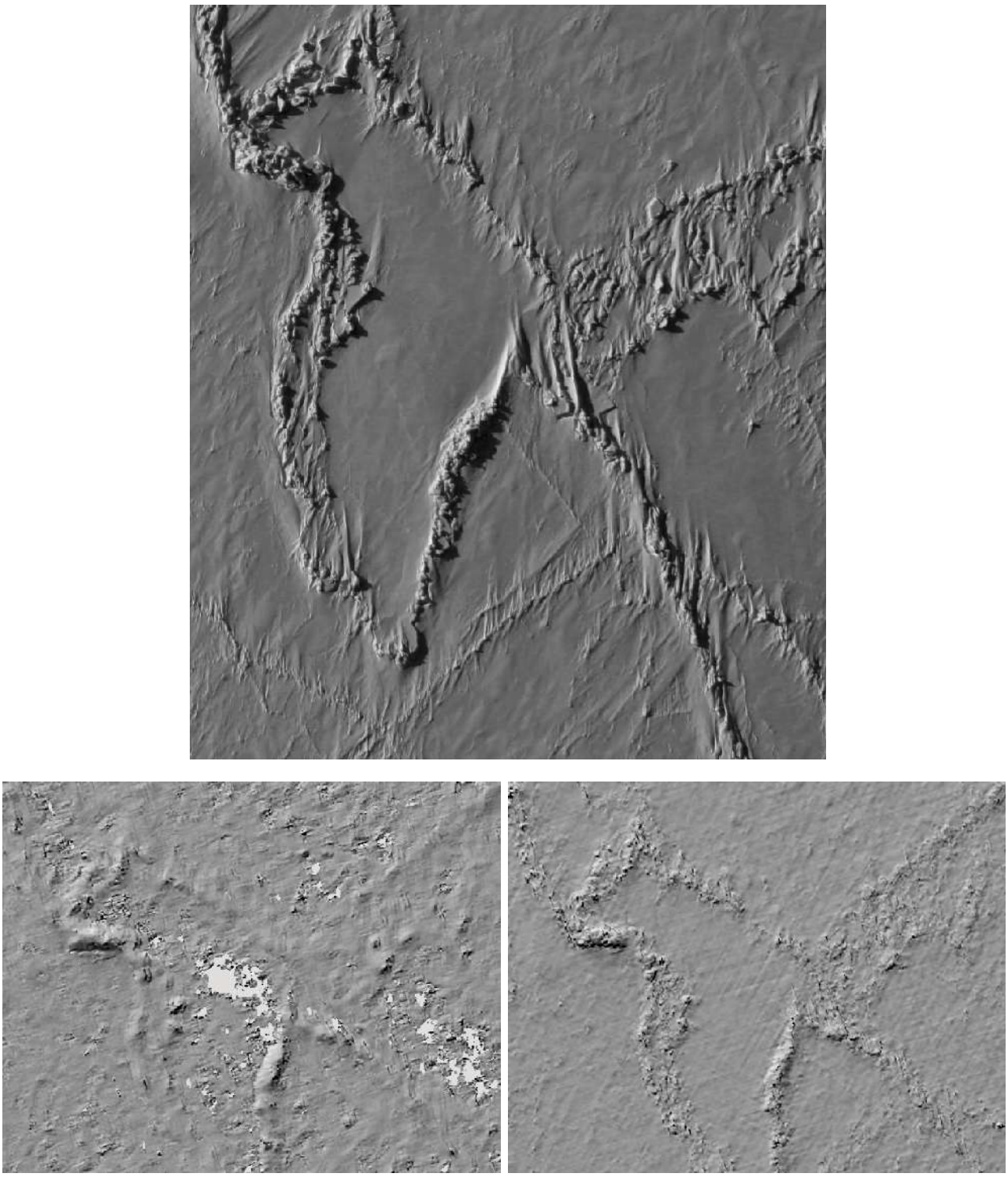


Figure 7.5: A section of a NASA IceBridge image with a pair of hill-shaded DEMs below it showing the difference between default ASP processing and processing using the SGM algorithm.

Some grid pattern noise is visible in the image produced using SGM. Using `--stereo-algorithm 2` should reduce it. And, as mentioned earlier, for large images which won't fit in memory, `--corr-tile-size` can be set to a value like 4096, and `parallel_stereo` should be used.

7.3 Sub-pixel Refinement

Once disparity map initialization is complete, every pixel in the disparity map will either have an estimated disparity value, or it will be marked as invalid. All valid pixels are then adjusted in the sub-pixel refinement stage based on the `subpixel-mode` setting.

The first mode is parabola-fitting sub-pixel refinement (`subpixel-mode 1`). This technique fits a 2D parabola to points on the correlation cost surface in an 8-connected neighborhood around the cost value that was the "best" as measured during disparity map initialization. The parabola's minimum can then be computed analytically and taken as as the new sub-pixel disparity value.

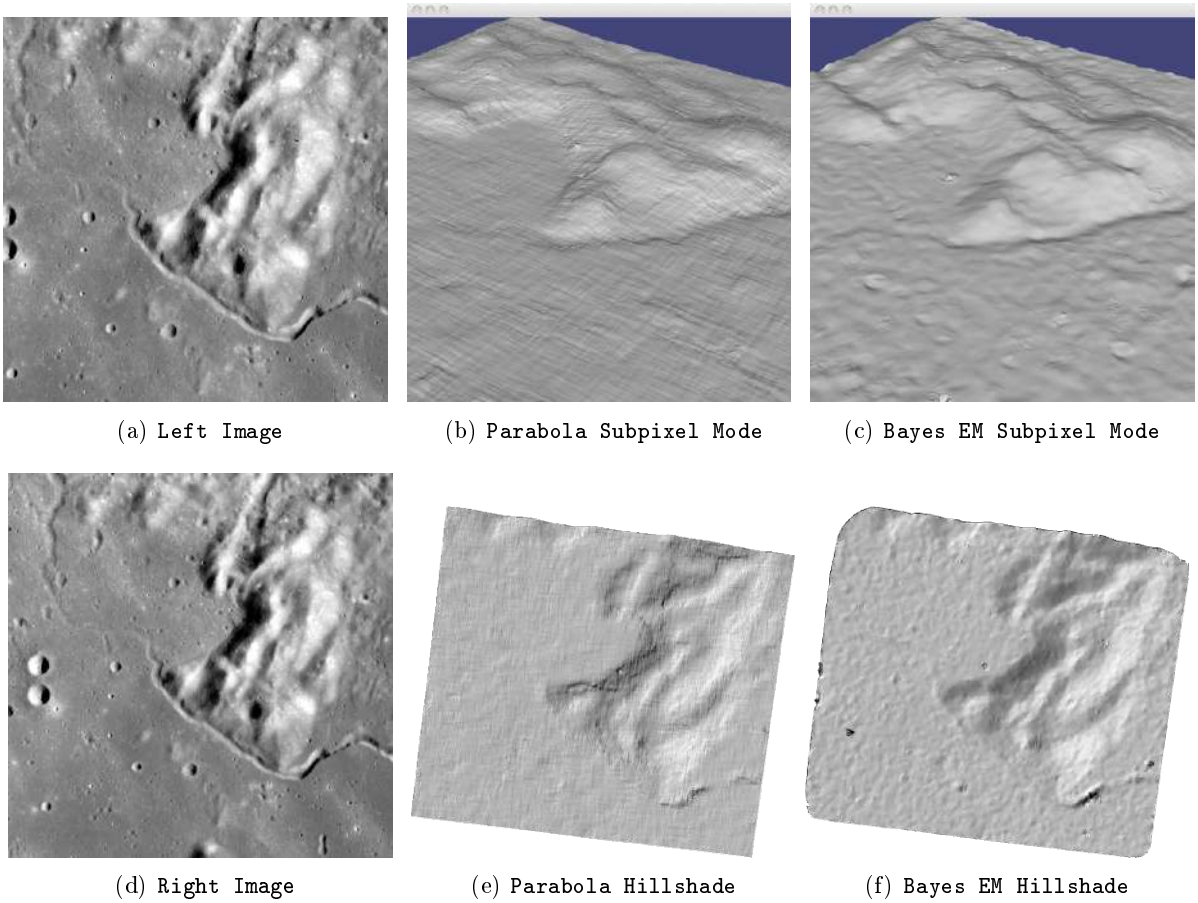
This method is easy to implement and extremely fast to compute, but it exhibits a problem known as pixel-locking: the sub-pixel disparities tend toward their integer estimates and can create noticeable "stair steps" on surfaces that should be smooth [140, 145]. See for example Figure 7.6(b). Furthermore, the parabola subpixel mode is not capable of refining a disparity estimate by more than one pixel, so although it produces smooth disparity maps, these results are not much more accurate than the results that come out of the disparity map initialization in the first place. However, the speed of this method makes it very useful as a "draft" mode for quickly generating a DEM for visualization (i.e. non-scientific) purposes. It is also beneficial in the event that a user will simply downsample their DEM after generation in Stereo Pipeline.

For high quality results, we recommend `subpixel-mode 2`: the Bayes EM weighted affine adaptive window correlator. This advanced method produces extremely high quality stereo matches that exhibit a high degree of immunity to image noise. For example Apollo Metric Camera images are affected by two types of noise inherent to the scanning process: (1) the presence of film grain and (2) dust and lint particles present on the film or scanner. The former gives rise to noise in the DEM values that wash out real features, and the latter causes incorrect matches or hard to detect blemishes in the DEM. Attenuating the effect of these scanning artifacts while simultaneously refining the integer disparity map to sub-pixel accuracy has become a critical goal of our system, and is necessary for processing real-world data sets such as the Apollo Metric Camera data.

The Bayes EM subpixel correlator also features a deformable template window from the left image that can be rotated, scaled, and translated as it zeros in on the correct match in the right image. This adaptive window is essential for computing accurate matches on crater or canyon walls, and on other areas with significant perspective distortion due to foreshortening.

This affine-adaptive behavior is based on the Lucas-Kanade template tracking algorithm, a classic algorithm in the field of computer vision [10]. We have extended this technique; developing a Bayesian model that treats the Lucas-Kanade parameters as random variables in an Expectation Maximization (EM) framework. This statistical model also includes a Gaussian mixture component to model image noise that is the basis for the robustness of our algorithm. We will not go into depth on our approach here, but we encourage interested readers to read our papers on the topic [109, 21].

However we do note that, like the computations in the disparity map initialization stage, we adopt a multi-scale approach for sub-pixel refinement. At each level of the pyramid, the algorithm is initialized with the disparity determined in the previous lower resolution level of the pyramid, thereby allowing the subpixel algorithm to shift the results of the disparity initialization stage by many pixels if a better match can be found using the affine, noise-adapted window. Hence, this sub-pixel algorithm is able to significantly improve upon the results to yield a high quality, high resolution result.



(a) `Left Image` (b) `Parabola Subpixel Mode` (c) `Bayes EM Subpixel Mode`

(d) `Right Image` (e) `Parabola Hillshade` (f) `Bayes EM Hillshade`

Figure 7.6: Left: Input images. Center: results using the parabola draft subpixel mode (`subpixel-mode = 1`). Right: results using the Bayes EM high quality subpixel mode (`subpixel-mode = 2`).

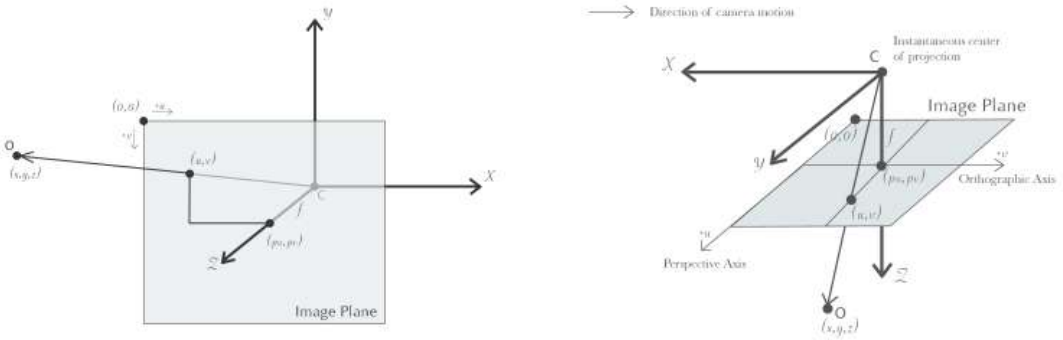
Another option when run time is important is `subpixel-mode 3`: the simple affine correlator. This is essentially the Bayes EM mode with the noise correction features removed in order to decrease the required run time. In data sets with little noise this mode can yield results similar to Bayes EM mode in approximately one fifth the time.

A different option is Phase Correlation, `subpixel-mode 4`, which implements the algorithm from [47]. It is slow and does not work well on slopes but since the algorithm is very different it might perform in situations where the other algorithms are not working well.

7.4 Triangulation

When running an ISIS session, the Stereo Pipeline uses geometric camera models available in ISIS [7]. These highly accurate models are customized for each instrument that ISIS supports. Each ISIS "cube" file contains all of the information that is required by the Stereo Pipeline to find and use the appropriate camera model for that observation.

Other sessions such as DG (*Digital Globe*) or Pinhole, require that their camera model be provided as additional arguments to the `stereo` command. Those camera models come in the form of an XML document for DG and as `*.pinhole, *.tsai, *.cahv, *.cahvor` for Pinhole sessions. Those files must be the third and forth arguments or immediately follow after the 2 input images for `stereo`.



(a) `Framing Camera Model` (b) `Pushbroom Camera Model`

Figure 7.7: Most remote sensing cameras fall into two generic categories based on their basic geometry. Framing cameras (left) capture an instantaneous two-dimensional image. Linescan cameras (right) capture images one scan line at a time, building up an image over the course of several seconds as the satellite moves through the sky.

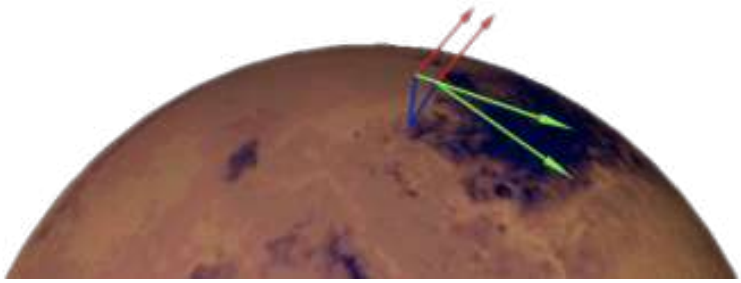


Figure 7.8: Once a disparity map has been generated and refined, it can be used in combination with the geometric camera models to compute the locations of 3D points on the surface of Mars. This figure shows the position (at the origins of the red, green, and blue vectors) and orientation of the Mars Global Surveyor at two points in time where it captured images in a stereo pair.

ISIS camera models account for all aspects of camera geometry, including both intrinsic (i.e. focal length, pixel size, and lens distortion) and extrinsic (e.g. camera position and orientation) camera parameters. Taken together, these parameters are sufficient to "forward project" a 3D point in the world onto the image plane of the sensor. It is also possible to "back project" from the camera's center of projection through a pixel corresponding to the original 3D point.

Notice, however, that forward and back projection are not symmetric operations. One camera is sufficient to "image" a 3D point onto a pixel located on the image plane, but the reverse is not true. Given only a single camera and a pixel location $x = (u, v)$, that is the image of an unknown 3D point $P = (x, y, z)$, it is only possible to determine that P lies somewhere along a ray that emanates from the camera's center of projection through the pixel location x on the image plane (see Figure 7.7).

Alas, once images are captured, the route from image pixel back to 3D points in the real world is through back projection, so we must bring more information to bear on the problem of uniquely reconstructing our 3D point. In order to determine P using back projection, we need *two* cameras that both contain pixel locations x_1 and x_2 where P was imaged. Now, we have two rays that converge on a point in 3D space (see Figure 7.8). The location where they meet must be the original location of P.

In practice, the two rays rarely intersect perfectly because any slight error in the camera position or pointing information will effect the rays' positions as well. Instead, we take the *closest point of intersection* of the two rays as the location of point P.

Additionally, the actual distance between the rays at this point is an interesting and important error metric that measures how self-consistent our two camera models are for this point. You will learn in the next chapter that this information, when computed and averaged over all reconstructed 3D points, can be a valuable statistic for determining whether to carry out bundle adjustment. Distance between the two rays at their closest intersection is recorded in the fourth channel of the point cloud file, *output-prefix*`-PC.tif`. This information can be brought to the same perspective as the output DEM by using the *--error* argument on the `point2dem` command.

This error in the triangulation, the distance between two rays, *is not the true accuracy of the DEM.* It is only another indirect measure of quality. A DEM with high triangulation error is always bad and should have its images bundle-adjusted. A DEM with low triangulation error is at least self consistent but could still be bad. A map of the triangulation error should only be interpreted as a relative measurement. Where small areas are found with high triangulation error came from correlation mistakes and large areas of error came from camera model inadequacies.

Chapter 8

Bundle Adjustment

8.1 Overview

Satellite position and orientation errors have a direct effect on the accuracy of digital elevation models produced by the Stereo Pipeline. If they are not corrected, these uncertainties will result in systematic errors in the overall position and slope of the DEM. Severe distortions can occur as well, resulting in twisted or "taco shaped" DEMs, though in most cases these effects are quite subtle and hard to detect. In the worst case, such as with old mission data like Voyager or Apollo, these gross camera misalignments can inhibit Stereo Pipeline's internal interest point matcher and block auto search range detection.

Errors in camera position and orientation can be corrected using a process called *bundle adjustment*. Bundle adjustment is the process of simultaneously adjusting the properties of many cameras and the 3D locations of the objects they see in order to minimize the error between the estimated, back-projected pixel locations of the 3D objects and their actual measured locations in the captured images.

This complex process can be boiled down to this simple idea: bundle adjustment ensures that the observations in multiple images of a single ground feature are self-consistent. If they are not consistent, then the position and orientation of the cameras as well as the 3D position of the feature must be adjusted until they are. This optimization is carried out along with thousands (or more) of similar constraints involving many different features observed in other images. Bundle adjustment is very powerful and versatile: it can operate on just two overlapping images, or on thousands. It is also a dangerous tool. Careful consideration

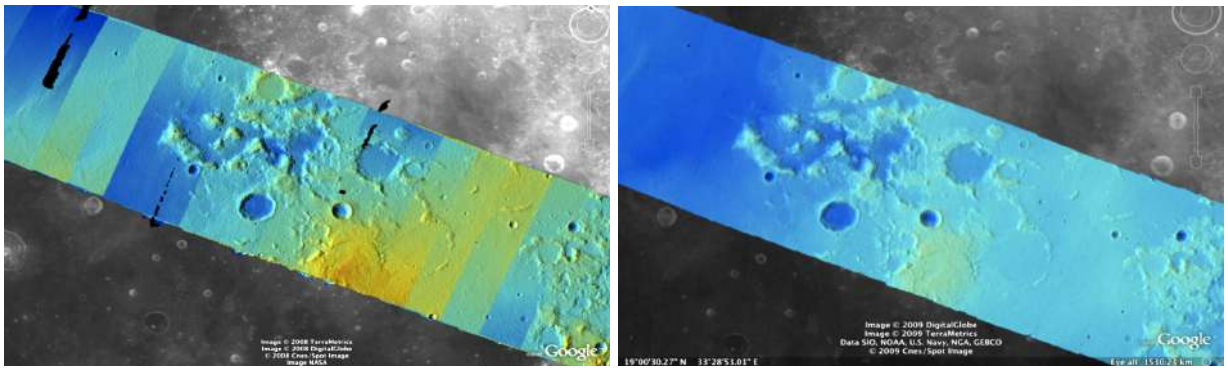


Figure 8.1: Bundle adjustment is illustrated here using a color-mapped, hill-shaded DEM mosaic from Apollo 15, Orbit 33, imagery. (a) Prior to bundle adjustment, large discontinuities can exist between overlapping DEMs made from different images. (b) After bundle adjustment, DEM alignment errors are minimized and no longer visible.

is required to insure and verify that the solution does represent reality.

Bundle adjustment can also take advantage of Ground control points (GCPs), which are 3D locations of features that are known a priori (often by measuring them by hand in another existing DEM). GCPs can improve the internal consistency of your DEM or align your DEM to an existing data product. Finally, even though bundle adjustment calculates the locations of the 3D objects it views, only the final properties of the cameras are recorded for use by the Ames Stereo Pipeline. Those properties can be loaded into the `stereo` program which uses its own method for triangulating 3D feature locations.

When using the Stereo Pipeline, bundle adjustment is an optional step between the capture of images and the creation of DEMs. The bundle adjustment process described below should be completed prior to running the `stereo` command.

Although bundle adjustment is not a required step for generating DEMs, it is *highly recommended* for users who plan to create DEMs for scientific analysis and publication. Incorporating bundle adjustment into the stereo work flow not only results in DEMs that are more internally consistent, it is also the correct way to co-register your DEMs with other existing data sets and geodetic control networks.

At the moment however, Bundle Adjustment does not automatically work against outside DEMs from sources such as laser altimeters. Hand-picked GCPs are the only way for ASP to register to those types of sources.

8.2 Bundle adjustment using ASP

Stereo Pipeline provides its own bundle adjustment tool, named `bundle_adjust`. Its usage is described in section A.4.

Here is an example of using this tool on a couple of Apollo 15 images, and its effect on decreasing the stereo triangulation error.

Running `stereo` without using bundle-adjusted camera models.

```
stereo AS15-M-1134.cub AS15-M-1135.cub run_noadjust/run
```

Performing bundle adjustment.

```
bundle_adjust AS15-M-1134.cub AS15-M-1135.cub -o run_ba/run
```

Running stereo while using the bundle-adjusted camera models.

```
stereo AS15-M-1134.cub AS15-M-1135.cub run_adjust/run \
  --bundle-adjust-prefix run_ba/run
```

A comparison of the two ways of doing stereo is shown in figure 8.2.

ASP also offers the tool `parallel_bundle_adjust` which can be much faster bundle adjusting many images at once.

8.2.1 Floating intrinsics and using a lidar or DEM ground truth

This section documents some advanced functionality, and it suggested the reader study it carefully and invest a certain amount of time to fully take advantage of these concepts.

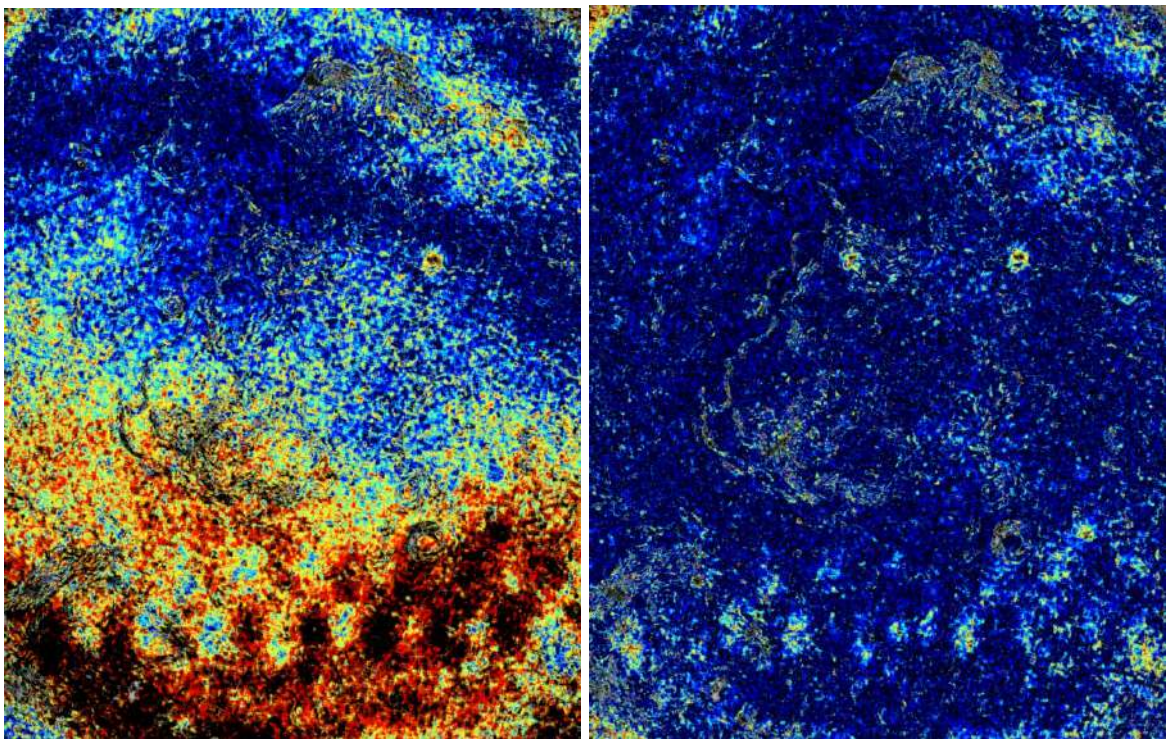


Figure 8.2: Illustration of the triangulation error map for a pair of images before (left) and after (right) using Stereo Pipeline's `bundle_adjust`. Red and black colors suggest higher error.

When the input cameras are of pinhole type, it is possible to optimize the intrinsic parameters, in addition to the extrinsics. It is also possible to take advantage of an existing terrain ground truth, such as a lidar file or a DEM, to correct imperfectly calibrated intrinsic parameters, which can result in greatly improved results, such as creating less distorted DEMs that agree much better with the ground truth.

A first attempt at floating the intrinsics

We recommend that first bundle adjustment is run with the intrinsics fixed, to get the extrinsics mostly correct, as optimizing for both of them at the same time may result in a non-convex problem which may lead to a suboptimal local minimum. Then, we will jointly optimize the intrinsics and extrinsics.

Note that when solving for intrinsics, `bundle_adjust` will by default optimize all intrinsic parameters and will share them across all cameras (which must be the same type). You can control this behavior with the `--intrinsics-to-float` and `--intrinsics-to-share` parameters.

Hence, the first invocation of camera optimization should be like:

```
bundle_adjust -t nadirpinhole --inline-adjustments      \
  left.tif right.tif left.tsai right.tsai -o run_ba/run
```

It is suggested that one run `stereo` with the obtained cameras, and then examine the intersection error:

```
stereo -t nadirpinhole --alignment-method epipolar left.tif right.tif \
   run_ba/run-left.tsai run_ba/run-right.tsai run_stereo/run
point2dem --tr RESOLUTION --errorimage run_stereo/run-PC.tif
```

```
gdalinfo -stats run_stereo/run-IntersectionErr.tif
colormap run_stereo/run-IntersectionErr.tif
stereo_gui run_stereo/run-IntersectionErr_CMAP.tif
```

If desired, fancier stereo correlation algorithms can be used, such as MGM, as detailed in chapter 7. For `colormap`, `--min` and `--max` bounds can be specified if the automatic range is too large. We also suggest inspecting the interest points:

```
stereo_gui left.tif right.tif run_ba/run
```

and then viewing the interest points from the menu.

If the interest points are not well-distributed, this may result in large ray intersection errors where they are missing. If so, they can be re-created by modifying `--ip-detect-method` and `--ip-per-tile`. Or, one can take advantage of the just-completed stereo run and invoke `stereo_tri` with the additional option

```
--num-matches-from-disp-triplets 10000
```

to create dense and uniformly distributed interest points with desired density (the latter creates a .match file that needs to be copied to the name `bundle_adjust` expects). This option also ensures that if three images are present, and stereo is invoked on the first and second image, and then on the second and the third, followed by interest point generation, many interest points will be triplets, that is, the same feature will often will be identified in all three images, which can be a very good constraint on bundle adjustment later.

If the interest points are good and the mean intersection error is acceptable, but this error shows an odd nonlinear pattern, that means it may be necessary to optimize the intrinsics. We do so by using the cameras with the optimized extrinsics found earlier, that is:

```
bundle_adjust -t nadirpinhole --inline-adjustments               \
  --solve-intrinsics --camera-weight 1                           \
  left.tif right.tif run_ba/run-left.tsai run_ba/run-right.tsai  \
  -o run_ba_intr/run
```

It is important to note that only the non-zero intrinsics will be optimized, and the step size used in optimizing a certain intrinsic parameter is proportional to it. Hence, if an intrinsic is 0 and it is desired to optimize it, it should be set to small non-zero value suggestive of its final estimated scale. If the algorithm fails to give a good solution, perhaps different initial values for the intrinsics should be tried. For example, one can try changing the sign of the initial distortion coefficients, or make their values much smaller.

Sometimes the camera weight may need to be decreased, even all the way to 0, if it appears that the solver is not aggressive enough, or it may need to be increased if perhaps it overfits. This will become less of a concern if there is some ground truth, as discussed later.

Next, one can run stereo as before, with the new cameras, and see if the obtained solution is more acceptable, that is, if the intersection error is smaller. It is good to note that a preliminary investigation can already be made right after bundle adjustment, by looking at the residual error files before and after bundle adjustment. They are in the output directory, with names containing the strings

```
initial_residuals_no_loss_function_pointmap
final_residuals_no_loss_function_pointmap
```

If desired, these csv files can be converted to a DEM with `point2dem`, which can be invoked with

```
--csv-format 1:lon,2:lat,4:height_above_datum
```

then one can look at their statistics, also have them colorized, and viewed in `stereo_gui`.

This file also shows how often each feature is seen in the images, so, if three images are present, hopefully many features will be seen three times.

Using ground truth when floating the intrinsics

If a ground truth lidar file (or DEM) is present, say named `lidar.csv`, it can be used as part of bundle adjustment. For that, the DEM obtained with the earlier stereo pass needs to be first aligned to this ground truth, such as:

```
pc_align --max-displacement VAL run_stereo/run-DEM.tif lidar.csv -o run_align/run
```

(see the manual page of this tool in section A.21 for more details).

This alignment can then be applied to the cameras as well:

```
bundle_adjust -t nadirpinhole --inline-adjustments --max-iterations 0  \
  --initial-transform run_align/run-inverse-transform.txt              \
  left.tif right.tif run_ba/run-left.tsai run_ba/run-right.tsai        \
  -o run_align/run
```

Here we have used 0 iterations because we simply want to move the cameras without any optimization. Note that your lidar file may have some conventions as to what each column means, and then any tools that use this cloud must set `--csv-format` and perhaps also `--datum` and/or `--csv-proj4`.

If `pc_align` is called with the clouds in reverse order (the denser cloud should always be the first), when applying the transform to the cameras in `bundle_adjust` one should use `transform.txt` instead of `inverse-transform.txt` above.

Next, we will need to create a disparity from the left and right images that we will use during bundle adjustment. For that we will take the disparity obtained in stereo and remove any intermediate transforms stereo applied to the images and the disparity. This can be done as follows:

```
stereo_tri -t nadirpinhole --alignment-method epipolar left.tif right.tif  \
  run_ba/run-left.tsai run_ba/run-right.tsai run_stereo/run                 \
  --unalign-disparity
```

and then bundle adjustment can be invoked with this disparity and the lidar/DEM file. Note that we use the cameras obtained after alignment:

```
bundle_adjust -t nadirpinhole --inline-adjustments --solve-intrinsics              \
  left.tif right.tif run_align/run-run-left.tsai run_align/run-run-right.tsai      \
  --reference-terrain lidar.csv --disparity-list run_stereo/run-unaligned-D.tif    \
  --camera-weight 0 --max-disp-error 50 --max-num-reference-points 1000000         \
  --parameter-tolerance 1e-12 --reference-terrain-weight 5 -o run_ba_intr_lidar/run
```

Here we set the camera weight all the way to 0, since it is hoped that having a reference terrain is a sufficient constraint to prevent over-fitting.

This tool will write some residual files of the form

```
initial_residuals_no_loss_function_reference_terrain.txt
final_residuals_no_loss_function_reference_terrain.txt
```

which may be studied to see if the error-to-lidar decreased. Each residual is defined as the distance, in pixels, between a terrain point projected into the left camera image and then transferred onto the right image via the unaligned disparity and its direct projection into the right camera.

If the initial errors in that file are large to start with, say more than 2-3 pixels, there is a chance something is wrong. Either the cameras are not well-aligned to each other or to the ground, or the intrinsics are off too much. In that case it is possible the errors are too large for this approach to reduce them effectively.

We strongly recommend that for this process one should not rely on bundle adjustment to create interest points, but to use the dense and uniformly distributed ones created with stereo, as suggested earlier.

The hope is that after these directions are followed, this will result in a smaller intersection error and a smaller error to the lidar/DEM ground truth (the later can be evaluated by invoking `geodiff --absolute` on the ASP-created aligned DEM and the reference lidar/DEM file).

When the lidar file is large, in bundle adjustment one can use the flag `--lon-lat-limit` to read only a relevant portion of it. This can speed up setting up the problem but does not affect the optimization.

Using the heights from a reference DEM

In some situations the DEM obtained with ASP is, after alignment, quite similar to the reference DEM, but the heights may be off. This can happen, for example, if the focal length is not accurately known. It is then possible after triangulating the interest point matches in bundle adjustment to replace their heights above datum with values obtained from the reference DEM, which are presumably more accurate. These triangulated points can be kept fixed while the extrinsics and intrinsics of the cameras are varied. The option for this is `--heights-from-dem arg`. To allow these triangulated points to vary somewhat, one can pass a positive value to `--heights-from-dem-weight`. The larger its value is, the more constrained those points will be.

This option can be used instead of the `--reference-terrain` option or together with it, and the DEM provided need not be the same for the two options.

It is important to note that here we assume that a simple height correction is enough. Hence this option is an approximation, and perhaps it should be used iteratively, and a subsequent pass of bundle adjustment should be done without it, or one should consider using a smaller weight above. This option can however be more effective than using `--reference-terrain` when there is a large uncertainty in camera intrinsics.

Using multiple images

Everything mentioned earlier works with more than two images, in fact, having more images is highly desirable, and ideally the images overlap a lot. For example, one can create stereo pairs consisting of first and second images, second and third, third and fourth, etc., invoke the above logic for each pair, that is, run stereo, alignment to the ground truth, dense interest point generation, creation of unaligned disparities, and transforming the cameras using the alignment transform matrix. Then, a directory can be made in which one can copy the dense interest point files, and run bundle adjustment with intrinsics optimization jointly for all cameras. Hence, one should use a command as follows (the example here is for 4 images):

```
disp1=run_stereo12/run-unaligned-D.tif
disp2=run_stereo23/run-unaligned-D.tif
disp3=run_stereo34/run-unaligned-D.tif
bundle_adjust -t nadirpinhole --inline-adjustments         \
  --solve-intrinsics  --camera-weight 0                    \
  img1.tif img2.tif img3.tif img4.tif                      \
  run_align_12/run-img1.tsai run_align12/run-img2.tsai     \
  run_align_34/run-img3.tsai run_align34/run-img4.tsai     \
  --reference-terrain lidar.csv                            \
  --disparity-list "$disp1 $disp2 $disp3"                  \
  --max-disp-error 50 --max-num-reference-points 1000000   \
  --overlap-limit 1 --parameter-tolerance 1e-12            \
  --reference-terrain-weight 5                             \
  -o run_ba_intr_lidar/run
```

In case it is desired to omit the disparity between one pair of images, for example, if they don't overlap, instead of the needed unaligned disparity one can put the word `none` in this list.

Notice that since this joint adjustment was initialized from several stereo pairs, the second camera picked above, for example, could have been either the second camera from the first pair, or the first camera from the second pair, so there was a choice to make. In section 11.16 an example is shown where a preliminary bundle adjustment happens at the beginning, without using a reference terrain, then those cameras are jointly aligned to the reference terrain, and then one continues as done above, but this time one need not have dealt with individual stereo pairs.

The option `--overlap-limit` can be used to control which images should be tested for interest point matches, and a good value for it is say 1 if one plans to use the interest points generated by stereo, though a value of 2 may not hurt either. One may want to decrease `--parameter-tolerance`, for example, to 1e-12, and set a value for `--max-disp-error`, e.g, 50, to exclude unreasonable disparities (this last number may be something one should experiment with, and the results can be somewhat sensitive to it). A larger value of `--reference-terrain-weight` can improve the alignment of the cameras to the reference terrain.

Also note the earlier comment about sharing and floating the intrinsics individually.

RPC lens distortion

If it is realized that the optimized intrinsics still do not make the ASP-generated DEMs agree very well with the ground truth, and some residual and systematic error can be seen either by comparing these two or in intersection error files, it may be convenient to convert the current camera models to ones with the distortion given by rational function coefficients (RPC) of a desired degre (section D.1). An RPC model can have a lot more coefficients to optimize, hence a better fit can be found. However, it is suggested to use low-degree polynomials as those are easy to fit, and go to higher degree only for refinement if needed.

An example showing how to convert a camera model to RPC is given in section A.40.

Working with map-projected images

If stereo was done with map-projected images, one can still extract dense interest point matches and the unaligned disparity from such a run, and these can be applied with the original unprojected images for the purpose of bundle adjustment (after being renamed appropriately). This may be convenient since while bundle adjustment must always happen with the original images, stereo could be faster and more accurate

when images are map-projected. It is suggested that the unaligned disparity and interest points obtained this way be examined carefully. Particularly the grid size used in mapprojection should be similar to the ground sample distance for the raw images for best results.

8.3 Bundle adjustment using ISIS

In what follows we describe how to do bundle adjustment using ISIS's tool-chain. It also serves to describe bundle adjustment in more detail, which is applicable to other bundle adjustment tools as well, including Stereo Pipeline's own tool.

In bundle adjustment, the position and orientation of each camera station are determined jointly with the 3D position of a set of image tie-points points chosen in the overlapping regions between images. Tie points, as suggested by the name, tie multiple camera images together. Their physical manifestation would be a rock or small crater than can be observed across more than one image.

Tie-points are automatically extracted using ISIS's `autoseed` and `pointreg` (alternatively one could use a number of outside methods such as the famous SURF[12]). Creating a collection of tie points, called a *control network*, is a three step process. First, a general geographic layout of the points must be decided upon. This is traditionally just a grid layout that has some spacing that allows for about 20-30 measurements to be made per image. This shows up in slightly different projected locations in each image due to their slight misalignments. The second step is to have an automatic registration algorithm try to find the same feature in all images using the prior grid as a starting location. The third step is to manually verify all measurements visually, checking to insure that each measurement is looking at the same feature.

Bundle Adjustment in ISIS is performed with the `jigsaw` executable. It generally follows the method described in [148] and determines the best camera parameters that minimize the projection error given by $\epsilon = \sum_k \sum_j (I_k - I(C_j, X_k))^2$ where I_k are the tie points on the image plane, C_j are the camera parameters, and X_k are the 3D positions associated with features I_k. $I(C_j, X_k)$ is an image formation model (i.e. forward projection) for a given camera and 3D point. To recap, it projects the 3D point, X_k, into the camera with parameters C_j. This produces a predicted image location for the 3D point that is compared against the observed location, I_k. It then reduces this error with the Levenberg-Marquardt algorithm (LMA). Speed is improved by using sparse methods as described in Hartley and Zisserman [54], Konolige [71], and Chen et al. [23].

Even though the arithmetic for bundle adjustment sounds clever, there are faults with the base implementation. Imagine a case where all cameras and 3D points were collapsed into a single point. If you evaluate the above cost function, you'll find that the error is indeed zero. This is not the correct solution if the images were taken from orbit. Another example is if a translation was applied equally to all 3D points and camera locations. This again would not affect the cost function. This fault comes from bundle adjustment's inability to control the scale and translation of the solution. It will correct the geometric shape of the problem, yet it cannot guarantee that the solution will have correct scale and translation.

ISIS attempts to fix this problem by adding two additional cost functions to bundle adjustment. First of which is $\epsilon = \sum_j (C_j^{initial} - C_j)^2$. This constrains camera parameters to stay relatively close to their initial values. Second, a small handful of 3D ground control points can be chosen by hand and added to the error metric as $\epsilon = \sum_k (X_k^{gcp} - X_k)^2$ to constrain these points to known locations in the planetary coordinate frame. A physical example of a ground control point could be the location of a lander that has a well known location. GCPs could also be hand-picked points against a highly regarded and prior existing map such as the THEMIS Global Mosaic or the LRO-WAC Global Mosaic.

Like other iterative optimization methods, there are several conditions that will cause bundle adjustment to terminate. When updates to parameters become insignificantly small or when the error, ϵ, becomes insignificantly small, then the algorithm has converged and the result is most likely as good as it will get.

However, the algorithm will also terminate when the number of iterations becomes too large in which case bundle adjustment may or may not have finished refining the parameters of the cameras.

8.3.1 Tutorial: Processing Mars Orbital Camera Imagery

This tutorial for ISIS's bundle adjustment tools is taken from [100] and [101]. These tools are not a product of NASA nor the authors of Stereo Pipeline. They were created by USGS and their documentation is available at [22].

What follows is an example of bundle adjustment using two MOC images of Hrad Vallis. We use images E02/01461 and M01/00115, the same as used in Chapter 3. These images are available from NASA's PDS (the ISIS `mocproc` program will operate on either the IMQ or IMG format files, we use the `.imq` below in the example). For reference, the following ISIS commands are how to convert the MOC images to ISIS cubes.

```
ISIS 3> mocproc from=e0201461.imq to=e0201461.cub mapping=no
ISIS 3> mocproc from=m0100115.imq to=m0100115.cub mapping=no
```

Note that the resulting images are not map-projected. Bundle adjustment requires the ability to project arbitrary 3D points into the camera frame. The process of map-projecting an image dissociates the camera model from the image. Map-projecting can be perceived as the generation of a new infinitely large camera sensor that may be parallel to the surface, a conic shape, or something more complex. That makes it

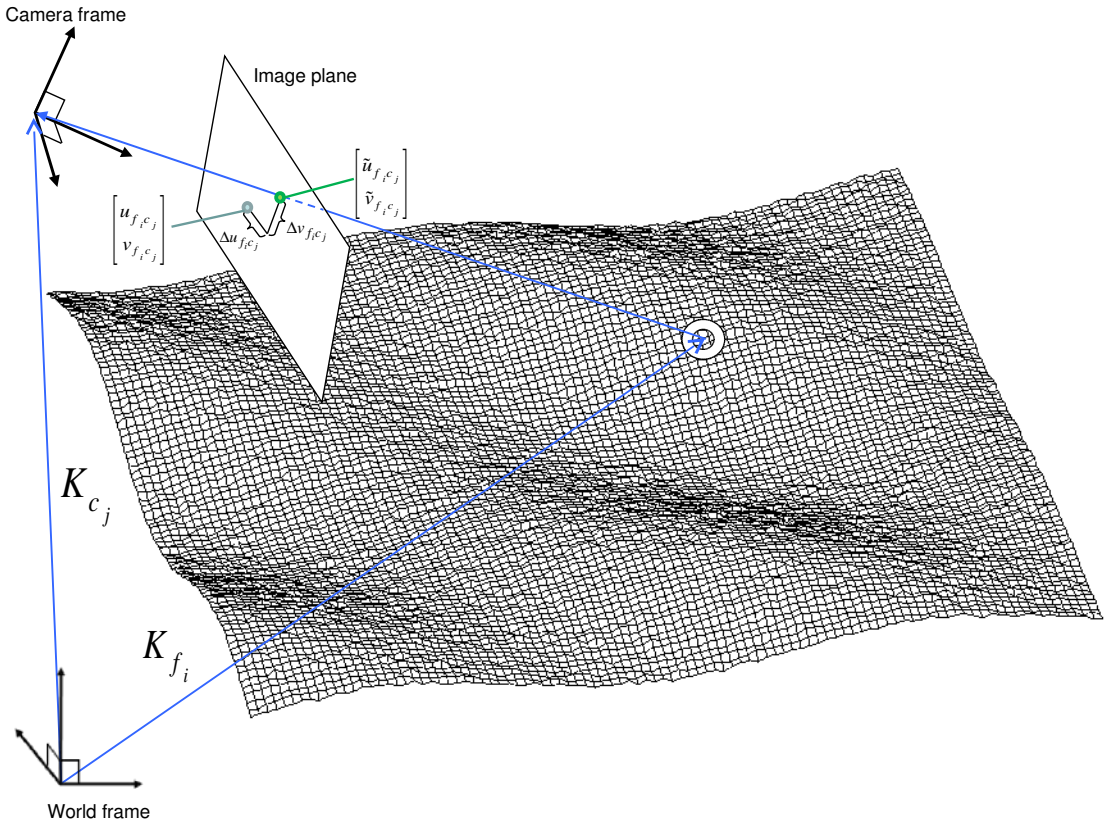


Figure 8.3: A feature observation in bundle adjustment, from Moore et al. [97]

extremely hard to project a random point into the camera's original model. The math would follow the transformation from projection into the camera frame, then projected back down to surface that ISIS uses, then finally up into the infinitely large sensor. `Jigsaw` does not support this and thus does not operate on map-projected imagery.

Before we can dive into creating our tie-point measurements we must finish prepping these images. The following commands will add a vector layer to the cube file that describes its outline on the globe. It will also create a data file that describes the overlapping sections between files.

```
ISIS 3> footprintinit from=e0201461.cub
ISIS 3> footprintinit from=m0100115.cub
ISIS 3> echo *cub |  xargs -n1 echo > cube.lis
ISIS 3> findimageoverlaps from=cube.lis overlaplist=overlap.lis
```

At this point, we are ready to start generating our measurements. This is a three step process that requires defining a geographic pattern for the layout of the points on the groups, an automatic registration pass, and finally a manual clean up of all measurements. Creating the ground pattern of measurements is performed with `autoseed`. It requires a settings file that defines the spacing in meters between measurements. For this example, write the following text into a *autoseed.def* file.

```
Group = PolygonSeederAlgorithm
      Name = Grid
      MinimumThickness = 0.01
      MinimumArea = 1
      XSpacing = 1000
      YSpacing = 2000
End_Group
```

The minimum thickness defines the minimum ratio between the sides of the region that can have points applied to it. A choice of 1 would define a square and anything less defines thinner and thinner rectangles. The minimum area argument defines the minimum square meters that must be in an overlap region. The last two are the spacing in meters between control points. Those values were specifically chosen for this pair so that about 30 measurements would be produced from `autoseed`. Having more control points just makes for more work later on in this process. Run `autoseed` with the following instruction.

```
ISIS 3> autoseed fromlist=cube.lis overlaplist=overlap.lis    \
          onet=control.net deffile=autoseed.def networkid=moc \
          pointid=???? description=hrad_vallis
```

The next step is to perform auto registration of these features between the two images using `pointreg`. This program also requires a settings file that describes how to do the automatic search. Copy the text box below into a *autoRegTemplate.def* file.

```
Object = AutoRegistration
 Group = Algorithm
   Name         = MaximumCorrelation
   Tolerance    = 0.7
 EndGroup
```

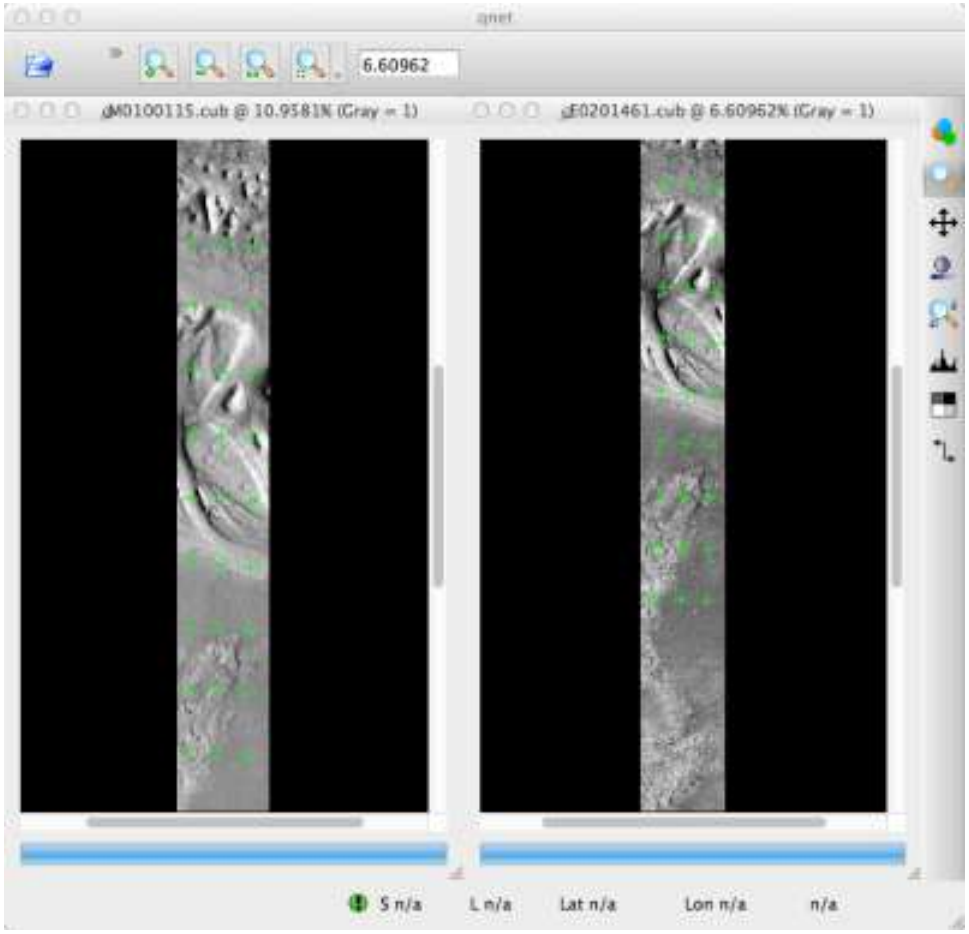


Figure 8.4: A visualization of the features laid out by `autoseed` in `qnet`. Note that the marks do not cover the same features between images. This is due to the poor initial spice data for MOC imagery.

```
    Group = PatternChip
      Samples = 21
      Lines   = 21
      MinimumZScore = 1.5
      ValidPercent = 80
    EndGroup

    Group = SearchChip
      Samples = 75
      Lines   = 1000
    EndGroup
  EndObject
```

The search chip defines the search range for which `pointreg` will look for matching imagery. The pattern chip is simply the kernel size of the matching template. The search range is specific for this image pair. The control network result after `autoseed` had a large vertical offset in the ball park of 500 px. The large misalignment dictated the need for the large search in the lines direction. Use `qnet` to get an idea for what the pixel shifts look like in your stereo pair to help you decide on a search range. In this example, only one measurement failed to match automatically. Here are the arguments to use in this example of `pointreg`.

```
ISIS 3> pointreg fromlist=cube.lis cnet=control.net              \
          onet=control_pointreg.net deffile=autoRegTemplate.def
```

The third step is to manually edit the control and verify the measurements in `qnet`. Type `qnet` in the terminal and then open *cube.lis* and lastly *control_pointreg.net*. From the Control Network Navigator window, click on the first point listed as *0001*. That opens a third window called the Qnet Tool. That window will allow you to play a flip animation that shows alignment of the feature between the two images. Correcting a measurement is performed by left clicking in the right image, then clicking *Save Measure*, and finally finishing by clicking *Save Point*.

In this tutorial, measurement *0025* ended up being incorrect. Your number may vary if you used different settings than the above or if MOC spice data has improved since this writing. When finished, go back to the main Qnet window. Save the final control network as *control_qnet.net* by clicking on *File*, and then *Save As*.

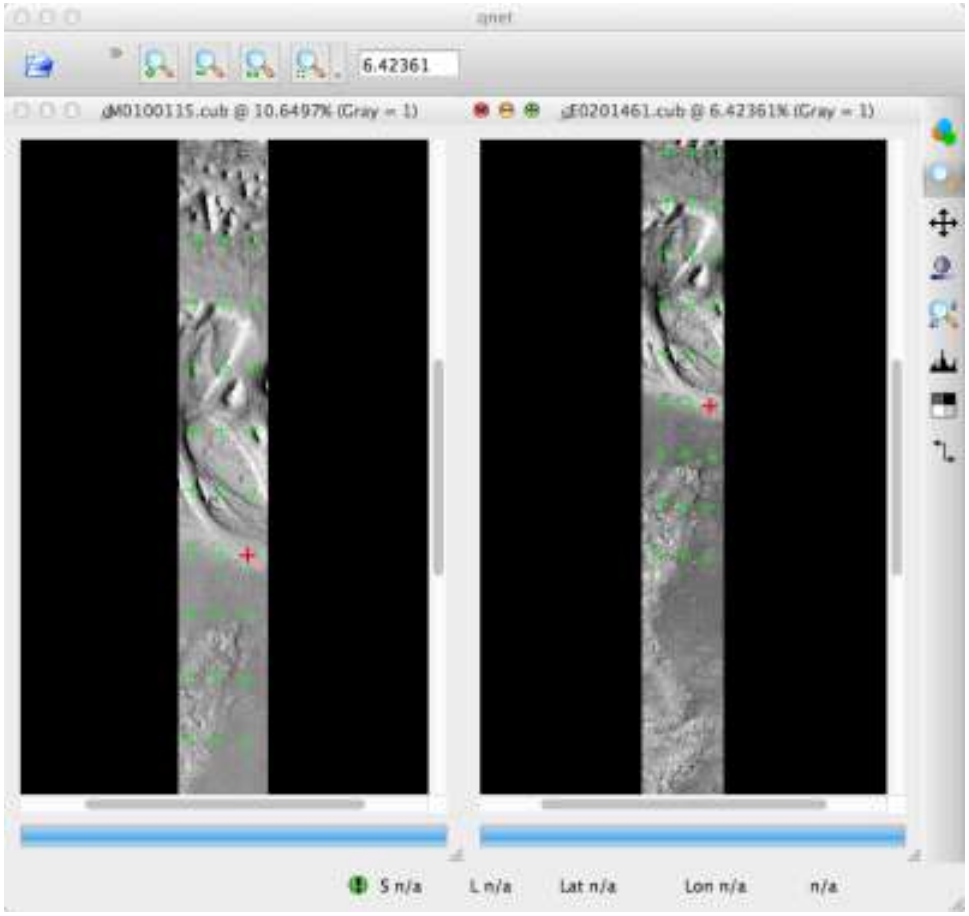


Figure 8.5: A visualization of the features after manual editing in `qnet`. Note that the marks now appear in the same location between images.

Once the control network is finished, it is finally time to start bundle adjustment. Here's what the call to `jigsaw` looks like:

```
ISIS 3> jigsaw fromlist=cube.lis update=yes twist=no radius=yes \
          cnet=control_qnet.net onet=control_ba.net
```

The update option defines that we would like to update the camera pointing, if our bundle adjustment converges. The *twist=no* says to not solve for the camera rotation about the camera bore. That property

is usually very well known as it is critical for integrating an image with a line-scan camera. The *radius=yes* means that the radius of the 3D features can be solved for. Using no will force the points to use height values from another source, usually LOLA or MOLA.

The above command will spew out a bunch of diagnostic information from every iteration of the optimization algorithm. The most important feature to look at is the *sigma0* value. It represents the mean of pixel errors in the control network. In our run, the initial error was 1065 px and the final solution had an error of 1.1 px.

Producing a DEM using the newly created camera corrections is the same as covered in the Tutorial on page 15. When using `jigsaw`, it modifies a copy of the spice data that is stored internally to the cube file. Thus when we want to create a DEM using the correct camera geometry, no extra information needs to be given to `stereo` since it is already contained in the file. In the event a mistake has been made, `spiceinit` will overwrite the spice data inside a cube file and provide the original uncorrected camera pointing.

```
ISIS 3> stereo E0201461.cub M0100115.cub bundled/bundled
```

Chapter 9

Solving for Camera Poses Based on Images

The ASP tool `camera_solve` offers several ways to find the true position of frame camera images that do not come with any attached pose metadata. This can be useful with aerial, hand-held, and historical imagery for which such information may be incomplete or inaccurate.

An overview of the tool and examples are provided in this chapter. Reference information for this tool can be found in Appendix A.39.

This tool can be optionally bypassed if, for example, the longitude and latitude of the corners of all images are known (section 9.4).

9.1 Camera Solve Overview

The `camera_solve` tool is implemented as a Python wrapper around two other tools. The first of these is the the THEIA software library, which is used to generate initial camera position estimates in a local coordinate space. You can learn more about THEIA at http://www.theia-sfm.org/index.html. The second tool is ASP's own `bundle_adjust` tool. The second step improves the solution to account for lens distortion and transforms the solution from local to global coordinates by making use of additional input data.

The tool only solves for the extrinsic camera parameters and the user must provide intrinsic camera information. You can use the `camera_calibrate` tool (see Appendix A.38) or other camera calibration software to solve for intrinsic parameters if you have access to the camera in question. The camera calibration information must be contained in a .tsai pinhole camera model file and must passed in using the `--calib-file` option. You can find descriptions of our supported pinhole camera models in section D.1.

If no intrinsic camera information is known, it can be guessed by doing some experimentation. This is discussed in section 9.5.

In order to transform the camera models from local to world coordinates, one of three pieces of information may be used. These sources are listed below and described in more detail in the examples that follow:

- A set of ground control points of the same type used by `pc_align`. The easiest way to generate these points is to use the ground control point writer tool available in the `stereo-gui` tool.
- A set of estimated camera positions (perhaps from a GPS unit) stored in a csv file.
- A DEM which a local point cloud can be registered to using `pc_align`. This method can be more accurate if estimated camera positions are also used. The user must perform alignment to a DEM, that step is not handled by `camera_solve`.

Power users can tweak the individual steps that `camera_solve` goes through to optimize their results. This primarily involves setting up a custom flag file for THEIA and/or passing in settings to `bundle_adjust`.

9.2 Example: Apollo 15 Metric Camera

To demonstrate the ability of the Ames Stereo Pipeline to process a generic frame camera we use images from the Apollo 15 Metric camera. The calibration information for this camera is available online and we have accurate digital terrain models we can use to verify our results.

First download a pair of images:

```
> wget http://apollo.sese.asu.edu/data/metric/AS15/png/AS15-M-0414_MED.png
> wget http://apollo.sese.asu.edu/data/metric/AS15/png/AS15-M-1134_MED.png
```

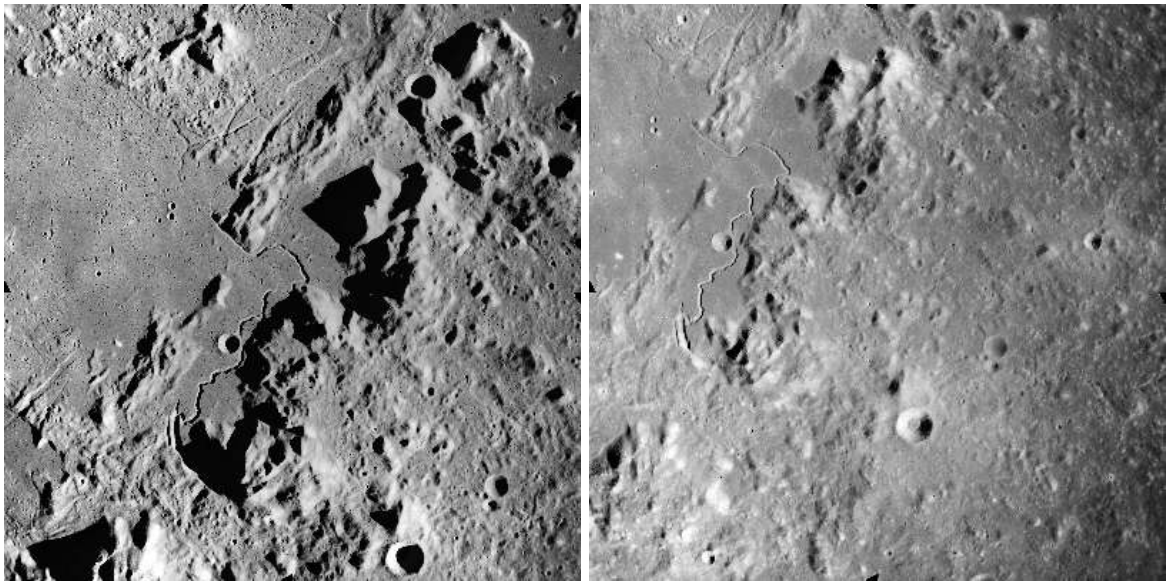


Figure 9.1: The two Apollo 15 images.

In order to make the example run faster we use downsampled versions of the original images. The images at those links have already been downsampled by a factor of $4\sqrt{2}$ from the original images. This means that the effective pixel size has increased from five microns (0.005 millimeters) to 0.028284 millimeters.

The next step is to fill out the rest of the pinhole camera model information we need. Using the data sheets available at http://apollo.sese.asu.edu/SUPPORT_DATA/AS15_SIMBAY_SUMMARY.pdf we can find the lens distortion parameters for metric camera. Looking at the ASP lens distortion models in section D.1, we see that the description matches ASP's Brown-Conrady model. Using the example in the appendix we can fill out the rest of the sensor model file (metric_model.tsai) so it looks as follows:

```
VERSION_3
fu = 76.080
fv = 76.080
cu = 57.246816
cv = 57.246816
u_direction = 1  0  0
```

```
v_direction = 0  1  0
w_direction = 0  0  1
C = 0 0 0
R = 1 0 0 0 1 0 0 0 1
pitch = 0.028284
BrownConrady
xp = -0.006
yp = -0.002
k1 = -0.13361854e-5
k2 = 0.52261757e-09
k3 = -0.50728336e-13
p1 = -0.54958195e-06
p2 = -0.46089420e-10
phi = 2.9659070
```

These parameters use units of millimeters so we have to convert the nominal center point of the images from 2024 pixels to units of millimeters. Note that for some older images like these the nominal image center can be checked by looking for some sort of marking around the image borders that indicates where the center should lie. For these pictures there are black triangles at the center positions and they line up nicely with the center of the image. Before we try to solve for the camera positions we can run a simple tool to check the quality of our camera model file:

```
> undistort_image AS15-M-0414_MED.png metric_model.tsai -o corrected_414.tif
```

It is difficult to tell if the distortion model is correct by using this tool but it should be obvious if there are any gross errors in your camera model file such as incorrect units or missing parameters. In this case the tool will fail to run or will produce a significantly distorted image. For certain distortion models the `undistort_image` tool may take a long time to run.

If your input images are not all from the same camera or were scanned such that the center point is not at the same pixel, you can run `camera_solve` with one camera model file per input image. To do so pass a space-separated list of files surrounded by quotes to the `-calib-file` option such as `-calib-file "c1.tsai c2.tsai c3.tsai"`.

If we do not see any obvious problems we can go ahead and run the `camera_solve` tool:

```
> camera_solve out/ AS15-M-0414_MED.png AS15-M-1134_MED.png --datum D_MOON \
  --calib-file metric_model.tsai
```

We should get some camera models in the output folder and see a printout of the final bundle adjustment error among the program output information:

```
Cost:
Initial                          1.450385e+01
Final                            7.461198e+00
Change                           7.042649e+00
```

We can't generate a DEM with these local camera models but we can run stereo anyways and look at the intersection error in the fourth band of the `PC.tif` file. While there are many speckles in this example where stereo correlation failed the mean intersection error is low and we don't see any evidence of lens distortion error.

```
> stereo AS15-M-0414_MED.png AS15-M-1134_MED.png out/AS15-M-0414_MED.png.final.tsai \
  out/AS15-M-1134_MED.png.final.tsai -t pinhole s_local/out  --corr-timeout 300 \
  --erode-max-size 100
> gdalinfo -stats s_local/out-PC.tif
...
Band 4 Block=256x256 Type=Float32, ColorInterp=Undefined
  Minimum=0.000, Maximum=56.845, Mean=0.340, StdDev=3.512
  Metadata:
    STATISTICS_MAXIMUM=56.844654083252
    STATISTICS_MEAN=0.33962282293374
    STATISTICS_MINIMUM=0
    STATISTICS_STDDEV=3.5124044818554
```

The tool `point2mesh` (section A.7) can be used to obtain a visualizable mesh from the point cloud.

In order to generate a useful DEM, we need to move our cameras from local coordinates to global coordinates. The easiest way to do this is to obtain known ground control points (GCPs) which can be identified in the frame images. This will allow an accurate positioning of the cameras provided that the GCPs and the camera model parameters are accurate. To create GCPs see the instructions for the `stereo_gui` tool in appendix A.4.1. For the Moon there are several ways to get DEMs and in this case we generated GCPs using `stereo_gui` and a DEM generated from LRONAC images.

After running this command:

```
> camera_solve out_gcp/ AS15-M-0414_MED.png AS15-M-1134_MED.png --datum D_MOON \
  --calib-file metric_model.tsai --gcp-file ground_control_points.gcp
```

we end up with results that can be compared with the a DEM created from LRONAC images. The stereo results on the Apollo 15 images leave something to be desired but the DEM they produced has been moved to the correct location. You can easily visualize the output camera positions using the `orbitviz` tool with the `-load-camera-solve` option as shown below. Green lines between camera positions mean that a sufficient number of matching interest points were found between those two images.

For GCP to be usable, they can be one of two kinds. The preferred option is for each of at least three GCP to show up in more than one image. Then their triangulated positions can be determined in local coordinates and in global (world) coordinates, and `bundle_adjust` will be able to compute the transform between these coordinate systems, and convert the cameras to world coordinates.

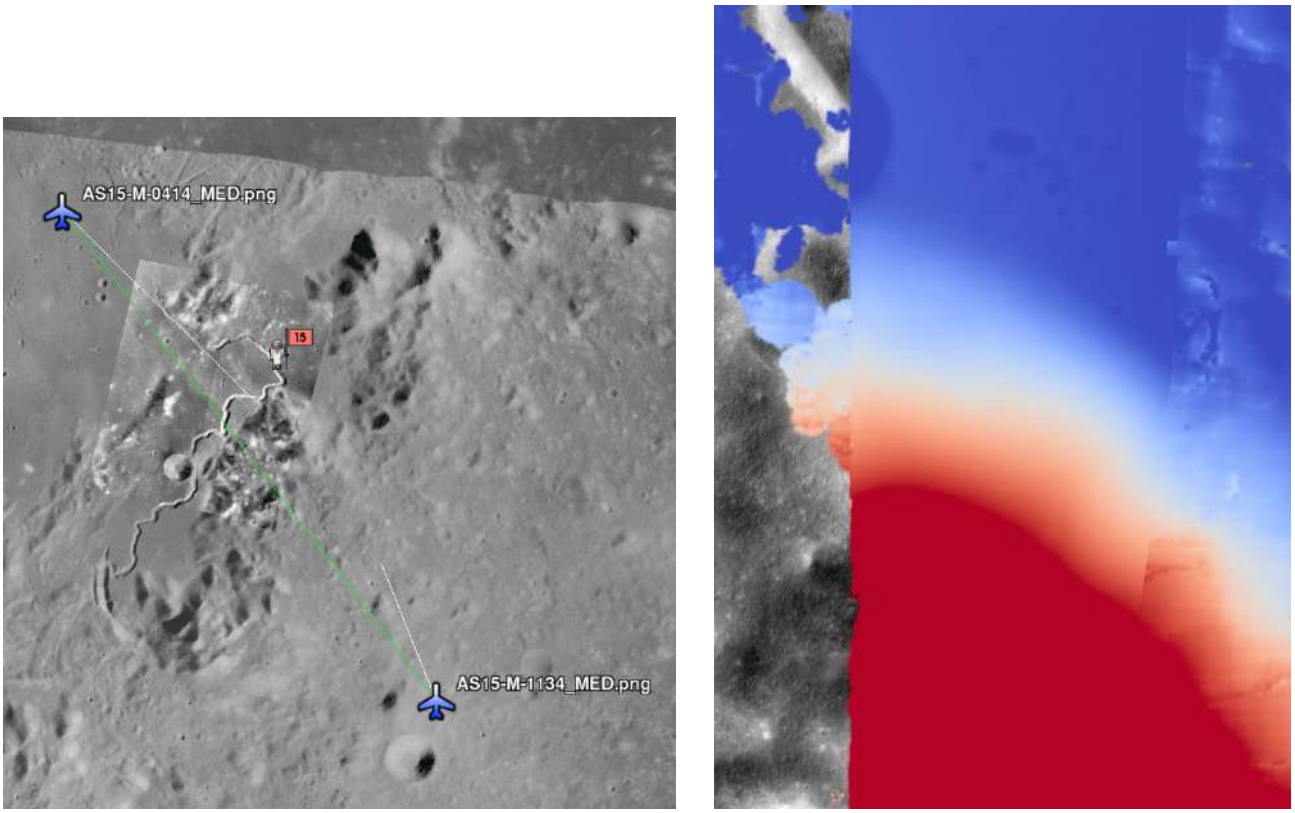
If this is not possible, then at least two of the images should have at least three GCP each, and they need not be shared among the images. For example, for each image the longitude, latitude, and height of each of its four corners can be known. Then, one can pass such a GCP file to `camera_solve` and also with the flag:

```
  --bundle-adjust-params "--transform-cameras-using-gcp"
```

and it will attempt to transform the cameras to world coordinates.

Next, one can run stereo.

```
> stereo AS15-M-0414_MED.png AS15-M-1134_MED.png out_gcp/AS15-M-0414_MED.png.final.tsai \
  out_gcp/AS15-M-1134_MED.png.final.tsai -t nadirpinhole s_global/out  --corr-timeout 300 \
  --erode-max-size 100
> orbitviz -t nadirpinhole -r moon out_gcp --load-camera-solve
```



(a) `orbitviz` display (b) `KML` Screenshot

Figure 9.2: (a) Solved for camera positions plotted using orbitviz. (b) A narrow LRONAC DEM overlaid on the resulting DEM, both colormapped to the same elevation range.

ASP also supports the method of initializing the `camera_solve` tool with estimated camera positions. This method will not move the cameras to exactly the right location but it should get them fairly close and at the correct scale, hopefully close enough to be used as-is or to be refined using `pc_align` or some other method. To use this method, pass additional bundle adjust parameters to `camera_solve` similar to the following line:

```
--bundle-adjust-params '--camera-positions nav.csv \
 --csv-format "1:file 12:lat 13:lon 14:height_above_datum" --camera-weight 0.2'
```

The nav data file you use must have a column (the "file" column) containing a string that can be matched to the input image files passed to `camera_solve`. The tool looks for strings that are fully contained inside one of the image file names, so for example the field value `2009_10_20_0778` would be matched with the input file `2009_10_20_0778.JPG`.

Chapter 5 will discuss the `stereo` program in more detail and the other tools in ASP.

9.3 Example: IceBridge DMS Camera

The DMS (Digital Mapping System) Camera is a frame camera flown on as part of the NASA IceBridge program to collect digital terrain imagery of polar and Antarctic terrain (http://nsidc.org/icebridge/portal/).

To process this data the steps are very similar to the steps described above for the Apollo Metric camera but there are some aspects which are particular to IceBridge. You can download DMS images from ftp://n5eil01u.ecs.nsidc.org/SAN2/ICEBRIDGE_FTP/IODMS0_DMSraw_v01/. A list of the available data types can be found at https://nsidc.org/data/icebridge/instr_data_summary.html. This example uses data from the November 5, 2009 flight over Antarctica. The following camera model (icebridge_model.tsai) was used (see section D.1 on Pinhole camera models):

```
VERSION_3
fu = 28.429
fv = 28.429
cu = 17.9712
cv = 11.9808
u_direction = 1  0  0
v_direction = 0  1  0
w_direction = 0  0  1
C = 0 0 0
R = 1 0 0 0 1 0 0 0 1
pitch = 0.0064
Photometrix
xp = 0.004
yp = -0.191
k1 = 1.31024e-04
k2 = -2.05354e-07
k3 = -5.28558e-011
p1 = 7.2359e-006
p2 = 2.2656e-006
b1 = 0.0
b2 = 0.0
```

Note that these images are RGB format which is not supported by all ASP tools. To use the files with ASP, first convert them to single channel images using a tool such as ImageMagick's `convert`, `gdal_translate`, or `gdal_edit.py`. Different conversion methods may produce slightly different results depending on the contents of your input images. Some conversion command examples are shown below:

```
convert rgb.jpg -colorspace Gray gray.jpg
gdal_calc.py  --overwrite --type=Float32 --NoDataValue=-32768       \
  -A rgb.tif --A_band=1 -B rgb.tif --B_band=2 -C rgb.tif             \
  --C_band=3 --outfile=gray.tif --calc="A*0.2989+B*0.5870+C*0.1140"
gdal_translate -b 1 rgb.jpg gray.jpg
```

In the third command we used `gdal_translate` to pick a single band rather than combining the three.

Obtaining ground control points for icy locations on Earth can be particularly difficult because they are not well surveyed or because the terrain shifts over time. This may force you to use estimated camera positions to convert the local camera models into global coordinates. To make this easier for IceBridge data sets, ASP provides the `icebridge_kmz_to_csv` tool (see appendix A.44) which extracts a list of estimated camera positions from the kmz files available for each IceBridge flight at http://asapdata.arc.nasa.gov/dms/missions.html.

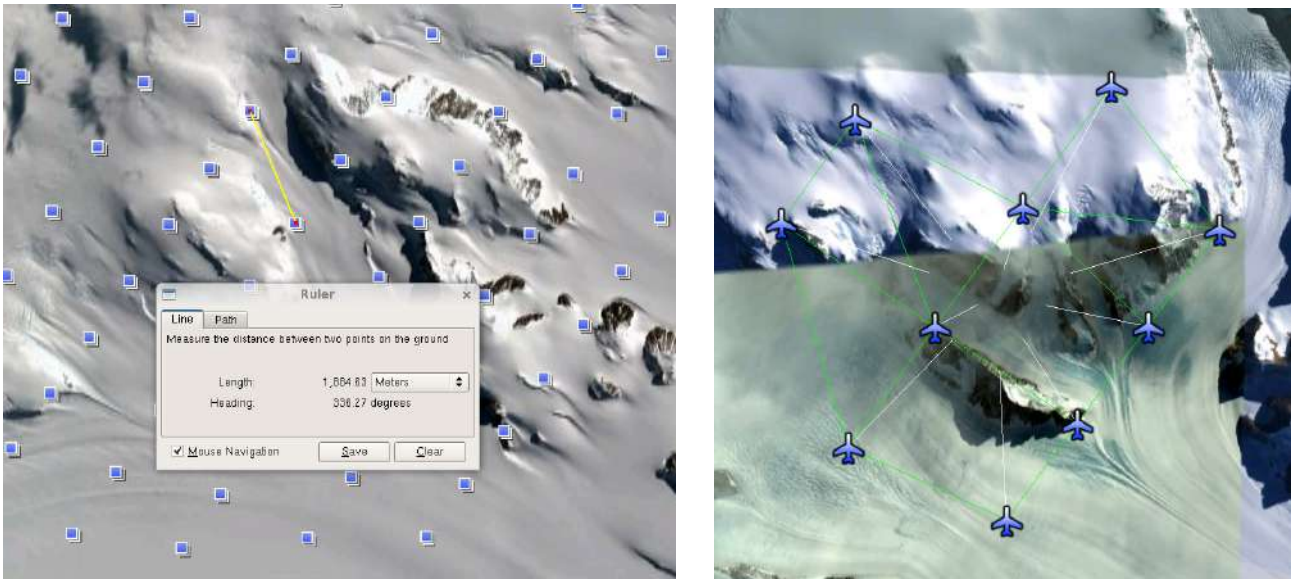
Another option which is useful when processing IceBridge data is the `--position-filter-dist` option for `bundle_adjust`. IceBridge data sets contain a large number of images and when processing many at once

you can significantly decrease your processing time by using this option to limit interest-point matching to image pairs which are actually close enough to overlap. A good way to determine what distance to use is to load the camera position kmz file from their website into Google Earth and use the ruler tool to measure the distance between a pair of frames that are as far apart as you want to match. Commands using these options may look like this:

```
icebridge_kmz_to_csv  1000123_DMS_Frame_Events.kmz  camera_positions.csv
camera_solve out 2009_11_05_00667.JPG 2009_11_05_00668.JPG  \
  2009_11_05_00669.JPG 2009_11_05_00670.JPG  2009_11_05_02947.JPG 2009_11_05_02948.JPG \
  2009_11_05_02949.JPG  2009_11_05_02950.JPG  2009_11_05_01381.JPG 2009_11_05_01382.JPG  \
  --datum WGS84 --calib-file icebridge_model.tsai  \
  --bundle-adjust-params '--camera-positions camera_positions.csv \
  --csv-format "1:file 2:lon 3:lat 4:height_above_datum" --position-filter-dist 2000'
orbitviz out --load-camera-solve --hide-labels -r wgs84 -t nadirpinhole
```

Alternatively, the `camera_solve` executable can be bypassed altogether. If a given image has already an orthoimage associated with it (check the IceBridge portal page), that provides enough information to guess an initial position of the camera, using the `ortho2pinhole` tool. Later, the obtained cameras can be bundle-adjusted. Here is how this tool can be used, on grayscale images:

```
ortho2pinhole raw_image.tif ortho_image.tif icebridge_model.tsai output_pinhole.tsai
```



(a) (b)

Figure 9.3: (a) Measuring the distance between estimated frame locations using Google Earth and an IceBridge kmz file. The kmz file is from the IceBridge website with no modifications. Using a position filter distance of 2000 meters will mostly limit image IP matching in this case to each image's immediate "neighbors". (b) Display of `camera_solve` results for ten IceBridge images using `orbitviz`.

Some IceBridge flights contain data from the Land, Vegetation, and Ice Sensor (LVIS) lidar which can be used to register DEMs created using DMS imagery. LVIS data can be downloaded at `ftp://n5eil01u.ecs.nsidc.org/SAN2/ICEBRIDGE/ILVIS2.001/`. The lidar data comes in plain text files that `pc_align` and `point2dem` can parse using the following option:

```
  --csv-format "5:lat 4:lon 6:height_above_datum"
```

ASP provides the `lvis2kml` tool to help visualize the coverage and terrain contained in LVIS files, see Appendix A.45 for details. The LVIS lidar coverage is sparse compared to the image coverage and you will have difficulty getting a good registration unless the region has terrain features such as hills or you are registering very large point clouds that overlap with the lidar coverage across a wide area. Otherwise `pc_align` will simply slide the flat terrain to an incorrect location to produce a low-error fit with the narrow lidar tracks. This test case was specifically chosen to provide strong terrain features to make alignment more accurate but `pc_align` still failed to produce a good fit until the lidar point cloud was converted into a smoothed DEM.

```
stereo 2009_11_05_02948.JPG  2009_11_05_02949.JPG  out/2009_11_05_02948.JPG.final.tsai \
  out/2009_11_05_02949.JPG.final.tsai st_run/out -t nadirpinhole
point2dem ILVIS2_AQ2009_1105_R1408_055812.TXT --datum WGS_1984 \
  --t_srs "+proj=stere +lat_0=-90 +lon_0=0 +k=1 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs" \
  --csv-format "5:lat 4:lon 6:height_above_datum"  --tr 30  \
  --search-radius-factor 2.0 -o lvis
pc_align  --max-displacement 1000 lvis-DEM.tif st_run/out-PC.tif  -o align_run/out \
  --save-transformed-source-points --datum wgs84 --outlier-ratio 0.55
point2dem align_run/out-trans_source.tif --datum WGS_1984 \
  --t_srs "+proj=stere +lat_0=-90 +lon_0=0 +k=1 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs"
colormap align_run_big/out-trans_source-DEM.tif --min 200 --max 1500
colormap lvis-DEM.tif --min 200 --max 1500
image2qtree lvis-DEM_CMAP.tif
image2qtree align_run_big/out-trans_source-DEM_CMAP.tif
```

Other IceBridge flights contain data from the Airborne Topographic Mapper (ATM) lidar sensor. Data from this sensor comes packed in one of several formats (variants of .qi or .h5) so ASP provides the `extract_icebridge_ATM_points` tool to convert them into plain text files, which later can be read into other ASP tools using the formatting:

```
  --csv-format "1:lat 2:lon 3:height_above_datum"
```

To run the tool, just pass in the name of the input file as an argument and a new file with a csv extension will be created in the same directory. Using the ATM sensor data is similar to using the LVIS sensor data.

For some IceBridge flights, lidar-aligned DEM files generated from the DMS image files are available, see the web page here: http://nsidc.org/data/iodms3 These files are improperly formatted and cannot be used by ASP as is. To correct them, run the `correct_icebridge_l3_dem` tool as follows:

```
correct_icebridge_l3_dem IODMS3_20120315_21152106_07371_DEM.tif  fixed_dem.tif 1
```

The third argument should be 1 if the DEM is in the northern hemisphere and 0 otherwise. The corrected DEM files can be used with ASP like any other DEM file.

Chapter 5 will discuss the `stereo` program in more detail and the other tools in ASP.

9.4 Solving for Pinhole cameras using GCP

If for a given image the intrinsics of the camera are known, and also the longitude and latitude (and optionally the heights above the datum) of its corners (or of some other pixels in the image), one can bypass

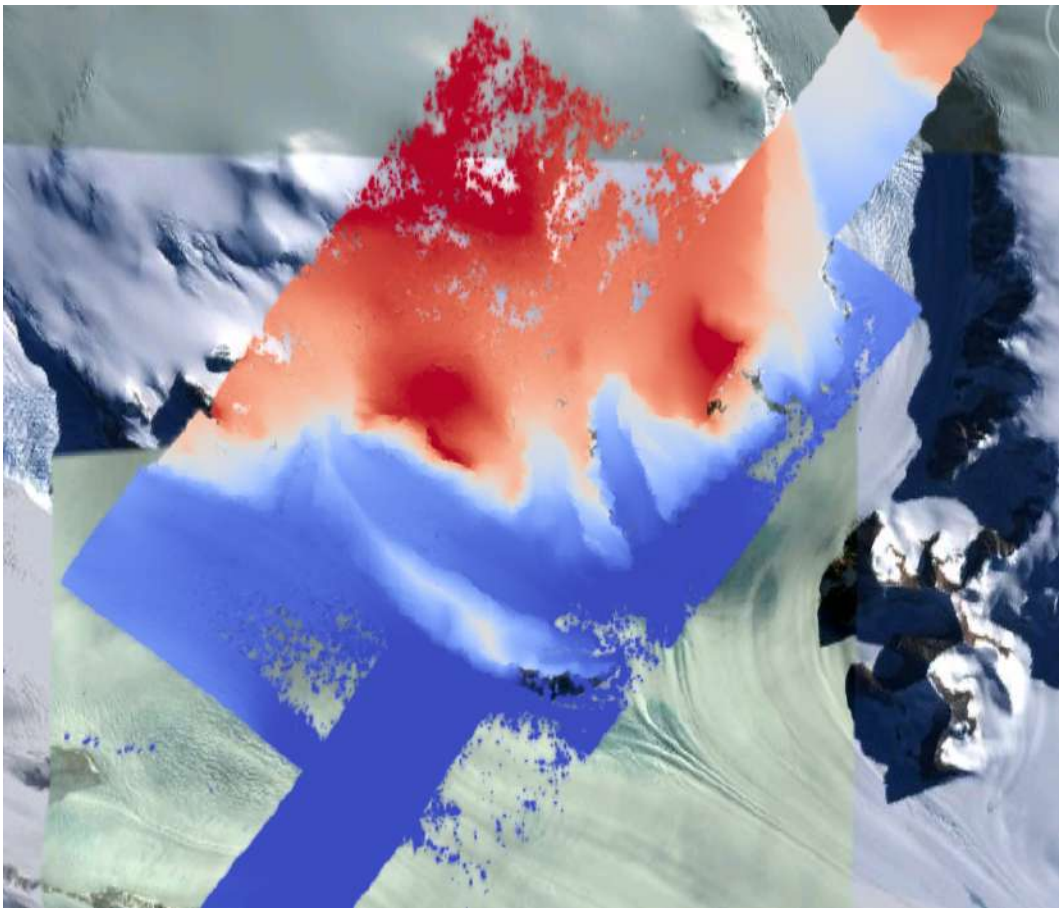


Figure 9.4: LVIS lidar DEM overlaid on the ASP created DEM, both colormapped to the same elevation range. The ASP DEM could be improved but the registration is accurate. Notice how narrow the LVIS lidar coverage is compared to the field of view of the camera. You may want to experiment using the SGM algorithm to improve the coverage.

the `camera_solve` tool and use `bundle_adjust` to get a rough initial camera position and orientation. This simple approach is often beneficial when, for example, one has historical images with rough geo-location information. Once a rough camera is created for each image, the cameras can then be bundle-adjusted jointly to refine them.

To achieve this, one creates a camera file, say called `init.tsai`, with only the intrinsics, and using trivial values for the camera center and rotation matrix:

```
VERSION_3
fu = 28.429
fv = 28.429
cu = 17.9712
cv = 11.9808
u_direction = 1  0  0
v_direction = 0  1  0
w_direction = 0  0  1
C = 0 0 0
R = 1 0 0 0 1 0 0 0 1
pitch = 0.0064
Photometrix
```

```
xp = 0.004
yp = -0.191
k1 = 1.31024e-04
k2 = -2.05354e-07
k3 = -5.28558e-011
p1 = 7.2359e-006
p2 = 2.2656e-006
b1 = 0.0
b2 = 0.0
```

Next, one creates a ground control points (GCP) file (section A.4.1), named, for example, `gcp.gcp`, containing the pixel positions and longitude and latitude of the corners or other known pixels (the heights above datum can be set to zero if not known). Here is a sample file, where the image is named `img.tif` (below the latitude is written before the longitude).

```
# id   lat     lon   height  sigmas  image   corners    sigmas
   1  37.62  -122.38   0     1 1 1  img.tif 0     0     1 1
   2  37.62  -122.35   0     1 1 1  img.tif 2560  0     1 1
   3  37.61  -122.35   0     1 1 1  img.tif 2560 1080   1 1
   4  37.61  -122.39   0     1 1 1  img.tif 0    1080   1 1
```

Such a file can be created with `stereo_gui` (section A.2.2).

One runs bundle adjustment with this data:

```
bundle_adjust -t nadirpinhole img.tif init.tsai gcp.gcp -o ba/run           \
   --datum WGS84 --inline-adjustments --camera-weight 0 --max-iterations 0  \
   --robust-threshold 10
```

which will write the desired correctly oriented camera file. Using a positive number of iterations will refine the camera.

It is important to look at the residual file

```
  run/run-final_residuals_no_loss_function_pointmap_point_log.csv
```

after this. The third column in this file is the optimized heights above the datum, while the fourth column has the reprojection errors from the corners on the ground into the camera.

If bundle adjustment is invoked with a positive number of iterations, and with a small value for the robust threshold, it tends to optimize only some of the corners and ignore the others, resulting in a large reprojection error, which is not desirable. If however, this threshold is too large, it may try to optimize the camera too aggressively, resulting in a poorly placed camera.

Sometimes it works to just get a rough camera estimate from this tool for each image individually, using zero iterations, as above, and then bundle adjust all images together with the obtained rough cameras and possibly also using the GCP files, this time with a positive number of iterations.

One can also use the bundle adjustment option `--fix-gcp-xyz` to not move the GCP during optimization, hence forcing the cameras to move more to conform to them.

ASP provides a tool named `cam_gen` which can also create a pinhole camera as above, and, in addition, is able to extract the heights of the corners from a DEM (section A.41).

9.5 Solving For Intrinsic Camera Parameters

If nothing is known about the intrinsic camera parameters, it may be possible to guess them with some experimentation. One can assume that the distortion is non-existent, and that the optical center is at the image center, which makes it possible to compute cu and cv. The pitch can be set to some small number, say 10^{-3} or 10^{-4}. The focal length can be initialized to equal cu or a multiple of it. Then `camera_solve` can be invoked, followed by `stereo`, `point2mesh`, and `point2dem --errorimage`. If, at least towards the center of the image, things are not exploding, we are on a good track.

Later, the camera parameters, especially the focal length, can be modified manually, and instead of using `camera_solve` again, just `bundle_adjust` can be called using the camera models found earlier, with the options to float some of the intrinsics, that is using `--solve-intrinsics` and `--intrinsics-to-float`.

If the overall results look good, but the intersection error after invoking `point2dem` around the image corners looks large, it is time to use some distortion model and float it, again using `bundle_adjust`. Sometimes if invoking this tool over many iterations the optical center and focal length may drift, and hence it may be helpful to have them fixed while solving for distortion.

If a pre-existing DEM is available, the tool `geodiff` can be used to compare it with what ASP is creating.

Such a pre-existing DEM can be used as a constraint when solving for intrinsics, as described in section 8.2.1.

Chapter 10

Shape-from-Shading

ASP provides a tool, named `sfs`, that can improve the level of detail of DEMs created by ASP or any other source using *shape-from-shading* (SfS). The tool takes as input one or more camera images, a DEM at roughly the same resolution as the images, and returns a refined DEM.

`sfs` works only with ISIS cub images. It has been tested thoroughly with Lunar LRO NAC datasets, and some experiments were done with Mars HiRISE images and with pictures from Charon, Pluto's moon. As seen later in the text, it returns reasonable results on the Moon as far as 85° South.

Currently, `sfs` is computationally expensive, and is practical only for DEMs whose width and height are several thousand pixels. It can be sensitive to errors in the position and orientation of the cameras, the accuracy of the initial DEM, and to the value of the two weights it uses. Yet, with some effort, it can work quite well.

A tool named `parallel_sfs` is provided (section A.36) that parallelizes `sfs` using multiple processes (optionally on multiple machines) by splitting the input DEM into tiles with padding, running `sfs` on each tile, and then blending the results.

The `sfs` program can model position-dependent albedo, different exposure values for each camera, shadows in the input images, and regions in the DEM occluded from the Sun. It can refine the positions and orientations of the cameras.

The tool works by minimizing the cost function

$$\iint \sum_k \left[I_k(\phi)(x,y) - T_k A(x,y) R_k(\phi)(x,y)\right]^2 + \mu \left\|\nabla^2 \phi(x,y)\right\|^2 + \lambda \left[\phi(x,y) - \phi_0(x,y)\right]^2 dx\, dy. \quad (10.1)$$

Here, $I_k(\phi)(x,y)$ is the k-th camera image interpolated at pixels obtained by projecting into the camera 3D points from the terrain $\phi(x,y)$, T_k is the k-th image exposure, $A(x,y)$ is the per-pixel albedo, $R_k(\phi)(x,y)$ is the reflectance computed from the terrain for k-th image, $\left\|\nabla^2 \phi(x,y)\right\|^2$ is the sum of squares of all second-order partial derivatives of ϕ, $\mu > 0$ is a smoothing term, and $\lambda > 0$ determines how close we should stay to the input terrain ϕ_0 (smaller μ will show more detail but may introduce some artifacts, and smaller λ may allow for more flexibility in optimization but the terrain may move too far from the input).

We use either the regular Lambertian reflectance model, or the Lunar-Lambertian model [90], more specifically as given in [80] (equations (3) and (4)). Also supported is the Hapke model, [66], [37], [52], [53]. Custom values for the coefficients of these models can be passed to the program.

10.1 How to get good test imagery

We obtain the images from http://wms.lroc.asu.edu/lroc/search (we search for EDR images of type NACL and NACR).

A faster (but not as complete) interface is provided by http://ode.rsl.wustl.edu/moon/indexproductsearch.aspx. The related site http://ode.rsl.wustl.edu/moon/indextools.aspx?displaypage=lolardr can provide LOLA datasets which can be used as (sparse) ground truth.

We advise the following strategy for picking images. First choose a small longitude-latitude window in which to perform a search for imagery. Pick two images that are very close in time and with a big amount of overlap (ideally they would have consecutive orbit numbers). Those can be passed to ASP's `stereo` tool to create an initial DEM. Then, search for other images close to the center of the maximum overlap of the first two images. Pick one or more of those, ideally with different illumination conditions than the first two. Those (together with one of the first two images) can be used for SfS.

To locate the area of spatial overlap, the images can be map-projected (either with `cam2map` with a coarse resolution) or with `mapproject`, using for example the LOLA DEM as the terrain to project onto, or the DEM obtained from running `stereo` on those images. Then the images can be overlayed in `stereo_gui`. A good sanity check is to examine the shadows in various images. If they point in different directions in the images and perhaps also have different lengths, that means that illumination conditions are different enough, which will help constrain the `sfs` problem better.

10.2 Running sfs at 1 meter/pixel using a single image

In both this and the next sections we will work with LRO NAC images taken close to the Lunar South Pole, at a latitude of 85° South (the tool was tested on equatorial regions as well). We will use four images, M139939938LE, M139946735RE, M173004270LE, and M122270273LE.

We first retrieve the data sets.

```
  wget http://lroc.sese.asu.edu/data/LRO-L-LROC-2-EDR-V1.0/\
LROLRC_0005/DATA/SCI/2010267/NAC/M139939938LE.IMG
  wget http://lroc.sese.asu.edu/data/LRO-L-LROC-2-EDR-V1.0/\
LROLRC_0005/DATA/SCI/2010267/NAC/M139946735RE.IMG
  wget http://lroc.sese.asu.edu/data/LRO-L-LROC-2-EDR-V1.0/\
LROLRC_0009/DATA/SCI/2011284/NAC/M173004270LE.IMG
  wget http://lroc.sese.asu.edu/data/LRO-L-LROC-2-EDR-V1.0/\
LROLRC_0002/DATA/MAP/2010062/NAC/M122270273LE.IMG
```

Then we convert them to ISIS cubes, initialize the SPICE kernels, and perform radiometric calibration and echo correction. Here are the steps, illustrated on the first image:

```
  lronac2isis from = M139939938LE.IMG     to = M139939938LE.cub
  spiceinit   from = M139939938LE.cub
  lronaccal   from = M139939938LE.cub     to = M139939938LE.cal.cub
  lronacecho  from = M139939938LE.cal.cub to = M139939938LE.cal.echo.cub
```

We rename, for simplicity, the obtained four processed datasets to A.cub, B.cub, C.cub, and D.cub.

The first step is to run stereo to create an initial guess DEM. We picked for this the first two of these images. These form a stereo pair, that is, they have a reasonable baseline and sufficiently close times of acquisition (hence very similar illuminations). These conditions are necessary to obtain a good stereo result.

```
parallel_stereo --job-size-w 1024 --job-size-h 1024 A.cub B.cub    \
                --left-image-crop-win 0 7998 2728 2696             \
                --right-image-crop-win 0 9377 2733 2505            \
                --threads 16 --corr-seed-mode 1  --subpixel-mode 3 \
                run_full1/run
```

Next we create a DEM at 1 meter/pixel, which is about the resolution of the input images. We use the stereographic projection since this dataset is very close to the South Pole. Then we crop it to the region we'd like to do SfS on.

```
point2dem -r moon --stereographic --proj-lon 0 \
  --proj-lat -90 run_full1/run-PC.tif
gdal_translate -projwin -15471.9 150986 -14986.7 150549  \
  run_full1/run-DEM.tif run_full1/run-crop-DEM.tif
```

This creates a DEM of size 456×410 pixels.

Then we run `sfs`:

```
sfs -i run_full1/run-crop-DEM.tif A.cub -o sfs_ref1/run           \
   --reflectance-type 1                                           \
  --smoothness-weight 0.08 --initial-dem-constraint-weight 0.0001 \
  --max-iterations 10 --use-approx-camera-models                  \
  --use-rpc-approximation --crop-input-images
```

The smoothness weight is a parameter that needs tuning. If it is too small, SfS will return noisy results, if it is too large, too much detail will be blurred. Here we used the Lunar Lambertian model. The meaning of the other `sfs` options can be looked up in section A.35.

We show the results of running this program in figure 10.1. The left-most figure is the hill-shaded original DEM, which was obtained by running:

```
hillshade --azimuth 300 --elevation 20 run_full1/run-crop-DEM.tif \
  -o run_full1/run-crop-hill.tif
```

The second image is the hill-shaded DEM obtained after running `sfs` for 10 iterations.

The third image is, for comparison, the map-projection of A.cub onto the original DEM, obtained via the command:

```
mapproject --tr 1 run_full1/run-crop-DEM.tif A.cub A_map.tif --tile-size 128
```

The forth image is the colored absolute difference between the original DEM and the SfS output, obtained by running:

```
geodiff --absolute sfs_ref1/run-DEM-final.tif run_full1/run-crop-DEM.tif
colormap --min 0 --max 2 --colormap-style binary-red-blue \
  run-DEM-final__run-crop-DEM-diff.tif
```

It can be seen that the optimized DEM provides a wealth of detail and looks quite similar to the input image. It also did not diverge significantly from the input DEM. We will see in the next section that SfS is in fact able to make the refined DEM more accurate than the initial guess (as compared to some known ground truth), though that is not guaranteed, and most likely did not happen here where just one image was used.

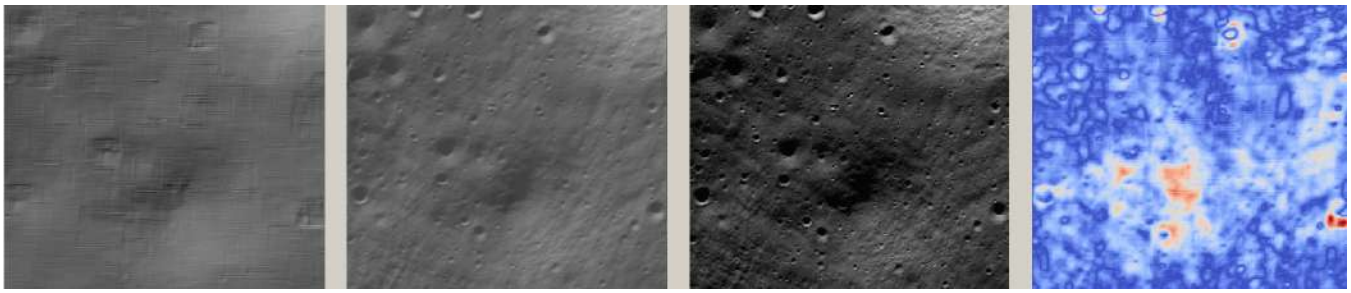


Figure 10.1: An illustration of `sfs`. The images are, from left to right, the original hill-shaded DEM, the hill-shaded DEM obtained from `sfs`, the image A.cub map-projected onto the original DEM, and the absolute difference of the original and final DEM, where the brightest shade of red corresponds to a 2 meter height difference.

10.3 SfS with multiple images in the presence of shadows

In this section we will run `sfs` with multiple images. We would like to be able to see if SfS improves the accuracy of the DEM rather than just adding detail to it. We evaluate this using the following (admittedly imperfect) approach. We resample the original images by a factor of 10, run stereo with them, followed by SfS using the stereo result as an initial guess and with the resampled images. As ground truth, we create a DEM from the original images at 1 meter/pixel, which we bring closer to the initial guess for SfS using `pc_align`. We would like to know if running SfS brings us even closer to this "ground truth" DEM.

The most significant challenge in running SfS with multiple images is that shape-from-shading is highly sensitive to errors in camera position and orientation. The `sfs` tool can improve these by floating them during optimization and by using a coarse-to-fine scheme, where the problem is first solved using subsampled images and terrain then it is successively refined.

If possible, it may still be desirable to bundle-adjust the cameras first (section A.4). It is important to note that bundle adjustment may fail if the images have sufficiently different illumination, as it will not be able to find matches among images. A solution to this is discussed in section 10.4.

To make bundle adjustment and stereo faster, we first crop the images, such as shown below (the crop parameters can be determined via `stereo_gui`).

```
crop from = A.cub to = A_crop.cub sample = 1 line = 6644 nsamples = 2192 nlines = 4982
crop from = B.cub to = B_crop.cub sample = 1 line = 7013 nsamples = 2531 nlines = 7337
crop from = C.cub to = C_crop.cub sample = 1 line = 1 nsamples = 2531 nlines = 8305
crop from = D.cub to = D_crop.cub sample = 1 line = 1 nsamples = 2531 nlines = 2740
```

Then we bundle-adjust and run stereo

```
bundle_adjust A_crop.cub B_crop.cub C_crop.cub D_crop.cub    \
  --min-matches 1 -o run_ba/run
stereo A_crop.cub B_crop.cub run_full2/run --subpixel-mode 3 \
  --bundle-adjust-prefix run_ba/run
```

This will result in a point cloud, `run_full2/run-PC.tif`, which will lead us to the "ground truth" DEM. As mentioned before, we'll in fact run SfS with images subsampled by a factor of 10. Subsampling is done by running the ISIS `reduce` command

```
for f in A B C D; do
  reduce from = ${f}_crop.cub to = ${f}_crop_sub10.cub sscale = 10 lscale = 10
done
```

We run bundle adjustment and stereo with the subsampled images using commands analogous to the above:

```
bundle_adjust A_crop_sub10.cub B_crop_sub10.cub C_crop_sub10.cub D_crop_sub10.cub \
  --min-matches 1 -o run_ba_sub10/run --ip-per-tile 100000
stereo A_crop_sub10.cub B_crop_sub10.cub run_sub10/run --subpixel-mode 3          \
  --bundle-adjust-prefix run_ba_sub10/run
```

We'll obtain a point cloud named `run_sub10/run-PC.tif`.

We'll bring the "ground truth" point cloud closer to the initial guess for SfS using `pc_align`:

```
pc_align --max-displacement 200 run_full2/run-PC.tif run_sub10/run-PC.tif \
  -o run_full2/run --save-inv-transformed-reference-points
```

This step is extremely important. Since we ran two bundle adjustment steps, and both were without ground control points, the resulting clouds may differ by a large translation, which we correct here. Hence we would like to make the "ground truth" terrain aligned with the datasets on which we will perform SfS.

Next we create the "ground truth" DEM from the aligned high-resolution point cloud, and crop it to a desired region:

```
point2dem -r moon --tr 10 --stereographic --proj-lon 0 --proj-lat -90 \
  run_full2/run-trans_reference.tif
gdal_translate -projwin -15540.7 151403 -14554.5 150473               \
  run_full2/run-trans_reference-DEM.tif run_full2/run-crop-DEM.tif
```

We repeat the same steps for the initial guess for SfS:

```
point2dem -r moon --tr 10 --stereographic --proj-lon 0 --proj-lat -90 \
  run_sub10/run-PC.tif
gdal_translate -projwin -15540.7 151403 -14554.5 150473               \
  run_sub10/run-DEM.tif run_sub10/run-crop-DEM.tif
```

After this, we run `sfs` itself. Since our dataset has many shadows, we found that specifying the shadow thresholds for the tool improves the results. The thresholds can be determined using `stereo_gui`. This can be done by turning on shadow-threshold mode from the GUI menu, and then clicking on a few points in the shadows. Then the thresholded images can be visualized/updated from the menu as well, and this process can be iterated.

```
sfs -i run_sub10/run-crop-DEM.tif A_crop_sub10.cub C_crop_sub10.cub \
  D_crop_sub10.cub -o sfs_sub10_ref1/run --threads 4                \
  --smoothness-weight 0.12 --initial-dem-constraint-weight 0.0001   \
  --reflectance-type 1 --float-exposure                             \
  --float-cameras --use-approx-camera-models                        \
  --max-iterations 10 --use-approx-camera-models                    \
  --use-rpc-approximation --crop-input-images                       \
  --bundle-adjust-prefix run_ba_sub10/run                           \
  --shadow-thresholds "0.00162484 0.0012166 0.000781663"
```

We compare the initial guess to `sfs` to the "ground truth" DEM obtained earlier and the same for the final refined DEM using `geodiff` as in the previous section. Before SfS:

```
geodiff --absolute run_full2/run-crop-DEM.tif run_sub10/run-crop-DEM.tif
gdalinfo -stats run-crop-DEM__run-crop-DEM-diff.tif | grep Mean=
```

and after SfS:

```
geodiff --absolute run_full2/run-crop-DEM.tif sfs_sub10_ref1/run-DEM-final.tif
gdalinfo -stats run-crop-DEM__run-DEM-final-diff.tif | grep Mean=
```

The mean error goes from 2.64 m to 1.29 m, while the standard deviation decreases from 2.50 m to 1.29 m. Visually the refined DEM looks more detailed as well as seen in figure 10.2. The same experiment can be repeated with the Lambertian reflectance model (reflectance-type 0), and then it is seen that it performs a little worse.

We also show in this figure the first of the images used for SfS, `A_crop_sub10.cub`, map-projected upon the optimized DEM. Note that we use the previously computed bundle-adjusted cameras when map-projecting, otherwise the image will show as shifted from its true location:

```
mapproject sfs_sub10_ref1/run-DEM-final.tif A_crop_sub10.cub A_crop_sub10_map.tif \
  --bundle-adjust-prefix run_ba_sub10/run
```

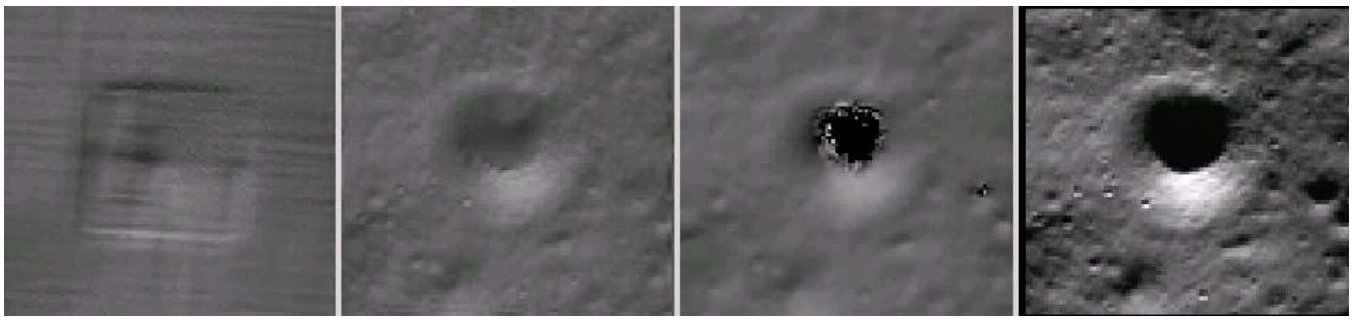


Figure 10.2: An illustration of `sfs`. The images are, from left to right, the hill-shaded initial guess DEM for SfS, the hill-shaded DEM obtained from `sfs`, the "ground truth" DEM, and the first of the images used in SfS map-projected onto the optimized DEM.

10.4 Dealing with large camera errors and LOLA comparison

SfS is very sensitive to errors in camera positions and orientations. These can be optimized as part of the problem, but if they are too far off, the solution will not be correct. In the previous section we used bundle adjustment to correct these errors, and then we passed the adjusted cameras to `sfs`. However, bundle adjustment may often fail, simply because the illumination conditions can be very different among the images, and interest point matching may not succeed.

The option `--coarse-levels` *int* can be passed to `sfs`, to solve for the terrain using a multi-resolution approach, first starting at a coarse level, where camera errors have less of an impact, and then jointly optimizing the cameras and the terrain at ever increasing levels of resolution. Yet, this may still fail if the terrain does not have large and pronounced features on the scale bigger than the errors in the cameras.

The approach that we found to work all the time is to manually select interest points in the images, as the human eye is much more skilled at identifying a given landmark in multiple images, even when the lightning changes drastically. Picking about 4 landmarks in each image is sufficient. Ideally they should be positioned far from each other, to improve the accuracy.

Below is one example of how we manually select interest points, run SfS, and then how we compare to LOLA, which is an independently acquired sparse dataset of 3D points on the Moon. According to [138], the LOLA accuracy is on the order of 1 m. To ensure a meaningful comparison of stereo and SfS with LOLA, we resample the LRO NAC images by a factor of 4, making them nominally 4 m/pixel. This is not strictly necessary, the same exercise can be repeated with the original images, but it is easier to see the improvement due to SfS when comparing to LOLA when the images are coarser than the LOLA error itself.

We work with the same images as before. To resample them, we do:

```
for f in A B C D; do
  reduce from = ${f}_crop.cub to = ${f}_crop_sub4.cub sscale=4 lscale=4
done
```

We run `stereo` and `point2dem` to get a first cut DEM. We don't do bundle adjustment at this stage yet.

```
stereo A_crop_sub4.cub B_crop_sub4.cub run_stereo_noba_sub4/run --subpixel-mode 3
point2dem --stereographic --proj-lon -5.7113451 --proj-lat -85.000351 \
  run_stereo_noba_sub4/run-PC.tif --tr 4
```

We would like now to manually pick interest points for the purpose of doing bundle adjustment. We found it it much easier to locate the landmarks if we first map-project the images, which brings them all into the same perspective. We then pick interest points in `stereo_gui`, and then project them back into the original cameras and do bundle adjustment. Here are the steps:

```
for f in A B C D; do
  mapproject --tr 4 run_stereo_noba_sub4/run-DEM.tif ${f}_crop_sub4.cub \
    ${f}_crop_sub4_v1.tif --tile-size 128
done
stereo_gui A_crop_sub4_v1.tif B_crop_sub4_v1.tif C_crop_sub4_v1.tif \
  D_crop_sub4_v1.tif run_ba_sub4/run
```

Interest points are selected by zooming and right-clicking with the mouse, one point at a time, from left to right, and then saving them. An illustration is shown in Figure 10.3.

Then bundle adjustment happens:

```
P='A_crop_sub4_v1.tif B_crop_sub4_v1.tif' # to avoid long lines below
Q='C_crop_sub4_v1.tif D_crop_sub4_v1.tif run_stereo_noba_sub4/run-DEM.tif'
bundle_adjust A_crop_sub4.cub B_crop_sub4.cub C_crop_sub4.cub D_crop_sub4.cub  \
  -o run_ba_sub4/run --mapprojected-data  "$P $Q"                               \
  --min-matches 1
```

A good sanity check to ensure that at this stage cameras are aligned properly is to map-project using the newly obtained camera adjustments and then overlay the obtained images in the GUI. The features in all images should be perfectly on top of each other.

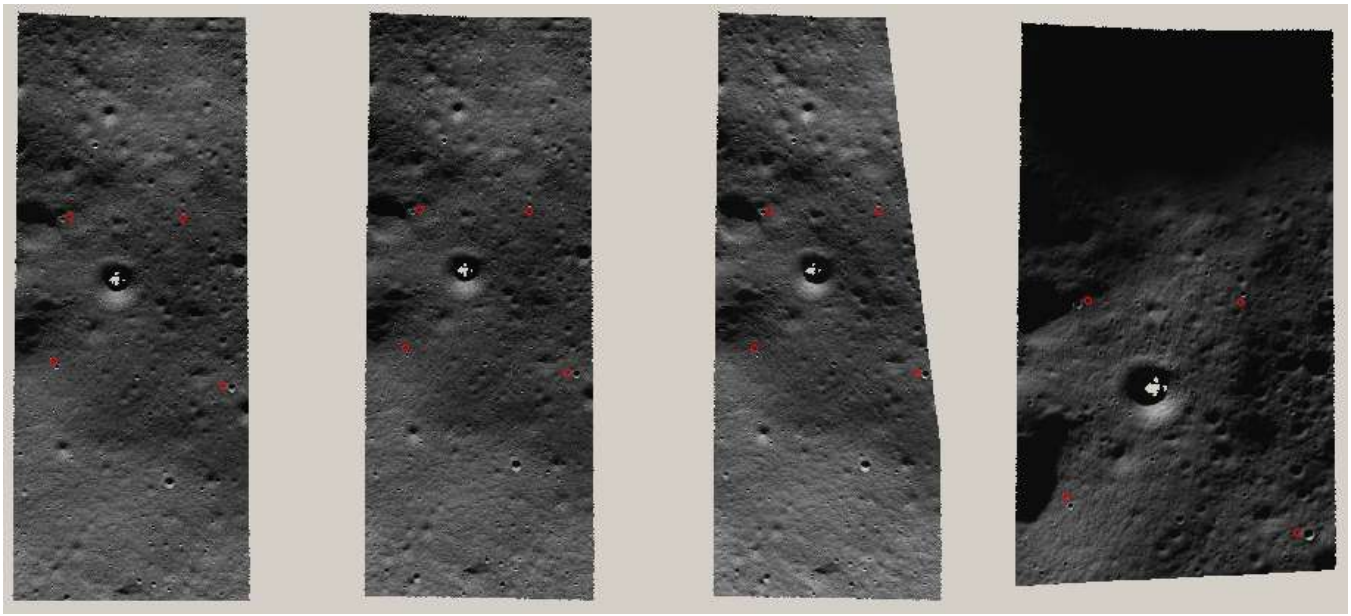


Figure 10.3: An illustration of how interest points are picked manually for the purpose of bundle adjustment and then SfS.

```
for f in A B C D; do
  mapproject --tr 4 run_stereo_noba_sub4/run-DEM.tif ${f}_crop_sub4.cub  \
   ${f}_crop_sub4_v2.tif --tile-size 128 --bundle-adjust-prefix run_ba_sub4/run
done
```

This will also show where the images overlap, and hence on what portion of the DEM we can run SfS.

Then we run stereo, followed by SfS.

```
stereo A_crop_sub4.cub B_crop_sub4.cub run_stereo_yesba_sub4/run             \
  --subpixel-mode 3 --bundle-adjust-prefix run_ba_sub4/run
point2dem --stereographic --proj-lon -5.7113451 --proj-lat -85.000351        \
  run_stereo_yesba_sub4/run-PC.tif --tr 4
gdal_translate -srcwin 138 347 273 506 run_stereo_yesba_sub4/run-DEM.tif     \
  run_stereo_yesba_sub4/run-crop1-DEM.tif
sfs -i run_stereo_yesba_sub4/run-crop1-DEM.tif A_crop_sub4.cub               \
  C_crop_sub4.cub D_crop_sub4.cub -o sfs_sub4_ref1_th_reg0.12_wt0.001/run    \
  --shadow-thresholds '0.00149055 0.00138248 0.000747531'                    \
  --threads 4 --smoothness-weight 0.12 --initial-dem-constraint-weight 0.001 \
  --reflectance-type 1 --float-exposure --float-cameras --max-iterations 20  \
  --use-approx-camera-models --use-rpc-approximation --crop-input-images     \
  --bundle-adjust-prefix run_ba_sub4/run
```

We fetch the portion of the LOLA dataset around the current DEM from the site described earlier, and save it as `RDR_354E355E_85p5S84SPointPerRow_csv_table.csv`. It is necessary to align our stereo DEM with this dataset to be able to compare them. We choose to bring the LOLA dataset into the coordinate system of the DEM, using:

```
pc_align --max-displacement 280 run_stereo_yesba_sub4/run-DEM.tif             \
```

```
    RDR_354E355E_85p5S84SPointPerRow_csv_table.csv -o run_stereo_yesba_sub4/run \
    --save-transformed-source-points
```

Then we compare to the aligned LOLA dataset the input to SfS and its output:

```
geodiff --absolute -o beg --csv-format '1:lon 2:lat 3:radius_km' \
  run_stereo_yesba_sub4/run-crop1-DEM.tif run_stereo_yesba_sub4/run-trans_source.csv
geodiff --absolute -o end --csv-format '1:lon 2:lat 3:radius_km' \
  sfs_sub4_ref1_th_reg0.12_wt0.001/run-DEM-final.tif             \
  run_stereo_yesba_sub4/run-trans_source.csv
```

We see that the mean error between the DEM and LOLA goes down, after SfS, from 1.14 m to 0.90 m, while the standard deviation decreases from 1.18 m to 1.06 m.

10.5 Running SfS with an external initial guess DEM

Sometimes it is convenient to run SfS with a DEM not created using ASP's `stereo`. For example, for the Moon, the LOLA gridded DEM is available. It is somewhat noisy and the nominal resolution is on the order of 10 m/pixel, but it is available even for permanently shadowed regions.

The main challenge in such a situation is that the images, such as coming from LRO NAC, may not be aligned well to this external DEM and among themselves, and then SfS will fail. To get an initial alignment, what worked for us to coarsen both this DEM and the images to about 40 meters/pixel (using the ISIS `reduce` command and `dem_mosaic` with the options to blur and change the grid size), and then run SfS over a reasonably large area (say about 500×500 pixels) with pronounced terrain and where images have a lot of overlap with the option `--float-all-cameras` (and of course `--float-exposure`, and an appropriate `--shadow-threshold`). Then, the SfS program will find adjustments to the cameras, writing them in the output directory. The `mapproject` tool can be used to map-project the coarse images onto the input DEM using `--bundle-adjust-prefix` pointing to the `sfs` output prefix. This should be used to verify that images are now aligned correctly, for example by overlaying them in `stereo_gui`. If so, these adjustments can be used as input for SfS with images at finer levels of resolution (after appropriately renaming the adjustment files and using the `--bundle-adjust-prefix` option of `sfs`).

Here, a higher value can be used for `--initial-dem-constraint-weight` to ensure we cameras have more motivation to align to the terrain.

When it comes to running SfS with many large high-resolution images, one runs very fast into memory constraints. It is then necessary to parallelize the problem using `parallel_sfs`, which will run `sfs` on multiple tiles. Yet, the camera adjustments need to be determined before running this tool, and then kept fixed when this tool is run, as otherwise each single tile will optimize its cameras independently, and as result there will be discontinuities at tile boundaries.

If the camera adjustments are determined by first running `sfs` on a clip representative of the entire terrain, far from that clip the cameras will start to disagree. For that reason, `sfs` makes it possible to optimize the cameras on an entire collection of clips, chosen so they are reasonably spread out over the entire terrain, and that each camera image is covered by at least a handful of such clips. The clips can be passed to `sfs` as a quoted string via the `-i` option. As before, at the end the `mapproject` program can be used to verify that the camera images mapprojected using the obtained camera adjustments are perfectly on top of each other, and if not, more clips can be added to the joint optimization problem. When a good enough set of camera adjustments is obtained, `parallel_sfs` can be run as before.

10.6 Insights for getting the most of SfS

Here are a few suggestions we have found helpful when running `sfs`:

- First determine the appropriate smoothing weight μ by running a small clip, and using just one image. A value between 0.06 and 0.12 seems to work all the time with LRO NAC, even when the images are subsampled. The other weight, λ, can be set to something small, like 0.0001. This can be increased to 0.001 if noticing that the output DEM strays too far.
- As stated before, more images with more diverse illumination conditions result in more accurate terrain. Ideally there should be at least 3 images, with the shadows being, respectively, on the left, right, and then perhaps missing or small.
- Bundle-adjustment for multiple images is crucial, to eliminate camera errors which will result in `sfs` converging to a local minimum. This is described in section 10.4.
- Floating the albedo (option `--float-albedo`) can introduce instability and divergence, it should be avoided unless obvious albedo variation is seen in the images.
- Floating the DEM at the boundary (option `--float-dem-at-boundary`) is also suggested to be avoided.
- Overall, the best strategy is to first use SfS for a single image and not float any variables except the DEM being optimized, and then gradually add images and float more variables and select whichever approach seems to give better results.
- If an input DEM is large, it may not be completely covered by a single set of imagery with various illumination conditions. It should then be broken up into smaller regions (with overlap), the SfS problem can be solved on each region, and then every output terrain can be transformed using `pc_align` into LOLA's global coordinate system, where they can be mosaicked together using `dem_mosaic`. Or, `sfs` can be run not on one clip, but on an entire collection of clips covering this area to get the adjustments, and then `parallel_sfs` can be run as described in the previous section.

  The easier case is when at least the two images in the stereo pair cover the entire terrain. Then, portions of this terrain can be used as an initial guess for each SfS sub-problem (even as the other images used for SfS change), the results can be mosaicked, and the alignment to LOLA can happen just once, after mosaicking. This approach is preferable, if feasible, as alignment to LOLA is more accurate if the terrain to align is larger in extent.
- The `mapproject` program can be used to map-project each image onto the resulting SfS DEM (with the camera adjustments solved using SfS). These orthoimages can be mosaicked using `dem_mosaic`. If the `--max` option is used with this tool, it create a mosaic with the most illuminated pixels from this image. If during SfS the camera adjustments were solved accurately, this mosaic should have little or no blur. An alterantive is to use the `--block-max` option which will pick the most lit of the images per each block, with the latter being specified via `--block-size`.

Chapter 11

Data Processing Examples

This chapter showcases a variety of results that are possible when processing different data sets with the Stereo Pipeline. It is also a shortened guide that shows the commands used to process specific mission data. There is no definitive method yet for making elevation models as each stereo pair is unique. We hope that the following sections serve as a cookbook for strategies that will get you started in processing your own data. We recommend that you second check your results against another source.

11.1 Guidelines for Selecting Stereo Pairs

When choosing image pairs to process, images that are taken with similar viewing angles, lighting conditions, and significant surface coverage overlap are best suited for creating terrain models. Depending on the characteristics of the mission data set and the individual images, the degree of acceptable variation will differ. Significant differences between image characteristics increases the likelihood of stereo matching error and artifacts, and these errors will propagate through to the resulting data products.

Although images do not need to be map-projected before running the `stereo` program, we recommend that you do run `cam2map` (or `cam2map4stereo.py`) beforehand, especially for image pairs that contain large topographic variation (and therefore large disparity differences across the scene, e.g., Valles Marineris). Map-projection is especially necessary when processing HiRISE images. This removes the large disparity differences between HiRISE images and leaves only the small detail for the Stereo Pipeline to compute. Remember that ISIS can work backwards through a map-projection when applying the camera model, so the geometric integrity of your images will not be sacrificed if you map-project first.

An alternative way of map-projection, that applies to non-ISIS imagery as well, is with the `mapproject` tool (section 5.1.7).

Excessively noisy images will not correlate well, so images should be photometrically calibrated in whatever fashion suits your purposes. If there are photometric problems with the images, those photometric defects can be misinterpreted as topography.

Remember, in order for `stereo` to process stereo pairs in ISIS cube format, the images must have had SPICE data associated by running ISIS's `spiceinit` program run on them first.

11.2 Mars Reconnaissance Orbiter HiRISE

HiRISE is one of the most challenging cameras to use when making 3D models because HiRISE exposures can be several gigabytes each. Working with this data requires patience as it will take time.

One important fact to know about HiRISE is that it is composed of multiple linear CCDs that are arranged side by side with some vertical offsets. These offsets mean that the CCDs will view some of the same terrain but at a slightly different time and a slightly different angle. Mosaicking the CCDs together to a single image is not a simple process and involves living with some imperfections.

One cannot simply use the HiRISE RDR products, as they do not have the required geometric stability. Instead, the HiRISE EDR products must be assembled using ISIS `noproj`. The USGS distributes a script in use by the HiRISE team that works forward from the team-produced 'balance' cubes, which provides a de-jittered, noproj'ed mosaic of a single observation, which is perfectly suitable for use by the Stereo Pipeline (this script was originally engineered to provide input for SOCET SET). However, the 'balance' cubes are not available to the general public, and so we include a program (`hiedr2mosaic.py`, written in Python) that will take PDS available HiRISE EDR products and walk through the processing steps required to provide good input images for `stereo`.

The program takes all the red CCDs and projects them using the ISIS `noproj` command into the perspective of the RED5 CCD. From there, `hijitreg` is performed to work out the relative offsets between CCDs. Finally the CCDs are mosaicked together using the average offset listed from `hijitreg` using the `handmos` command, and the mosaic is normalized with `cubenorm`. Below is an outline of the processing.

```
hi2isis           # Import HiRISE IMG to Isis
hical             # Calibrate
histitch          # Assemble whole-CCD images from the channels
spiceinit
spicefit          # For good measure
noproj            # Project all images into perspective of RED5
hijitreg          # Work out alignment between CCDs
handmos           # Mosaic to single file
cubenorm          # Normalize the mosaic
```

To use our script, first download a set of HiRISE data. Here is an example, using wget to fetch all RED CCDs for a dataset and process them.

```
wget -r -l1 -np \
  "http://hirise-pds.lpl.arizona.edu/PDS/EDR/ESP/ORB_029400_029499/ESP_029421_2300/" \
  -A "*RED*IMG"
```

Alternately, you can pass the `--download-folder` option to `hiedr2mosaic.py` and pass in the URL of the web page containing the EDR files as the only positional argument. This will cause the tool to first download all of the RED CCD images to the specified folder and then continue with processing.

```
hiedr2mosaic.py --download-folder hirise_example/ \
  http://hirise-pds.lpl.arizona.edu/PDS/EDR/ESP/ORB_029400_029499/ESP_029421_2300/
```

Assuming you downloaded the files manually, go to the directory containing the files. You can run the `hiedr2mosaic.py` program without any arguments to view a short help statement, with the `-h` option to view a longer help statement, or just run the program on the EDR files like so:

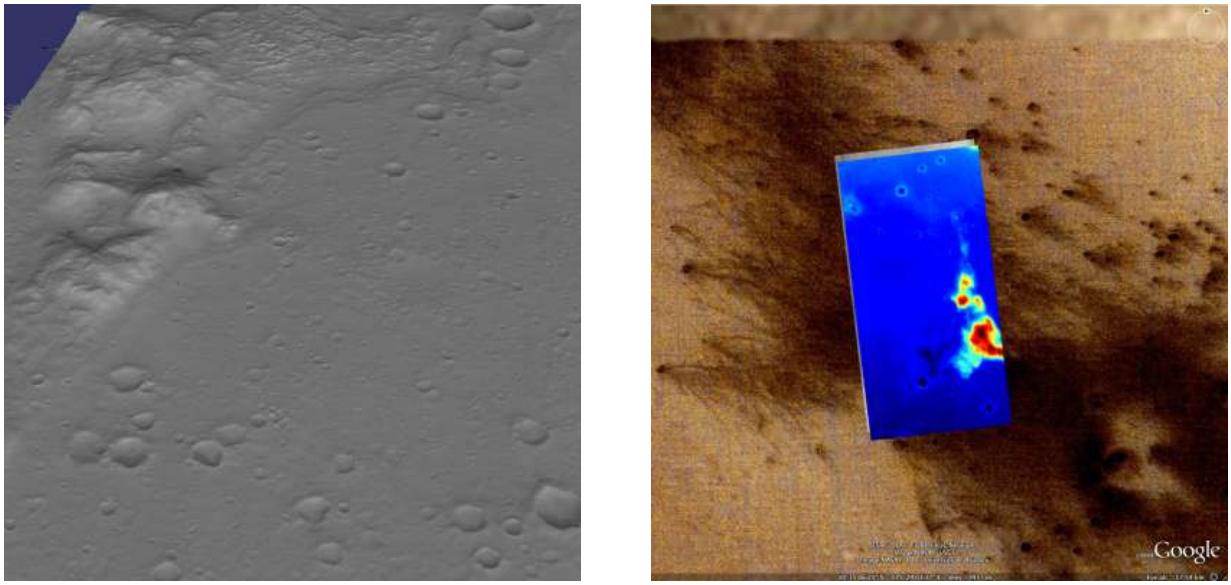
```
hiedr2mosaic.py *.IMG
```

If you have more than one observation's worth of EDRs in that directory, then limit the program to just one observation's EDRs at a time, e.g. `hiedr2mosaic.py PSP_001513_1655*IMG`. If you run into problems, try using the `-k` option to retain all of the intermediary image files to help track down the issue. The `hiedr2mosaic.py` program will create a single mosaic file with the extension `.mos_hijitreged.norm.cub`. Be warned that the operations carried out by `hiedr2mosaic.py` can take many hours to complete on the very large HiRISE images.

An example of using ASP with HiRISE data is included in the `examples/HiRISE` directory (just type 'make' there).

11.2.1 Columbia Hills

HiRISE observations PSP_001513_1655 and PSP_001777_1650 are on the floor of Gusev Crater and cover the area where the MER Spirit landed and has roved, including the Columbia Hills.



(a) `3D Rendering` (b) `KML Screenshot`

Figure 11.1: Example output using HiRISE images PSP_001513_1655 and PSP_001777_1650 of the Columbia Hills.

Commands

Download all 20 of the RED EDR `.IMG` files for each observation.

```
ISIS 3> hiedr2mosaic.py PSP_001513_1655_RED*.IMG
ISIS 3> hiedr2mosaic.py PSP_001777_1650_RED*.IMG
ISIS 3> cam2map4stereo.py PSP_001777_1650_RED.mos_hijitreged.norm.cub \
                          PSP_001513_1655_RED.mos_hijitreged.norm.cub
ISIS 3> stereo PSP_001513_1655.map.cub \
               PSP_001777_1650.map.cub result/output
```

stereo.default

The stereo.default example file (appendix B) should apply well to HiRISE. Just set `alignment-method` to `none` if using map-projected imagery. If you are not using map-projected imagery, set `alignment-method` to `homography` or `affineepipolar`. The `corr-kernel` value can usually be safely reduced to 21 pixels to resolve finer detail and faster processing for images with good contrast.

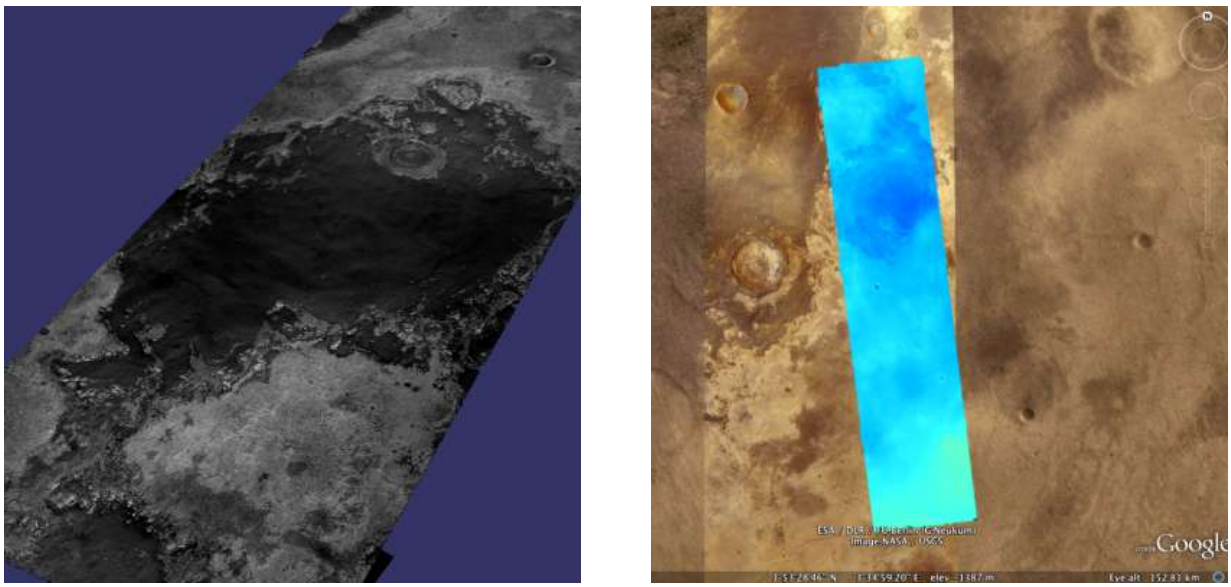
11.3 Mars Reconnaissance Orbiter CTX

Context Camera (CTX) is a moderate camera to work with. Processing times for CTX can be pretty long when using Bayes EM subpixel refinement. Otherwise the disparity between images is relatively small, allowing efficient computation and a reasonable processing time.

11.3.1 North Terra Meridiani

In this example, we use map-projected images. Map-projecting the images is the most reliable way to align the images for correlation. However when possible, use non-map-projected images with the `alignment-method affineepipolar` option. This greatly reduces the time spent in triangulation. For all cases using linescan cameras, triangulation of map-projected images is 10x slower than non-map-projected images.

This example is distributed in the `examples/CTX` directory (type 'make' there to run it).



(a) `3D Rendering` (b) `KML Screenshot`

Figure 11.2: Example output possible with the CTX imager aboard MRO.

Commands

Download the CTX images P02_001981_1823_XI_02N356W.IMG and P03_002258_1817_XI_01N356W.IMG from the PDS.

```
ISIS 3> mroctx2isis from=P02_001981_1823_XI_02N356W.IMG to=P02_001981_1823.cub
ISIS 3> mroctx2isis from=P03_002258_1817_XI_01N356W.IMG to=P03_002258_1817.cub
ISIS 3> spiceinit from=P02_001981_1823.cub
ISIS 3> spiceinit from=P03_002258_1817.cub
ISIS 3> ctxcal from=P02_001981_1823.cub to=P02_001981_1823.cal.cub
ISIS 3> ctxcal from=P03_002258_1817.cub to=P03_002258_1817.cal.cub
  you can also optionally run ctxevenodd on the cal.cub files, if needed
ISIS 3> cam2map4stereo.py P02_001981_1823.cal.cub P03_002258_1817.cal.cub
ISIS 3> stereo P02_001981_1823.map.cub P03_002258_1817.map.cub results/out
```

stereo.default

The stereo.default example file (appendix B) works generally well with all CTX pairs. Just set `alignment-method` to `homography` or `affineepipolar`.

11.4 Mars Global Surveyor MOC-NA

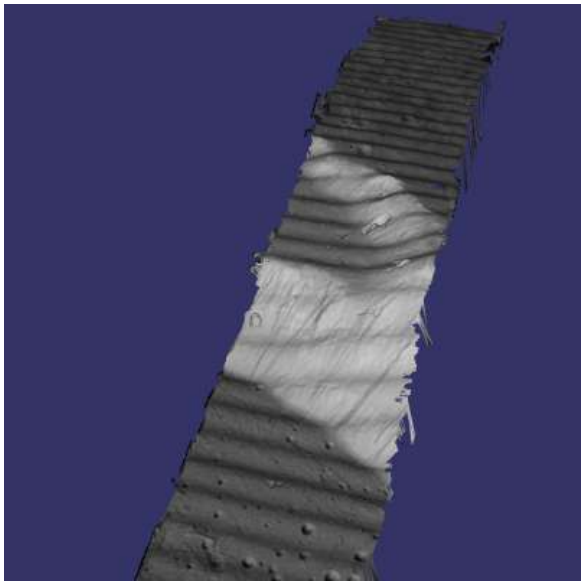
In the Stereo Pipeline Tutorial in Chapter 3, we showed you how to process a narrow angle MOC stereo pair that covered a portion of Hrad Vallis. In this section we will show you more examples, some of which exhibit a problem common to stereo pairs from linescan imagers: "spacecraft jitter" is caused by oscillations of the spacecraft due to the movement of other spacecraft hardware. All spacecraft wobble around to some degree but some are particularly susceptible.

Jitter causes wave-like distortions along the track of the satellite orbit in DEMs produced from linescan camera images. This effect can be very subtle or quite pronounced, so it is important to check your data products carefully for any sign of this type of artifact. The following examples will show the typical distortions created by this problem.

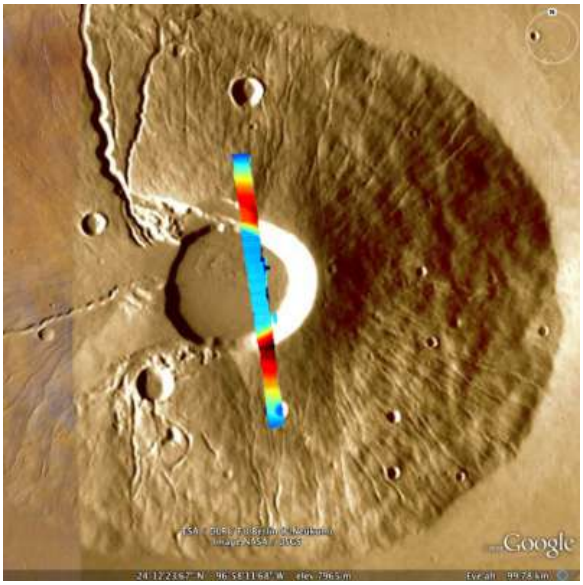
Note that the science teams of HiRISE and Lunar Reconnaissance Orbiter Camera (LROC) are actively working on detecting and correctly modeling jitter in their respective SPICE data. If they succeed in this, the distortions will still be present in the raw imagery, but the jitter will no longer produce ripple artifacts in the DEMs produced using ours or other stereo reconstruction software.

11.4.1 Ceraunius Tholus

Ceraunius Tholus is a volcano in northern Tharsis on Mars. It can be found at 23.96 N and 262.60 E. This DEM crosses the volcano's caldera.



(a) `3D Rendering`



(b) `KML Screenshot`

Figure 11.3: Example output for MOC-NA of Ceraunius Tholus. Notice the presence of severe washboarding artifacts due to spacecraft "jitter."

Commands

Download the M08/06047 and R07/01361 images from the PDS.

```
ISIS 3> moc2isis f=M0806047.img t=M0806047.cub
```

```
ISIS 3> moc2isis f=R0701361.img t=R0701361.cub
ISIS 3> spiceinit from=M0806047.cub
ISIS 3> spiceinit from=R0701361.cub
ISIS 3> cam2map4stereo.py M0806047.cub R0701361.cub
ISIS 3> stereo M0806047.map.cub R0701361.map.cub result/output
```

stereo.default

The stereo.default example file (appendix B) works generally well with all MOC-NA pairs. Just set `alignment-method` to `none` when using map-projected imagery. If the images are not map-projected, use `homography` or `affineepipolar`.

11.5 Mars Exploration Rovers

The Mars Exploration Rovers (MER) have several cameras on board and they all seem to have a stereo pair. With ASP you are able to process the PANCAM, NAVCAM, and HAZCAM camera imagery. ISIS has no telemetry or camera intrinsic supports for these images. That however is not a problem as their raw imagery contains the cameras' information in JPL's CAHV, CAHVOR, and CHAVORE formats.

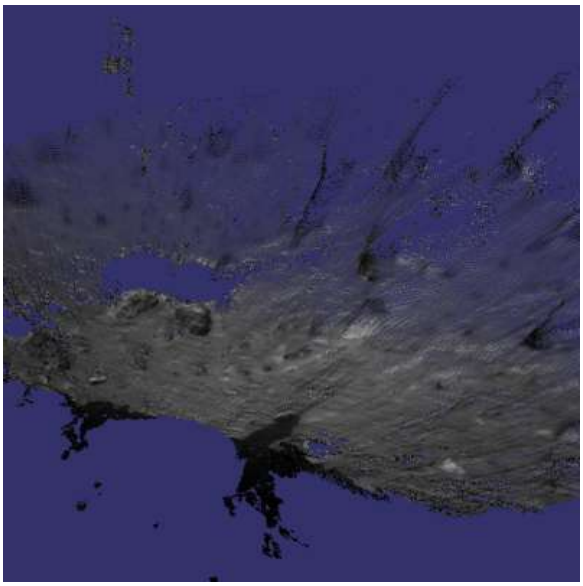
These cameras are all variations of a simple pinhole camera model so they are processed with ASP in the `Pinhole` session instead of the usual `ISIS`. ASP only supports creating of point clouds. *The *-PC.tif is a raw point cloud with the first 3 channels being XYZ in the rover site's coordinate frame.* We don't support the creation of DEMs from these images and that is left as an exercise for the user.

An example of using ASP with MER data is included in the `examples/MER` directory (just type 'make' there).

11.5.1 PANCAM, NAVCAM, HAZCAM

All of these cameras are processed the same way. We'll be showing 3D processing of the front hazard cams. The only new things in the pipeline is the new executable `mer2camera` along with the use of `alignment-method epipolar`. This example is also provided in the MER data example directory.

(a) `Rectified Input`



(b) `Output Point Cloud`

Figure 11.4: Example output possible with the front hazard cameras.

Commands

Download 2f194370083effap00p1214l0m1.img and 2f194370083effap00p1214r0m1.img from the PDS.

```
ISIS 3> mer2camera 2f194370083effap00p1214l0m1.img
ISIS 3> mer2camera 2f194370083effap00p1214r0m1.img
ISIS 3> stereo 2f194370083effap00p1214l0m1.img 2f194370083effap00p1214r0m1.img \
               2f194370083effap00p1214l0m1.cahvore 2f194370083effap00p1214r0m1.cahvore \
               fh01/fh01
```

stereo.default

The default stereo settings will work but change the following options. The universe option filters out points that are not triangulated well because they are too close *robot's hardware* or are extremely far away.

additional settings for MER

```
alignment-method epipolar
force-use-entire-range

# This deletes points that are too far away
# from the camera to truly triangulate.
universe-center Camera
near-universe-radius 0.7
far-universe-radius 80.0
```

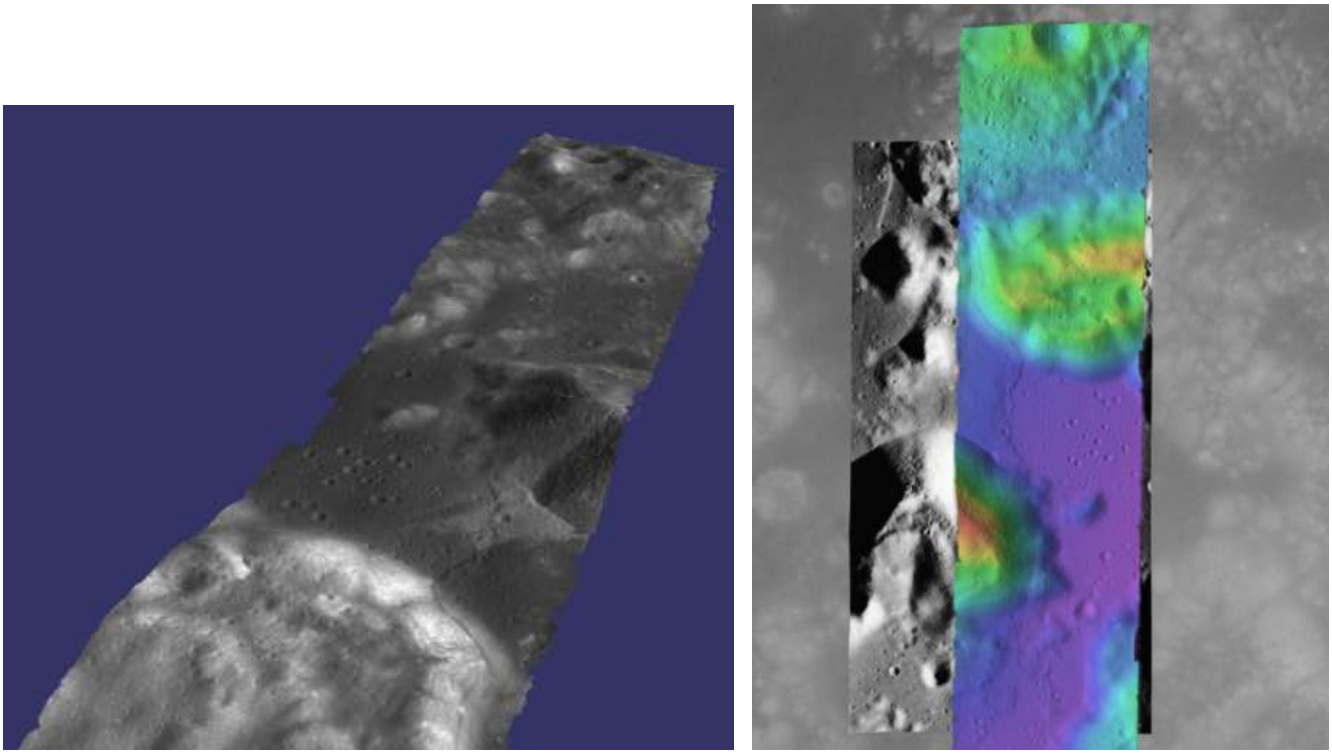
11.6 K10

K10 is an Earth-based research rover within the Intelligent Robotics Group at NASA Ames, the group ASP developers belong to. The cameras on this rover use a simple Pinhole model. The use of ASP with these cameras is illustrated in the `examples/K10` directory (just type 'make' there). Just as for the MER datatset (section 11.5), only the creation of a point cloud is supported.

11.7 Lunar Reconnaissance Orbiter LROC NAC

11.7.1 Lee-Lincoln Scarp

This stereo pair covers the Taurus-Littrow valley on the Moon where, on December 11, 1972, the astronauts of Apollo 17 landed. However, this stereo pair does not contain the landing site. It is slightly west; focusing on the Lee-Lincoln scarp that is on North Massif. The scarp is an 80 m high feature that is the only visible sign of a deep fault.



(a) `3D Rendering` (b) `KML Screenshot`

Figure 11.5: Example output possible with a LROC NA stereo pair, using both CCDs from each observation courtesy of the lronac2mosaic.py tool.

Commands

Download the EDRs for the left and right CCDs for observations M104318871 and M104318871 from http://wms.lroc.asu.edu/lroc/search. Alternatively you can search by original IDs of 2DB8 and 4C86 in the PDS.

All ISIS preprocessing of the EDRs is performed via the `lronac2mosaic.py` command. This runs `lronac2isis`, `lronaccal`, `lronacecho`, `spiceinit`, `noproj`, and `handmos` to create a stitched unprojected image for a single observation. In this example we don't map-project the images as ASP can usually get good results. More aggressive terrain might require an additional `cam2map4stereo.py` step.

```
ISIS 3> lronac2mosaic.py M104318871LE.img M104318871RE.img
ISIS 3> lronac2mosaic.py M104311715LE.img M104311715RE.img
ISIS 3> stereo M104318871LE*.mosaic.norm.cub M104311715LE*.mosaic.norm.cub \
          result/output --alignment-method affineepipolar
```

stereo.default

The defaults work generally well with LRO-NAC pairs, so you don't need to provide a stereo.default file. Map-projecting is optional. When map-projecting the images use `alignment-method none`, otherwise use `alignment-method affineepipolar`. Better map-project results can be achieved by projecting on a higher resolution elevation source like the WAC DTM. This is achieved using the ISIS command `demprep` and attaching to cube files via `spiceinit`'s SHAPE and MODEL options.

11.8 Apollo 15 Metric Camera Images

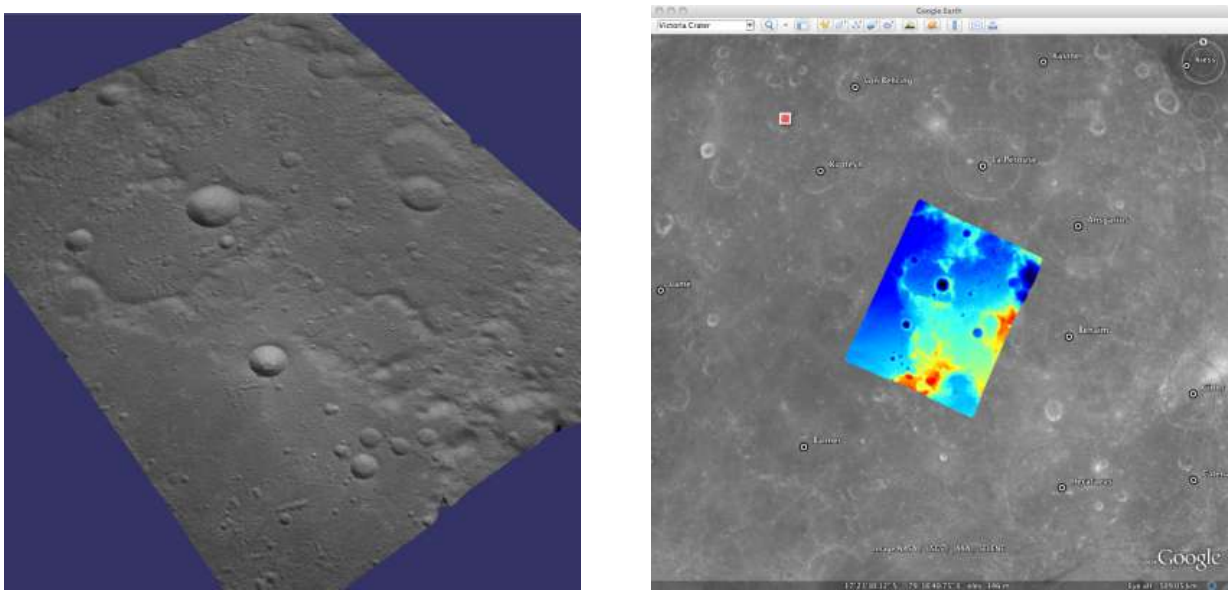
Apollo Metric images were all taken at regular intervals, which means that the same `stereo.default` can be used for all sequential pairs of images. Apollo Metric images are ideal for stereo processing. They produce consistent, excellent results.

The scans performed by ASU are sufficiently detailed to exhibit film grain at the highest resolution. The amount of noise at the full resolution is not helpful for the correlator, so we recommend subsampling the images by a factor of 4.

Currently the tools to ingest Apollo TIFFs into ISIS are not available, but these images should soon be released into the PDS for general public usage.

11.8.1 Ansgarius C

Ansgarius C is a small crater on the west edge of the far side of the Moon near the equator. It is east of Kapteyn A and B.



(a) `3D Rendering` (b) `KML Screenshot`

Figure 11.6: Example output possible with Apollo Metric frames AS15-M-2380 and AS15-M-2381.

Commands

Process Apollo TIFF files into ISIS.

```
ISIS 3> reduce from=AS15-M-2380.cub to=sub4-AS15-M-2380.cub sscale=4 lscale=4
ISIS 3> reduce from=AS15-M-2381.cub to=sub4-AS15-M-2381.cub sscale=4 lscale=4
ISIS 3> spiceinit from=sub4-AS15-M-2380.cub
ISIS 3> spiceinit from=sub4-AS15-M-2381.cub
ISIS 3> stereo sub4-AS15-M-2380.cub sub4-AS15-M-2381.cub result/output
```

stereo.default

The stereo.default example file (appendix B) works generally well with all Apollo pairs. Just set `alignment-method` to `homography` or `affineepipolar`.

11.9 Mars Express High Resolution Stereo Camera (HRSC)

The HRSC camera on the Mars Express satellite is a complicated system, consisting of multiple channels pointed in different directions plus another super resolution channel. The best option to create DEMs is to use the two dedicated stereo channels. These are pointed ahead of and behind the nadir channel and collect a stereo observation in a single pass of the satellite. Data can be downloaded from the Planetary Data System (PDS) http://pds-geosciences.wustl.edu/missions/mars_express/hrsc.htm or you can use the online graphical tool located at http://hrscview.fu-berlin.de/cgi-bin/ion-p?page=entry2.ion. Since each observation contains both stereo channels, one observation is sufficient to create a DEM.

HRSC data is organized into categories. Level 2 is radiometrically corrected, level 3 is corrected and map projected onto MOLA, and level 4 is corrected and map projected on to a DEM created from the HRSC data. You should use the level 2 data for creating DEMs with ASP. If you would like to download one of the already created DEMs, it may be easiest to use the areoid referenced version (.da4 extension) since that is consistent with MOLA.

What follows is an example for how to process HRSC data. One starts by fetching the two stereo channels from:

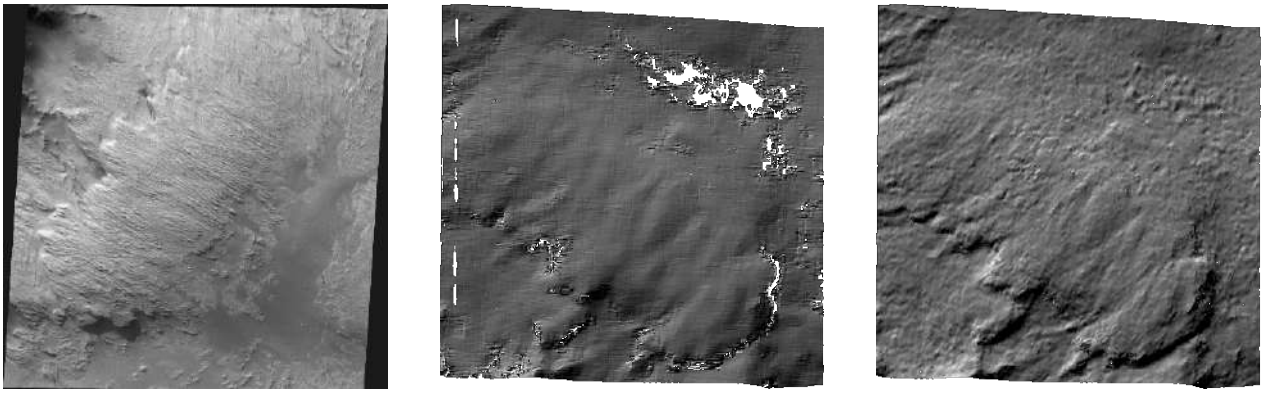
```
http://pds-geosciences.wustl.edu/mex/mex-m-hrsc-3-rdr-v3/mexhrs_1001/data/1995/h1995_0000_s12.img
http://pds-geosciences.wustl.edu/mex/mex-m-hrsc-3-rdr-v3/mexhrs_1001/data/1995/h1995_0000_s22.img
```

Commands

You may need to download the HRSC kernel files in case using `web=true` with `spiceinit` does not work. You will also probably need to include the `ckpredicted=true` flag with `spiceinit`. HRSC images are large and may have compression artifacts so you should experiment on a small region to make sure your stereo parameters are working well. For this frame, the MGM stereo algorithm performed better than block matching with subpixel mode 3.

```
ISIS 3> hrsc2isis from=h1995_0000_s12.img to=h1995_0000_s12.cub
ISIS 3> hrsc2isis from=h1995_0000_s22.img to=h1995_0000_s22.cub
ISIS 3> spiceinit from=h1995_0000_s12.cub ckpredicted=true
ISIS 3> spiceinit from=h1995_0000_s22.cub ckpredicted=true
```

```
ISIS 3> stereo h1995_0000_s12.cub  h1995_0000_s22.cub \
         --stereo-algorithm 2 --cost-mode 3 mgm/out
```



(a) `Cropped input` (b) `Block matching with subpixel mode 3` (c) `MGM algorithm with cost mode 3`

Figure 11.7: Sample outputs from a cropped region of HRSC frame 1995

11.10 Cassini ISS NAC

This is a proof of concept showing the strength of building the Stereo Pipeline on top of ISIS. Support for processing ISS NAC stereo pairs was not a goal during our design of the software, but the fact that a camera model exists in ISIS means that it too can be processed by the Stereo Pipeline.

Identifying stereo pairs from spacecraft that do not orbit their target is a challenge. We have found that one usually has to settle with images that are not ideal: different lighting, little perspective change, and little or no stereo parallax. So far we have had little success with Cassini's data, but nonetheless we provide this example as a potential starting point.

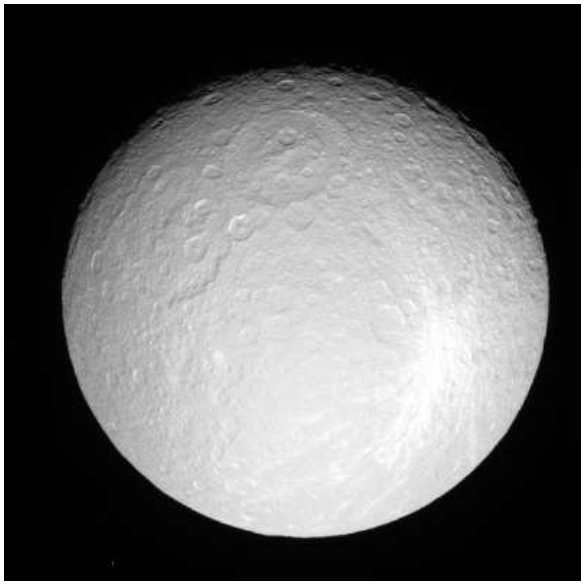
11.10.1 Rhea

Rhea is the second largest moon of Saturn and is roughly a third the size of our own Moon. This example shows, at the top right of both images, a giant impact basin named Tirawa that is 220 miles across. The bright white area south of Tirawa is ejecta from a new crater. The lack of texture in this area poses a challenge for our correlator. The results are just barely useful: the Tirawa impact can barely be made out in the 3D data while the new crater and ejecta become only noise.

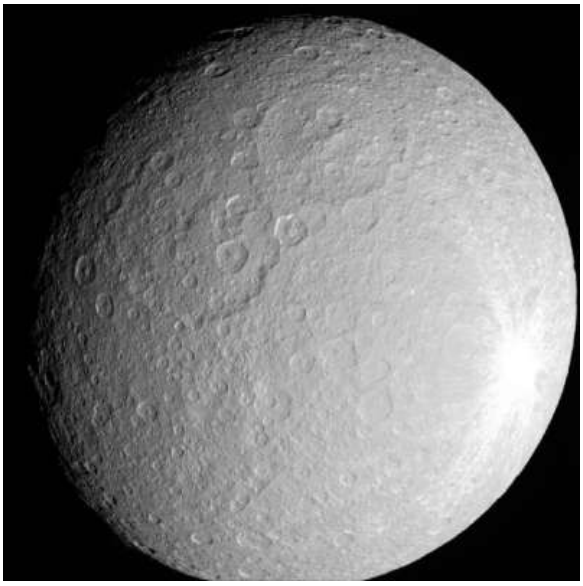
Commands

Download the N1511700120_1.IMG and W1567133629_1.IMG images and their label (.LBL) files from the PDS.

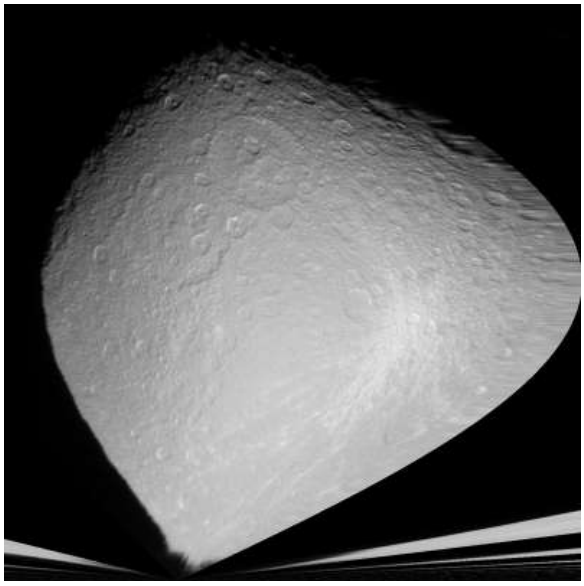
```
ISIS 3> ciss2isis f=N1511700120_1.LBL t=N1511700120_1.cub
ISIS 3> ciss2isis f=W1567133629_1.LBL t=W1567133629_1.cub
ISIS 3> cisscal from=N1511700120_1.cub to=N1511700120_1.lev1.cub
ISIS 3> cisscal from=W1567133629_1.cub to=W1567133629_1.lev1.cub
ISIS 3> fillgap from=W1567133629_1.lev1.cub to=W1567133629_1.fill.cub %Only one image
                                                                     %exhibits the problem
ISIS 3> cubenorm from=N1511700120_1.lev1.cub to=N1511700120_1.norm.cub
ISIS 3> cubenorm from=W1567133629_1.fill.cub to=W1567133629_1.norm.cub
ISIS 3> spiceinit from=N1511700120_1.norm.cub
ISIS 3> spiceinit from=W1567133629_1.norm.cub
ISIS 3> cam2map from=N1511700120_1.norm.cub to=N1511700120_1.map.cub
ISIS 3> cam2map from=W1567133629_1.norm.cub map=N1511700120_1.map.cub \
ISIS 3>           to=W1567133629_1.map.cub matchmap=true
ISIS 3> stereo N1511700120_1.map.equ.cub W1567133629_1.map.equ.cub result/rhea
```



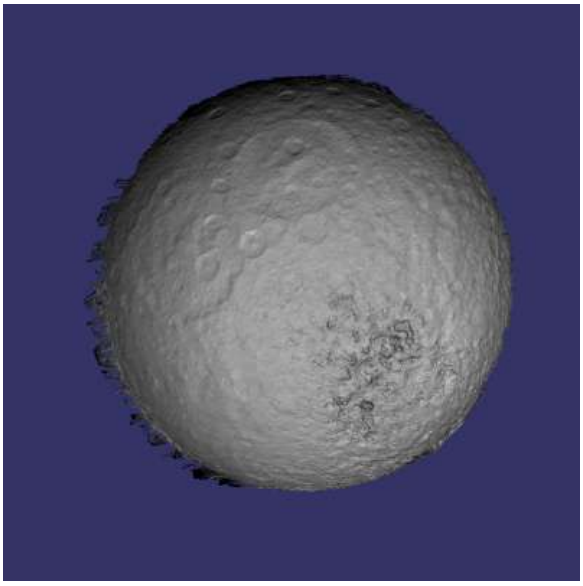
(a) `Original Left Image`



(b) `Original Right Image`



(c) `Map-Projected Left`



(d) `3D Rendering`

Figure 11.8: Example output of what is possible with Cassini's ISS NAC

stereo.default

stereo.default for Cassini ISS

```
### PREPROCESSING
alignment-method none
force-use-entire-range
individually-normalize

### CORRELATION
prefilter-mode 2
prefilter-kernel-width 1.5

cost-mode 2

corr-kernel 25 25
corr-search -55 -2 -5 10

subpixel-mode 3
subpixel-kernel 21 21

### FILTERING
rm-half-kernel 5 5
rm-min-matches 60 # Units = percent
rm-threshold 3
rm-cleanup-passes 1
```

11.11 Digital Globe Imagery

Processing of Digital Globe images is described extensively in the tutorial in chapter 4.

11.12 RPC Imagery, including GeoEye, Astrium, Cartosat-1, and PeruSat-1

Some vendors, such as GeoEye with its Ikonos and two GeoEye satellites, and Astrium, with its SPOT and Pleiades satellites, the Indian Cartosat-1 satellite provide only Rational Polynomial Camera (RPC) models. Digital Globe provides both exact linescan camera models and their RPC approximations and ASP supports both. Apparently such is the case as well for PeruSat-1, but ASP supports only the RPC model for this satellite.

RPC represents four 20-element polynomials that map geodetic coordinates (longitude-latitude-height above datum) to image pixels. Since they are easy to implement and fast to evaluate, RPC represents a universal camera model providing a simple approximation to complex exact camera models that are unique to each vendor. The only downside is that it has less precision in our opinion compared to the exact camera models.

In addition to supporting vendor-provided RPC models, ASP provides a tool named `cam2rpc` (section A.13), that can be used to create RPC camera models from ISIS and all other cameras that ASP understands, including for non-Earth planets (currently only the Earth, Moon and Mars are supported). In such situations, the planet datum must be passed to the tools reading the RPC models, as shown below.

Our RPC read driver is GDAL. If the command `gdalinfo` can identify the RPC information inside the headers of your image files (whether that information is actually embedded in the images, or stored separately in some auxiliary files with a convention GDAL understands), ASP will likely be able to see it as

well. This means that sometimes we can get away with only providing a left and right image, with no extra files containing camera information. This is specifically the case for GeoEye, and Cartosat-1. Otherwise, the camera files must be specified separately in XML files, as done for Digital Globe images (section 4.1) and PeruSat-1.

For a first test, you can download an example stereo pair from GeoEye's website at [41]. When we accessed the site, we downloaded a GeoEye-1 image of Hobart, Australia. As previously stated in the Digital Globe section, these types of images are not ideal for ASP. This is both a forest and a urban area which makes correlation difficult. ASP was designed more for modeling bare rock and ice. Any results we produce in other environments is a bonus but is not our objective.

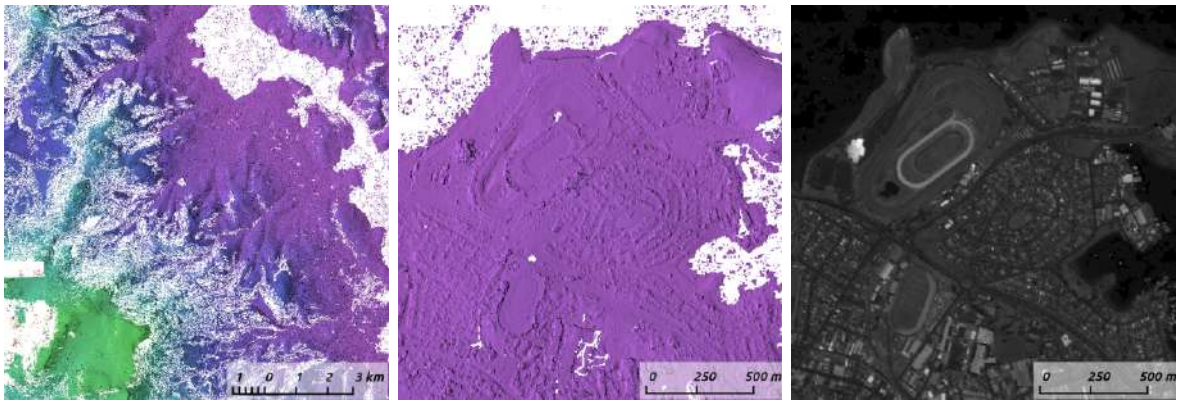


Figure 11.9: Example colorized height map and ortho image output.

Command

```
stereo -t rpc po_312012_pan_0000000.tif po_312012_pan_0010000.tif geoeye/geoeye
```

(For Cartosat data sometimes one should overwrite the *RPC.TXT files that are present with the ones that end in RPC_ORG.TXT.)

If RPC cameras are specified separately, the `stereo` command looks as follows. This example is for Mars, with the RPC models created with `cam2rpc` from ISIS cubes. So the datum has to be set.

```
stereo -t rpc --datum D_MARS left.tif right.tif left.xml right.xml run/run
```

For terrains having steep slopes, we recommend that images be map-projected onto an existing DEM before running stereo. This is described in section 5.1.7. As above, if the cameras are specified separately (as xml files), they should be on the command line, otherwise they can be omitted.

If the RPC coefficients are not stored in the original Tif images, but rather in associated .RPB or _RPC.TXT files, `mapproject` creates these files automatically for each map-projected image.

stereo.default

The stereo.default example file (appendix B) works generally well with all GeoEye pairs. Just set `alignment-method` to `affineepipolar` or `homography`.

11.13 SPOT5 Imagery

SPOT5 is a CNES (Space Agency of France) satellite launched on May 2002 and decommissioned in March 2015. SPOT5 contained two High Resolution Stereoscopic (HRS) instruments with a ground resolution of 5 meters. These two cameras were pointed forwards and backwards, allowing capture of a stereo image pair in a single pass of the satellite.

ASP supports only images from the HRS sensors on SPOT5. These images come in two parts, the data file (extension `.bil` or `.tif`) and the header file the data file (extension `.dim`). The data file can be either a plain binary file with no header information or a GeoTIFF file. The header file is a plain text XML file. When using SPOT5 images with ASP tools, pass in the data file as the image file and the header file as the camera model file.

All ASP tools can handle `.bil` images (and also `.bip` and `.bsq`) as long as a similarly named `.dim` file exists that can be looked up. The lookup succeeds if, for example, the `.dim` and `.bil` files differ only by extension (lower or upper case), or, as below, when an IMAGERY.BIL file has a corresponding METADATA file.

You can find a sample SPOT5 image at http://www.geo-airbusds.com/en/23-sample-imagery.

One issue to watch out for is that SPOT5 data typically comes in a standard directory structure where the image and header files always have the same name. The header (camera model) files cannot be passed into the `bundle_adjust` tool with the same file name even if they are in different folders. A simple workaround is to create symbolic links to the original header files with different names:

```
> ln -s  front/SEGMT01/METADATA.DIM front/SEGMT01/METADATA_FRONT.DIM
> ln -s  back/SEGMT01/METADATA.DIM  back/SEGMT01/METADATA_BACK.DIM
> bundle_adjust -t spot5 front/SEGMT01/IMAGERY.BIL back/SEGMT01/IMAGERY.BIL \
  front/SEGMT01/METADATA_FRONT.DIM back/SEGMT01/METADATA_BACK.DIM -o ba_run/out
> stereo -t spot5 front/SEGMT01/IMAGERY.BIL back/SEGMT01/IMAGERY.BIL  \
  front/SEGMT01/METADATA_FRONT.DIM back/SEGMT01/METADATA_BACK.DIM \
  st_run/out --bundle-adjust-prefix ba_run/out
```

You can also map project the SPOT5 images before they are passed to the `stereo` tool. In order to do so, you must first use the `add_spot_rpc` tool to generate an RPC model approximation of the SPOT5 sensor model, then use the `spot5maprpc` session type when running stereo on the map projected images.

```
> add_spot_rpc front/SEGMT01/METADATA.DIM -o front/SEGMT01/METADATA.DIM
> add_spot_rpc back/SEGMT01/METADATA.DIM  -o back/SEGMT01/METADATA.DIM
> mapproject sample_dem.tif front/SEGMT01/IMAGERY.BIL front/SEGMT01/METADATA.DIM
  front_map_proj.tif -t rpc
> mapproject sample_dem.tif back/SEGMT01/IMAGERY.BIL back/SEGMT01/METADATA.DIM
  back_map_proj.tif -t rpc
> stereo -t spot5maprpc front_map_proj.tif back_map_proj.tif  \
  front/SEGMT01/METADATA.DIM back/SEGMT01/METADATA.DIM \
  st_run/out sample_dem.tif
```

11.14 Dawn (FC) Framing Camera

This is a NASA mission to visit two of the largest objects in the asteroid belt, Vesta and Ceres. The framing camera on board Dawn is quite small and packs only a resolution of 1024x1024 pixels. This means

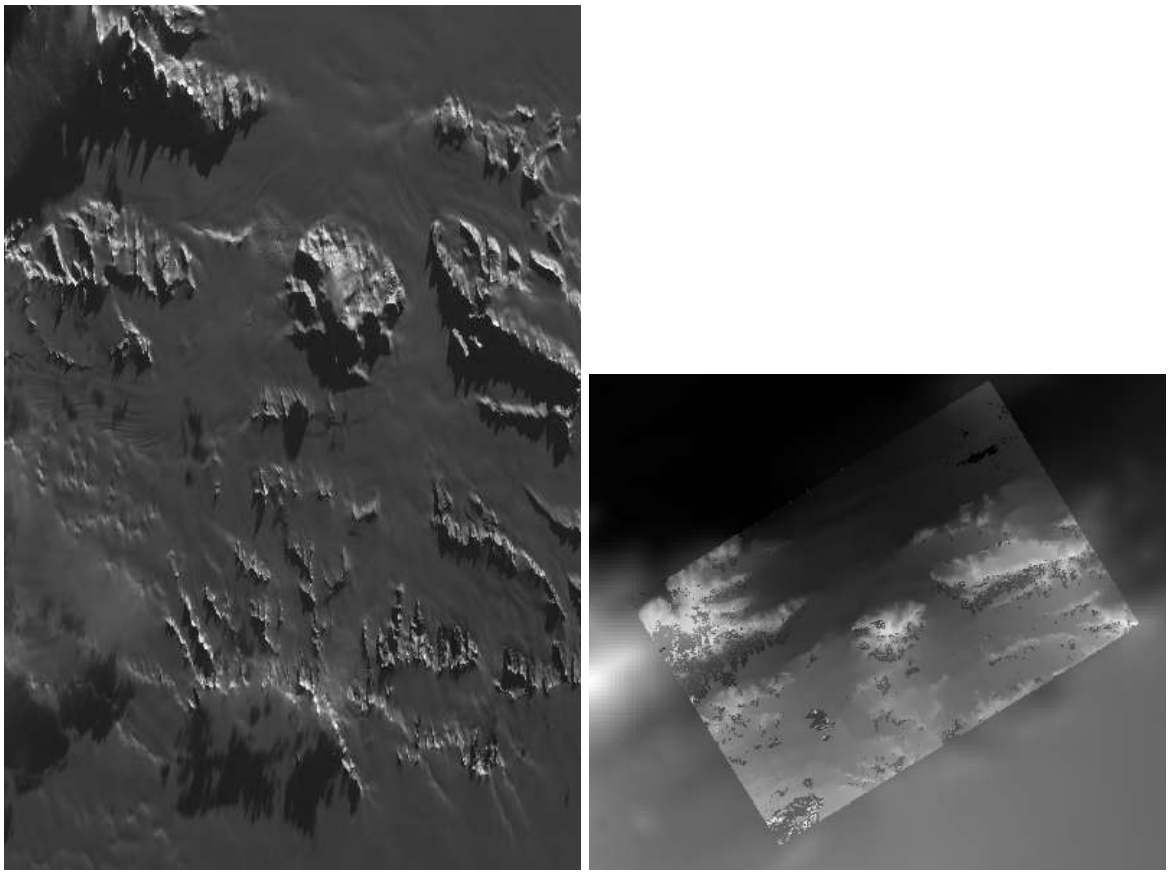


Figure 11.10: Cropped region of SPOT5 image and a portion of the associated stereo DEM overlaid on a low resolution Bedmap2 DEM.

processing time is extremely short. To its benefit, it seems that the mission planners leave the framing camera on taking shots quite rapidly. On a single pass, they seem to usually take a chain of FC images that have a high overlap percentage. This opens the idea of using ASP to process not only the sequential pairs, but also the wider baseline shots. Then someone could potentially average all the DEMs together to create a more robust data product.

For this example, we downloaded the images

`FC21A0010191_11286212239F1T.IMG` and `FC21A0010192_11286212639F1T.IMG`

which show the Cornelia crater. We found these images by looking at the popular anaglyph shown on the Planetary Science Blog [84].

Commands

First you must download the Dawn FC images from PDS.

```
ISIS3 > dawnfc2isis from=FC21A0010191_11286212239F1T.IMG \
                    to=FC21A0010191_11286212239F1T.cub
ISIS3 > dawnfc2isis from=FC21A0010192_11286212639F1T.IMG \
                    to=FC21A0010192_11286212639F1T.cub
```

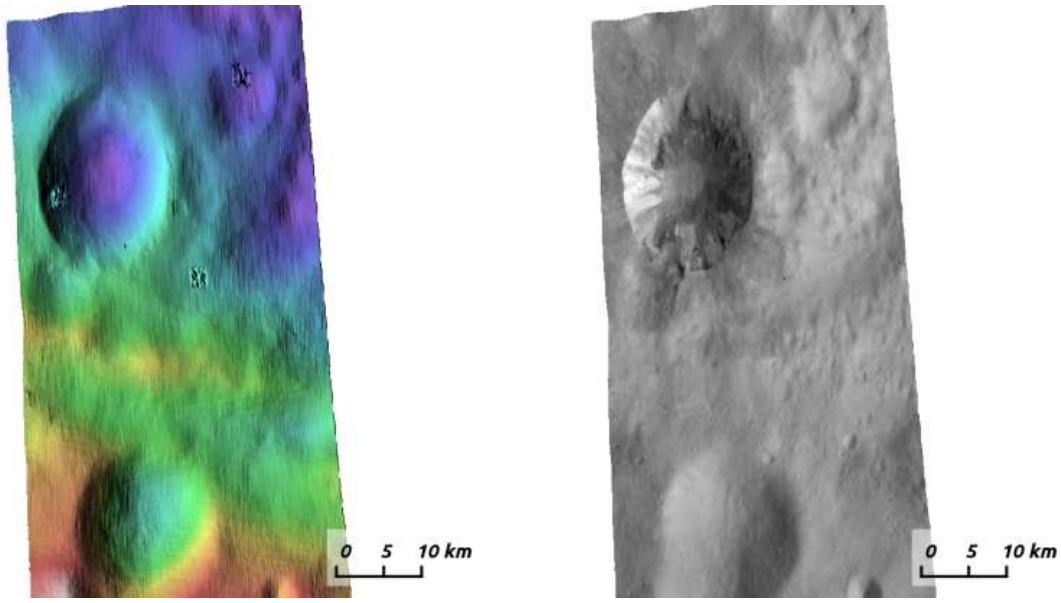


Figure 11.11: Example colorized height map and ortho image output.

```
 ISIS3 > spiceinit from=FC21A0010191_11286212239F1T.cub
 ISIS3 > spiceinit from=FC21A0010192_11286212639F1T.cub
 ISIS3 > stereo FC21A0010191_11286212239F1T.cub \
               FC21A0010192_11286212639F1T.cub stereo/stereo
 ISIS3 > point2dem stereo-PC.tif --orthoimage stereo-L.tif \
--t_srs "+proj=eqc +lat_ts=-11.5 +a=280000 +b=229000 +units=m"
```

stereo.default

The stereo.default example file (appendix B) works well for this stereo pair. Just set `alignment-method` to `affineepipolar` or `homography`.

11.15 ASTER Imagery

In this example we will describe how to process ASTER Level 1A VNIR imagery. The data can be obtained for free from https://search.earthdata.nasa.gov/search. Select a region on the map, search for AST_L1A, and choose "ASTER L1A Reconstructed Unprocessed Instrument Data V003". (The same interface can be used to obtain pre-existing ASTER DEMs.)

There are two important things to keep in mind when ordering the data. First, at the very last step, when finalizing the order options, choose GeoTIFF as the data format, rather than HDF-EOS. This way the imagery and metadata will come already extracted from the HDF file.

Second, note that ASP cannot process ASTER Level 1B imagery, as those images lack camera information.

Below, we will use the dataset `AST_L1A_00307182000191236_20160404141337_21031` near San Luis Reservoir in Northern California. This dataset will come as a directory containing TIFF imagery and meta-information as text files. We use the tool `aster2asp` (section A.33) to parse it (also there is described the data contained in this directory):

```
aster2asp 030353697511879 -o out
```

This command will create 4 files, named

```
out-Band3N.tif out-Band3B.tif out-Band3N.xml out-Band3B.xml
```

We refer again to the tool's documentation page regarding details of how these files were created.

Next, we run stereo. We can use either the exact camera model (`-t aster`), or its RPC approximation (`-t rpc`). The former is much slower but more accurate.

```
stereo -t aster --subpixel-mode 3 out-Band3N.tif out-Band3B.tif \
   out-Band3N.xml out-Band3B.xml out_stereo/run
```

or

```
stereo -t rpc --subpixel-mode 3 out-Band3N.tif out-Band3B.tif \
   out-Band3N.xml out-Band3B.xml out_stereo/run
```

This is followed by DEM creation:

```
point2dem -r earth --tr 0.000277777777778 out_stereo/run-PC.tif
```

The value 0.000277777777778 is the desired output DEM resolution, specified in degrees. It is approximately 31 meters/pixel, the same as the publicly available ASTER DEM, and about twice the 15 meters/pixel image resolution.

Much higher quality results, but still not as detailed as the public ASTER DEM can be obtained by doing stereo as before, followed by map-projection onto a coarser and smoother version of the obtained DEM, and then redoing stereo with map-projected images (per the suggestions in chapter 6). Using `--subpixel-mode 2`, while much slower, yields the best results. The flow is as follows:

```
# Initial stereo
stereo -t aster --subpixel-mode 3 out-Band3N.tif out-Band3B.tif \
   out-Band3N.xml out-Band3B.xml out_stereo/run

# Create a coarse and smooth DEM at 300 meters/pixel
point2dem -r earth --tr 0.0026949458523585 out_stereo/run-PC.tif \
  -o out_stereo/run-300m

# Map-project onto this DEM at 10 meters/pixel
mapproject --tr 0.0000898315284119 out_stereo/run-300m-DEM.tif \
  out-Band3N.tif out-Band3N.xml out-Band3N_proj.tif
mapproject --tr 0.0000898315284119 out_stereo/run-300m-DEM.tif \
  out-Band3B.tif out-Band3B.xml out-Band3B_proj.tif

# Run stereo with the map-projected images with subpixel-mode 2
stereo -t aster --subpixel-mode 2 out-Band3N_proj.tif out-Band3B_proj.tif \
  out-Band3N.xml out-Band3B.xml out_stereo_proj/run              \
  out_stereo/run-300m-DEM.tif

# Create the final DEM
point2dem -r earth --tr 0.000277777777778 out_stereo_proj/run-PC.tif
```

Here we could have again used `-t rpc` instead of `-t aster`. The map-projection was done using `--tr 0.0000898315284119` which is about 10 meters/pixel.

It is possible to increase the resolution of the final DEM slightly by instead map-projecting at 7 meters/pixel, hence using

```
--tr .0000628820698883
```

or smaller correlation and subpixel-refinement kernels, that is

```
--corr-kernel 15 15 --subpixel-kernel 25 25
```

instead of the defaults (21 21 and 35 35) but this comes with increased noise as well, and using a finer resolution results in longer run-time.

We also tried to first bundle-adjust the cameras, using ASP's `bundle_adjust`. We did not notice a noticeable improvement in results.

11.16 SkySat Imagery

In this section we will discuss how to process the SkySat "Video" product.

It is very important to note that this is a very capricious dataset, so some patience will be needed to work with it. That is due to the following factors:

- The baseline can be small, so the perspective of the left and right image can be too similar.
- The footprint on the ground is small, on the order of 2 km.
- The terrain can be very steep.
- The known longitude-latitude corners of each image have only a few digits of precision, which can result in poor initial estimated cameras.

Below a recipe for how to deal with this data is described, together with things to watch for and advice when things don't work.

11.16.1 The input data

We will use as an illustration a mountainous terrain close to Breckenridge, Colorado. The dataset we fetched is called `s4_20181107T175036Z_video.zip`. We chose to work with the following four images from it:

```
1225648254.44006968_sc00004_c1_PAN.tiff
1225648269.40892076_sc00004_c1_PAN.tiff
1225648284.37777185_sc00004_c1_PAN.tiff
1225648299.37995577_sc00004_c1_PAN.tiff
```

A sample picture from this image set is shown in figure 11.12.

It is very important to pick images that have sufficient difference in perspective, but which are still reasonably similar, as otherwise the procedure outlined in this section will fail.

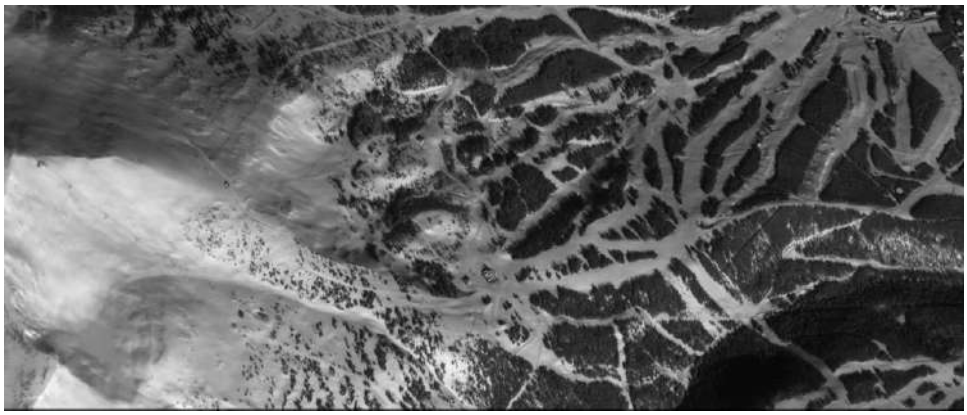


Figure 11.12: An image used in the SkySat example. Reproduced with permission.

11.16.2 Initial camera models and a reference DEM

Based on vendor's documentation, these images are 2560×1080 pixels. We use the geometric center of the image as the optical center, which turned out to be a reasonable enough assumption (verified by allowing it to float later). Since the focal length is given as 3.6 m and the pixel pitch is 6.5×10^{-6} m, the focal length in pixels is

$$3.6/6.5 \times 10^{-6} = 553846.153846.$$

We will fetch an SRTM DEM of the area, which will be used as a reference for registration, from location:

```
https://e4ftl01.cr.usgs.gov/provisional/MEaSUREs/NASADEM/NorthAmerica/hgt_merge/n39w107.hgt.zip
```

After unzipping it, we clip it to the area of interest:

```
gdal_translate -projwin -106.1679167 39.5120833 -106.0034722 39.3895833 \
  n39w107.hgt ref_dem_clipped.tif
```

It is good to be a bit generous with clipping, so that the output DEM goes a few km or more beyond the region of interest. If the region of interest is not fully covered by an SRTM tile, a neighboring one can be downloaded as well. They can be merged with `dem_mosaic` and then cropped as before.

It appears that SRTM stores heights above the geoid, rather than above the datum. Hence it needs to be adjusted, as follows:

```
dem_geoid --reverse-adjustment ref_dem_clipped.tif -o run/run
mv run/run-adj.tif ref_dem.tif
```

This may adjust the DEM by up to 100 meters.

Using the tool `cam_gen` (section A.41) bundled with ASP, we create an initial camera model and a GCP file (section A.4.1) for the first image as as follows:

```
cam_gen output/video/frames/1225648254.44006968_sc00004_c1_PAN.tiff   \
  --reference-dem ref_dem.tif --focal-length 553846.153846            \
  --optical-center 1280 540 --pixel-pitch 1 --height-above-datum 4000 \
  --refine-camera --frame-index output/video/frame_index.csv          \
  --gcp-std 1 -o v1.tsai --gcp-file v1.gcp
```

This tool works by reading the longitude and latitude of each image corner on the ground from the file `frame_index.csv`, and finding the position and orientation of the camera that best fits this data. The camera is written to `v1.tsai`. A GCP file is written to `v1.gcp`. This will help later with bundle adjustment.

In this command, the optical center and focal length are as mentioned earlier. The reference SRTM DEM is used to infer the height above datum for each image corner based on its longitude and latitude. The height value specified via `--height-above-datum` is used as a fallback option, if for example, the DEM is incomplete, and is not strictly necessary for this example. This tool also accepts the longitude and latitude of the corners as an option, via `--lon-lat-values`.

The flag `--refine-camera` makes `cam_gen` solve a least square problem to refine the output camera. In some rare cases it can get the refinement wrong, though by and large it it greatly improves the cameras.

For simplicity of notation, we will create a symbolic link from this image to the shorter name `v1.tif`, and the GCP file needs to be edited to reflect this. The same will apply to the other files. We will have then four images, `v1.tif, v2.tif, v3.tif, v4.tif`, and corresponding camera and GCP files.

A good sanity check is to visualize these computed cameras in ASP's `orbitviz` tool. It can be invoked as:

```
orbitviz v[1-4].tif v[1-4].tsai -o orbit.kml
```

The output KML file can then be opened in Google Earth. We very strongly recommend this step, since it may catch inaccurate cameras which will cause problems later.

Another important check is to map-project these images using the cameras and overlay them in `stereo_gui` on top of the reference DEM. Here is an example for the first image:

```
mapproject --t_srs \
'+proj=stere +lat_0=39.4702 +lon_0=253.908 +k=1 +x_0=0 +y_0=0 +datum=WGS84 +units=m' \
ref_dem.tif v1.tif v1.tsai v1_map.tif
```

Notice that we used above a longitude and latitude around the area of interest. This will need to be modified for your specific example.

11.16.3 Bundle adjustment

At this stage, the cameras should be about right, but not quite exact. We will take care of this using bundle adjustment. We will invoke this tool twice. In the first call we will make the cameras self-consistent, which can make them move away, however, and in the second call we will bring them back to the original location.

```
parallel_bundle_adjust -t nadirpinhole --disable-tri-ip-filter \
  --disable-pinhole-gcp-init --skip-rough-homography           \
  --force-reuse-match-files --ip-inlier-factor 2.0             \
  --ip-uniqueness-threshold 0.9 --ip-per-tile 2000             \
  --datum WGS84 --inline-adjustments --camera-weight 0         \
  --overlap-limit 10 --robust-threshold 10                     \
  --remove-outliers-params '75 3 4 5'                          \
  --ip-num-ransac-iterations 1000                              \
  --num-passes 2 --num-iterations 2000                         \
  v[1-4].tif v[1-4].tsai -o ba/run

parallel_bundle_adjust -t nadirpinhole --datum WGS84           \
```

```
  --force-reuse-match-files --inline-adjustments               \
  --num-passes 1 --num-iterations 0                            \
  --transform-cameras-using-gcp                                \
  v[1-4].tif ba/run-v[1-4].tsai v[1-4].gcp -o ba/run
```

It is very important to not use the "pinhole" session here, rather "nadirpinhole" as the former does not filter well interest points in this steep terrain.

The output optimized cameras will be named `ba/run-run-v[1-4].tsai`. The reason one has the word "run" repeated is because we ran this tool twice. The intermediate cameras from the first run were called `ba/run-v[1-4].tsai`.

Here we use `--ip-per-tile 2000` to create a lot of interest points. This will help with alignment later. It is suggested that the user study all these options and understand what they do. We also used `--robust-threshold 10` to force the solver to work the bigger errors. That is necessary since the initial cameras could be pretty inaccurate.

It is very important to examine the residual file named

```
ba/run-final_residuals_no_loss_function_pointmap_point_log.csv
```

Here, the third column are the heights of triangulated interest points, while the fourth column are the reprojection errors. Normally these errors should be a fraction of a pixel, as otherwise the solution did not converge. The last entries in this file correspond to the GCP, and those should be looked at carefully as well. The reprojection errors for GCP should be on the order of tens of pixels because the longitude and latitude of each GCP are not well-known.

It is also very important to examine the obtained match files in the output directory. If there are too few matches, particularly among very similar images, one may need to increase the value of `--epipolar-threshold` (or of `--ip-inlier-factor` for the not-recommended pinhole session). Note that a large value here may allow more outliers.

Another thing one should keep an eye on is the height above datum of the camera centers as printed by bundle adjustment towards the end. Any large difference in camera heights (say more than a few km) could be a symptom of some failure.

11.16.4 Creating terrain models

The next step is to run stereo and create DEMs.

We will run the following command for each pair of images. Note that we reuse the filtered match points created by bundle adjustment.

```
i=1
((j=i+1))
st=stereo_v${i}${j}
rm -rfv $st
mkdir -p $st
cp -fv ba/run-v${i}__v${j}-clean.match $st/run-v${i}__v${j}.match
parallel_stereo --skip-rough-homography -t nadirpinhole --stereo-algorithm 2 \
  v${i}.tif v${j}.tif ba/run-run-v${i}.tsai ba/run-run-v${j}.tsai $st/run
point2dem --stereographic --proj-lon 253.90793 --proj-lat 39.47021 --tr 4  \
  --errorimage $st/run-PC.tif
```

(Repeat this for other values of i.)

Here we chose to use a stereographic projection in `point2dem` centered on this region to create the DEM in units of meter. One can can also use a different projection that can be passed to the option `--t_srs`, or if doing as above, the center of the projection would need to change if working on a different region.

It is important to examine the mean intersection error for each DEM:

```
gdalinfo -stats stereo_v12/run-IntersectionErr.tif | grep Mean
```

which should hopefully be no more than 0.5 meters, otherwise likely bundle adjustment failed. One should also compare the DEMs among themselves:

```
geodiff --absolute stereo_v12/run-DEM.tif stereo_v23/run-DEM.tif -o tmp
gdalinfo -stats tmp-diff.tif | grep Mean
```

(And so on for any other pair.) Here the mean error should be on the order of 2 meters, or hopefully less.

11.16.5 Mosaicking and alignment

If more than one image pair was used, the obtained DEMs can be mosaicked:

```
dem_mosaic stereo_v12/run-DEM.tif stereo_v23/run-DEM.tif \
  stereo_v34/run-DEM.tif -o mosaic.tif
```

This DEM can be hillshaded and overlayed on top of the reference DEM.

The next step is aligning it to the reference.

```
pc_align --max-displacement 1000 --save-transformed-source-points \
  --alignment-method similarity-point-to-point                    \
  ref_dem.tif mosaic.tif -o align/run
```

It is important to look at the errors printed by this tool before and after alignment, as well as details about the alignment that was applied. The obtained aligned cloud can be made into a DEM again:

```
point2dem --stereographic --proj-lon 253.90793 --proj-lat 39.47021 --tr 4  \
  align/run-trans_source.tif
```

The absolute difference before and after alignment can be found as follows:

```
geodiff --absolute mosaic.tif ref_dem.tif -o tmp
gdalinfo -stats tmp-diff.tif | grep Mean

geodiff --absolute  align/run-trans_source-DEM.tif ref_dem.tif -o tmp
gdalinfo -stats tmp-diff.tif | grep Mean
```

In this case the mean error after alignment was about 6.5 m, which is not too bad given that the reference DEM resolution is about 30 m/pixel.

11.16.6 Alignment of cameras

The transform computed with `pc_align` can be used to bring the cameras in alignment to the reference DEM. That can be done as follows:

```
parallel_bundle_adjust -t nadirpinhole --datum wgs84     \
  --force-reuse-match-files                              \
  --inline-adjustments --num-passes 1 --num-iterations 0 \
  --initial-transform align/run-transform.txt            \
  v[1-4].tif ba/run-run-v[1-4].tsai -o ba/run
```

creating the aligned cameras `ba/run-run-run-v[1-4].tsai`. If `pc_align` was called with the reference DEM being the second cloud, one should use above the file

```
align/run-inverse-transform.txt
```

as the initial transform.

11.16.7 Mapprojection

If the steep topography prevents good DEMs from being created, one can map-project the images first onto the reference DEM:

```
for i in 1 2 3 4; do
  mapproject ref_dem.tif v${i}.tif ba/run-run-run-v${i}.tsai v${i}_map.tif
done
```

and then run stereo with the mapprojected images, such as:

```
i=1
((j=i+1))
rm -rfv stereo_map_v${i}${j}
stereo v${i}_map.tif v${j}_map.tif                                    \
  ba/run-run-run-v${i}.tsai ba/run-run-run-v${j}.tsai                 \
  stereo_map_v${i}${j}/run ref_dem.tif --session-type pinhole         \
  --cost-mode 4 --stereo-algorithm 2 --corr-seed-mode 1               \
  --alignment-method none --corr-tile-size 9000
point2dem --stereographic --proj-lon 253.90793 --proj-lat 39.47021 --tr 4  \
  --errorimage stereo_map_v${i}${j}/run-PC.tif
```

It is important to note that here we used the cameras that were aligned with the reference DEM. We could have as well mapprojected onto a lower-resolution version of the mosaicked and aligned DEM with its holes filled.

11.16.8 When things fail

Processing SkySat images is difficult, for various reasons mentioned earlier. A few suggestions were also offered along the way when things go wrong.

Problems are usually due to cameras being initialized inaccurately by `cam_gen` or bundle adjustment not optimizing them well. The simplest solution is often to just try a different pair of images from the sequence, say from earlier or later in the flight, or a pair with less overlap, or with more time elapsed between the two acquisitions. Modifying various parameters may help as well.

We have experimented sufficiently with various SkySat datasets to be sure that the intrinsics (focal length, optical center, and pixel pitch) are usually not the issue, rather the positions and orientations of the cameras.

11.16.9 Structure from motion

In case `cam_gen` does not create sufficiently good cameras, one can attempt to use the `camera_solve` tool (chapter 9). This will create hopefully good cameras but in an arbitrary coordinate system. Then we will transfer those to the world coordinates using GCP.

Here is an example for two cameras:

```
out=out_v12
ba_params="--num-passes 1 --num-iterations 0 --transform-cameras-using-gcp"
theia_overdides="--sift_num_levels=6 --lowes_ratio=0.9
  --min_num_inliers_for_valid_match=10
  --min_num_absolute_pose_inliers=10
  --bundle_adjustment_robust_loss_function=CAUCHY
  --post_rotation_filtering_degrees=180.0 --v=2
  --max_sampson_error_for_verified_match=100.0
  --max_reprojection_error_pixels=100.0
  --triangulation_reprojection_error_pixels=100.0
  --min_num_inliers_for_valid_match=10
  --min_num_absolute_pose_inliers=10"
rm -rfv $out
camera_solve $out --datum WGS84 --calib-file v1.tsai                \
    --bundle-adjust-params "$ba_params v1.gcp v2.gcp" v1.tif v2.tif
```

The obtained cameras should be bundle-adjusted as done for the outputs of `cam_gen`. Note that this tool is capricious and its outputs can be often wrong. In the future it will be replaced by something more robust.

11.16.10 RPC models

Some SkySat datasets come with RPC camera models, typically embedded in the images. This can be verified by running

```
gdalinfo -stats output/video/frames/1225648254.44006968_sc00004_c1_PAN.tiff
```

We found that these models are not sufficiently robust for stereo. But they can be used to create initial guess cameras with `cam_gen` instead of using longitude and latitude of corners. Here is an example:

```
img=output/video/frames/1225648254.44006968_sc00004_c1_PAN.tiff
cam_gen $img --reference-dem ref_dem.tif --focal-length 553846.153846  \
   --optical-center 1280 540 --pixel-pitch 1 --height-above-datum 4000 \
   --refine-camera --gcp-std 1 --input-camera $img                     \
   -o v1_rpc.tsai --gcp-file v1_rpc.gcp
```

(Note that the Breckenridge dataset does not have RPC data, but other datasets do.)

Then one can proceed as earlier (particularly the GCP file can be edited to reflect the shorter image name).

One can also regenerate the provided SkySat RPC model as:

```
cam2rpc -t rpc --dem-file dem.tif input.tif output.xml
```

Here, the reference DEM should go beyond the extent of the image. This tool makes it possible to decide how finely to sample the DEM, and one can simply use longitude-latitude and height ranges instead of the DEM.

We assumed in the last command that the input image implicitly stores the RPC camera model, as is the case for SkySat.

Also, any pinhole camera models obtained using our software can be converted to RPC models as follows:

```
cam2rpc --dem-file dem.tif input.tif input.tsai output.xml
```

11.16.11 Bundle adjustment using reference terrain

At this stage, if desired, but this is rather unnecessary, one can do joint optimization of the cameras using dense and uniformly distributed interest points, and using the reference DEM as a constraint. This should make the DEMs more consistent among themselves and closer to the reference DEM.

It is also possible to float the intrinsics, per section 8.2.1, which sometimes can improve the results further.

For that, one should repeat the `stereo_tri` part of of the stereo commands from section 11.16.4 with the flags `--num-matches-from-disp-triplets 10000` and `--unalign-disparity` to obtain dense interest points and unaligned disparity.

The match points can be examined as:

```
stereo_gui v1.tif v2.tif stereo_v12/run-disp-v1__v2.match
```

and the same for the other image pairs. Hopefully they will fill as much of the images as possible. One should also study the unaligned disparities, for example

```
stereo_v12/run-v1__v2-unaligned-D.tif
```

by invoking `disparitydebug` on it and then visualizing the two obtained images. Hopefully these disparities are dense and with few holes.

The dense interest points should be copied to the new bundle adjustment directory, such as

```
mkdir -p ba_ref_terrain
cp stereo_v12/run-disp-v1__v2.match ba_ref_terrain/run-v1__v2.match
```

and the same for the other ones (note the convention for match files in the new directory). The unaligned disparities can be used from where they are.

Then bundle adjustment using the reference terrain constraint proceeds as follows:

```
disp_list=$(ls stereo_v[1-4][1-4]/*-unaligned-D.tif)
bundle_adjust v[1-4].tif  ba/run-run-run-v[1-4].tsai -o ba_ref_terrain/run    \
--reference-terrain ref_dem.tif --disparity-list "$disp_list"                \
--max-num-reference-points 10000000 --reference-terrain-weight 50            \
--parameter-tolerance 1e-12 -t nadirpinhole --max-iterations 500             \
--overlap-limit 1 --inline-adjustments --robust-threshold 2                  \
--force-reuse-match-files --max-disp-error 100 --camera-weight 0
```

If invoking this creates new match files, it means that the dense match files were not copied successfully to the new location. If this optimization is slow, perhaps too many reference terrain points were picked.

This will create, as before, the residual file named

```
ba_ref_terrain/run-final_residuals_no_loss_function_pointmap_point_log.csv
```

showing how consistent are the cameras among themselves, and in addition, a file named

```
ba_ref_terrain/run-final_residuals_no_loss_function_reference_terrain.txt
```

which tells how well the cameras are aligned to the reference terrain. The errors in the first file should be under 1 pixel, and in the second one should be mostly under 2-3 pixels (both are the fourth column in these files).

The value of `--reference-terrain-weight` can be increased to make the alignment to the reference terrain a little tighter.

It is hoped that after running stereo with these refined cameras, the obtained DEMs will differ by less than 2 m among themselves, and by less than 4 m as compared to the reference DEM.

11.16.12 Floating the camera intrinsics

If desired to float the focal length as part of the optimization, one should pass in addition, the options

```
--solve-intrinsics --intrinsics-to-float 'focal_length'
```

Floating the optical center can be done by adding it in as well.

It is important to note that for SkySat the intrinsics seem to be already quite good, and floating them is not necessary and is only shown for completeness. If one wants to float them, one should vary the focal length while keeping the optical center fixed, and vice versa, and compare the results. Then, with the result that shows most promise, one should vary the other parameter. If optimizing the intrinsics too aggressively, it is not clear if they will still deliver better results with other images or if comparing with a different reference terrain.

Yet, if desired, one can float even the distortion parameters. For that, the input camera files need to be converted to some camera model having these (see section D.1), and their values can be set to something very small. One can use the Brown-Conrady model, for example, so each camera file must have instead of `NULL` at the end the fields:

```
BrownConrady
xp  = -1e-12
yp  = -1e-12
k1  = -1e-10
k2  = -1e-14
k3  = -1e-22
p1  = -1e-12
p2  = -1e-12
phi = -1e-12
```

There is always a chance when solving these parameters that the obtained solution is not optimal. Hence, one can also try using as initial guesses different values, for example, by negating the above numbers.

One can also try to experiment with the option `--heights-from-dem`, and also with `--robust-threshold` if it appears that the large errors are not minimized enough.

11.17 Declassified satellite images: KH-4B

ASP supports the declassified high-resolution CORONA KH-4B images. These images can be processed using either optical bar (panoramic) camera models or as pinhole camera models with RPC distortion. Most of the steps are similar to the example in section 11.12. The optical bar camera model is based on [132] and [139], whose format is described in section D.3.

11.17.1 Fetching the data

KH-4B images are available via the USGS Earth Explorer, at

https://earthexplorer.usgs.gov/

(an account is required to download the data). We will work with the KH-4B image pair

```
DS1105-2248DF076
DS1105-2248DA082
```

To get these from Earth Explorer, click on the `Data Sets` tab and select the three types of declassified data available, then in the `Additional Criteria` tab choose `Declass 1`, and in the `Entity ID` field in that tab paste the above frames (if no results are returned, one can attempt switching above to `Declass 2`, etc). Clicking on the `Results` tab presents the user with information about these frames.

Clicking on `Show Metadata and Browse` for every image will pop-up a table with meta-information. That one can be pasted into a text file, named for example, `DS1105-2248DF076.txt` for the first image, from which later the longitude and latitude of each image corner will be parsed. Then one can click on `Download Options` to download the data.

11.17.2 Stitching the images

Each downloaded image will be made up of 2-4 portions, presumably due to the limitations of the scanning equipment. They can be stitched together using ASP's `image_mosaic` tool (section A.9).

For some reason, the KH-4B images are scanned in an unusual order. To mosaic them, the last image must be placed first, the next to last should be second, etc. In addition, as seen from the tables of metadata discussed earlier, some images correspond to the `Aft` camera type. Those should be rotated 180 degrees after mosaicking, hence below we use the `--rotate` flag for that one. The overlap width is manually determined by looking at two of the sub images in `stereo_gui`.

With this in mind, image mosaicking for these two images will happen as follows:

```
image_mosaic DS1105-2248DF076_d.tif  DS1105-2248DF076_c.tif              \
  DS1105-2248DF076_b.tif  DS1105-2248DF076_a.tif -o DS1105-2248DF076.tif \
  --ot byte --overlap-width 7000 --blend-radius 2000
image_mosaic DS1105-2248DA082_d.tif DS1105-2248DA082_c.tif              \
  DS1105-2248DA082_b.tif  DS1105-2248DA082_a.tif -o DS1105-2248DA082.tif \
  --ot byte --overlap-width 7000 --blend-radius 2000 --rotate
```

In order to process with the optical bar camera model these images need to be cropped to remove the most of empty area around the image. The four corners of the valid image area can be manually found by clicking on the corners in `stereo_gui`. Note that for some input images it can be unclear where the proper location for the corner is due to edge artifacts in the film. Do your best to select the image corners such that obvious artifacts are kept out and all reasonable image sections are kept in. ASP provides a simple Python tool called `historical_helper.py` to rotate the image so that the top edge is horizontal while also cropping the boundaries. Pass in the corner coordinates as shown below in the order top-left, top-right, bot-right, bot-left (column then row). This is also a good opportunity to simplify the file names going forwards.

```
historical_helper.py rotate-crop --input-path DS1105-2248DA082.tif  --output-path aft.tif \
  --interest-points '4523 1506  114956 1450  114956 9355  4453 9408'
historical_helper.py rotate-crop --input-path DS1105-2248DF076.tif  --output-path for.tif \
  --interest-points '6335 1093  115555 1315  115536 9205  6265 8992'
```

11.17.3 Fetching a ground truth DEM

To create initial cameras to use with these images, and to later refine and validate the terrain model made from them, we will need a ground truth source. Several good sets of DEMs exist, including SRTM, ASTER, and TanDEM-X. Here we will work with SRTM, which provides DEMs with a 30-meter post spacing. The bounds of the region of interest are inferred from the tables with meta-information from above. We will use `wget` to fetch

https://e4ftl01.cr.usgs.gov/provisional/MEaSUREs/NASADEM/Eurasia/hgt_merge/n31e099.hgt.zip

and also tiles `n31e100` and `n31e101`. After unzipping, these can be merged and cropped as follows:

```
dem_mosaic n*.hgt --t_projwin 99.6 31.5 102 31 -o dem.tif
```

Determining these bounds and the visualization of all images and DEMs can be done in `stereo_gui`.

The SRTM DEM may need adjustment, as discussed in section 11.16.2.

11.17.4 Creating camera files

ASP provides the tool named `cam_gen` that, based on a camera's intrinsics and the positions of the image corners on Earth's surface will create initial camera models that will be the starting point for aligning the cameras.

To create optical bar camera models, an example camera model file is needed. This needs to contain all of the expected values for the camera, though image_size, image_center, iC, and IR can be any value since they will be recalculated. The pitch is determined by the resolution of the scanner used, which is seven microns. The other values are determined by looking at available information about the satellite. For the first image (DS1105-2248DF076) the following values were used:

```
VERSION_4
OPTICAL_BAR
image_size = 13656 1033
image_center = 6828 517
pitch = 7.0e-06
f = 0.61000001430511475
scan_time = 0.5
forward_tilt = 0.2618
iC = -1030862.1946224371 5468503.8842079658 3407902.5154047827
iR = -0.95700845635275322 -0.27527006183758934 0.091439638698163225 \
     -0.26345593052063937 0.69302501329766897 -0.67104940475144637 \
     0.1213498543172795 -0.66629027007731101 -0.73575232847574434
speed = 7700
mean_earth_radius = 6371000
mean_surface_elevation = 4000
motion_compensation_factor = 1.0
scan_dir = right
```

For a description of each value, see D.3. For the other image (aft camera) the forward tilt was set to -0.2618 and scan_dir was set to 'left'. The correct values for scan_dir (left or right) and use_motion_compensation (1.0 or -1.0) are not known for certain due to uncertainties about how the images were recorded and may even change between launches of the KH-4 satellite. You will need to experiment to see which combination of settings produces the best results for your particular data set.

The metadata table from Earth Explorer has the following entries for DS1105-2248DF076:

```
NW Corner Lat dec   31.266
NW Corner Long dec  99.55
NE Corner Lat dec   31.55
NE Corner Long dec  101.866
SE Corner Lat dec   31.416
SE Corner Long dec  101.916
SW Corner Lat dec   31.133
SW Corner Long dec  99.55
```

These correspond to the upper-left, upper-right, lower-right, and lower-left pixels in the image. We will invoke `cam_gen` as follows:

```
cam_gen --sample-file sample_kh4b_for_optical_bar.tsai --camera-type opticalbar \
```

```
  --lon-lat-values '99.55 31.266 101.866 31.55 101.916 31.416 99.55 31.133' \
  for.tif --reference-dem dem.tif --refine-camera -o for.tsai

cam_gen --sample-file sample_kh4b_aft_optical_bar.tsai --camera-type opticalbar
  --lon-lat-values '99.566 31.266 101.95 31.55 101.933 31.416 99.616 31.15' \
  aft.tif --reference-dem dem.tif --refine-camera -o aft.tsai
```

It is very important to note that if, for example, the upper-left image corner is in fact the NE corner from the metadata, then that corner should be the first in the longitude-latitude list when invoking this tool.

An important sanity check is to mapproject the images with these cameras, for example as:

```
mapproject dem.tif for.tif for.tsai for.map.tif
```

and then overlay the mapprojected image on top of the DEM in `stereo_gui`. If it appears that the image was not projected correctly, likely the order of image corners was incorrect. At this stage it is not unusual that the mapprojected images are shifted from where they should be, that will be corrected later.

11.17.5 Bundle adjustment and stereo

Before processing the input images it is a good idea to experiment with reduced resolution copies in order to accelerate testing. You can easily generate reduced resolution copies of the images using `stereo_gui` as shown below. When making a copy of the camera model files, make sure to update image_size, image_center (divide by N), and the pitch (multiply by N) to account for the downsample amount.

```
stereo_gui for.tif aft.tif --create-image-pyramids-only
ln -s for_sub8.tif  for_small.tif
ln -s aft_sub8.tif  aft_small.tif
cp for.tsai  for_small.tsai
cp aft.tsai  aft_small.tsai
```

You can now run bundle adjustment on the downsampled images:

```
bundle_adjust for_small.tif aft_small.tif                \
  for_small.tsai aft_small.tsai                          \
  -o ba_small/run --max-iterations 100 --camera-weight 0  \
  --disable-tri-ip-filter --disable-pinhole-gcp-init      \
  --skip-rough-homography --inline-adjustments            \
  --ip-detect-method 1 -t opticalbar --datum WGS84
```

Followed by stereo and DEM creation:

```
parallel_stereo for_small.tif aft_small.tif                        \
  ba_small/run-for_small.tsai ba_small/run-aft_small.tsai          \
  stereo_small_mgm/run --alignment-method affineepipolar           \
  -t opticalbar --skip-rough-homography --disable-tri-ip-filter  \
  --skip-low-res-disparity-comp --ip-detect-method 1               \
  --stereo-algorithm 2

point2dem --stereographic --proj-lon 100.50792 --proj-lat 31.520417 \
  --tr 30 stereo_small_mgm/run-PC.tif
```

This will create a very rough initial DEM. It is sufficient however to align and compare with the SRTM DEM:

```
pc_align --max-displacement -1                                        \
  --initial-transform-from-hillshading similarity                     \
  --save-transformed-source-points --num-iterations 0                 \
  --max-num-source-points 1000 --max-num-reference-points 1000        \
  dem.tif stereo_small_mgm/run-DEM.tif -o stereo_small_mgm/run

point2dem --stereographic --proj-lon 100.50792 --proj-lat 31.520417 \
  --tr 30 stereo_small_mgm/run-trans_source.tif
```

This will hopefully create a DEM aligned to the underlying SRTM. There is a chance that this may fail as the two DEMs to align could be too different. In that case, one can re-run `point2dem` to re-create the DEM to align with a coarser resolution, say with `--tr 120`, then re-grid the SRTM DEM to the same resolution, which can be done as:

```
pc_align --max-displacement -1 dem.tif dem.tif -o dem/dem             \
  --num-iterations 0 --max-num-source-points 1000                     \
  --max-num-reference-points 1000 --save-transformed-source-points

point2dem --stereographic --proj-lon 100.50792 --proj-lat 31.520417   \
  --tr 120 dem/dem-trans_source.tif
```

You can then try to align the newly obtained coarser SRTM DEM to the coarser DEM from stereo.

11.17.6 Floating the intrinsics

The obtained alignment transform can be used to align the cameras as well, and then one can experiment with floating the intrinsics, as in section 11.16.

11.17.7 Modeling the camera models as pinhole cameras with RPC distortion

Once sufficiently good optical bar cameras are produced and the DEMs from them are reasonably similar to some reference terrain ground truth, such as SRTM, one may attempt to improve the accuracy further by modeling these cameras as simple pinhole models with the nonlinear effects represented as a distortion model given by Rational Polynomial Coefficients (RPC) of any desired degree (see section D.1). The best fit RPC representation can be found for both optical bar models, and the RPC can be further optimized using the reference DEM as a constraint.

To convert from optical bar models to pinhole models with RPC distortion one does

```
convert_pinhole_model for_small.tif for_small.tsai -o for_small_rpc.tsai \
  --output-type RPC --sample-spacing 50 --rpc-degree 2
```

and the same for the other camera. The obtained cameras should be bundle-adjusted as before. One can create a DEM and compare it with the one obtained with the earlier cameras. Likely some shift in the position of the DEM will be present, but hopefully not too large. The `pc_align` tool can be used to make this DEM aligned to the reference DEM.

Next, one follows the same process as outlined in sections 11.16 and 8.2.1 to refine the RPC coefficients. We will float the RPC coefficients of the left and right images independently, as they are unrelated. Hence the command we will use is:

```
bundle_adjust for_small.tif aft_small.tif for_small_rpc.tsai aft_small_rpc.tsai \
  -o ba_rpc/run --max-iterations 200 --camera-weight 0                           \
  --disable-tri-ip-filter --disable-pinhole-gcp-init                             \
  --skip-rough-homography --inline-adjustments                                   \
  --ip-detect-method 1 -t nadirpinhole --datum WGS84                             \
  --force-reuse-match-files --reference-terrain-weight 1000                      \
  --parameter-tolerance 1e-12 --max-disp-error 100                               \
  --disparity-list stereo/run-unaligned-D.tif                                    \
  --max-num-reference-points 40000 --reference-terrain srtm.tif                  \
  --solve-intrinsics --intrinsics-to-share 'focal_length optical_center'         \
  --intrinsics-to-float other_intrinsics --robust-threshold 10                   \
  --initial-transform pc_align/run-transform.txt
```

Here it is suggested to use a match file with dense interest points. The initial transform is the transform written by `pc_align` applied to the reference terrain and the DEM obtained with the camera models `for_small_rpc.tsai` and `aft_small_rpc.tsai` (with the reference terrain being the first of the two clouds passed to the alignment program). The unaligned disparity in the disparity list should be from the stereo run with these initial guess camera models (hence stereo should be used with the `--unalign-disparity` option). It is suggested that the optical center and focal lengths of the two cameras be kept fixed, as RPC distortion should be able model any changes in those quantities as well.

One can also experiment with the option `--heights-from-dem` instead of `--reference-terrain`. The former seems to be able to handle better large height differences between the DEM with the initial cameras and the reference terrain, while the former is better at refining the solution.

Then one can create a new DEM from the optimized camera models and see if it is an improvement.

11.18 Declassified satellite images: KH-7

KH-7 was an effective observation satellite that followed the Corona program. It contained an index (frame) camera and a single strip (pushbroom) camera. ASP does currently have a dedicated camera model for this camera, so we will have to try to approximate it with a pinhole model. Without a dedicated solution for this camera, you may only be able to get good results near the central region of the image.

For this example we find the following images in Earth Explorer declassified collection 2:

```
DZB00401800038H025001
DZB00401800038H026001
```

Make note of the lat/lon corners of the images listed in Earth Explorer, and note which image corners correspond to which compass locations.

After downloading and unpacking the images, we merge them with the `image_mosaic` tool. These images have a large amount of overlap and we need to manually lower the blend radius so that we do not have memory problems when merging the images. Note that the image order is different for each image.

```
image_mosaic DZB00401800038H025001_b.tif  DZB00401800038H025001_a.tif      \
```

```
  -o DZB00401800038H025001.tif  --ot byte --blend-radius 2000  --overlap-width 10000 \
image_mosaic DZB00401800038H026001_a.tif  DZB00401800038H026001_b.tif      \
  -o DZB00401800038H026001.tif  --ot byte --blend-radius 2000  --overlap-width 10000 \
```

For this image pair we will use the following SRTM images from Earth Explorer:

```
n22_e113_1arc_v3.tif
n23_e113_1arc_v3.tif
dem_mosaic n22_e113_1arc_v3.tif n23_e113_1arc_v3.tif -o srtm_dem.tif
```

(The SRTM DEM may need adjustment, as discussed in section 11.16.2.)

Next we crop the input images so they only contain valid image area.

```
historical_helper.py rotate-crop --input-path DZB00401800038H025001.tif \
--output-path 5001.tif  --interest-points '1847 2656  61348 2599  61338 33523  1880 33567'
historical_helper.py rotate-crop --input-path DZB00401800038H026001.tif \
--output-path 6001.tif  --interest-points '566 2678  62421 2683  62290 33596  465 33595'
```

We will try to approximate the KH7 camera using a pinhole model. The pitch of the image is determined by the scanner, which is 7.0e-06 meters per pixel. The focal length of the camera is reported to be 1.96 meters, and we will set the optical center at the center of the image. We need to convert the optical center to units of meters, which means multiplying the pixel coordinates by the pitch to get units of meters.

Using the image corner coordinates which we recorded earlier, use the `cam_gen` tool to generate camera models for each image, being careful of the order of coordinates.

```
cam_gen --pixel-pitch 7.0e-06 --focal-length 1.96                                  \
  --optical-center 0.2082535  0.1082305                                            \
  --lon-lat-values '113.25 22.882  113.315 23.315  113.6 23.282  113.532 22.85'    \
  5001.tif --reference-dem srtm_dem.tif --refine-camera -o 5001.tsai
cam_gen --pixel-pitch 7.0e-06 --focal-length 1.96                                  \
  --optical-center 0.216853 0.108227                                               \
  --lon-lat-values '113.2 22.95  113.265 23.382  113.565 23.35  113.482 22.915'    \
  6001.tif --reference-dem srtm_dem.tif --refine-camera  -o 6001.tsai
```

A quick way to evaluate the camera models is to use the `camera_footprint` tool to create KML footprint files, then look at them in Google Earth. For a more detailed view, you can map project them and overlay them on the reference DEM in `stereo_gui`.

```
camera_footprint 5001.tif  5001.tsai  --datum  WGS_1984 --quick \
  --output-kml  5001_footprint.kml -t nadirpinhole --dem-file srtm_dem.tif
camera_footprint 6001.tif  6001.tsai  --datum  WGS_1984 --quick \
  --output-kml  6001_footprint.kml -t nadirpinhole --dem-file srtm_dem.tif
```

The output files from `cam_gen` will be roughly accurate but they may still be bad enough that `bundle_adjust` has trouble finding a solution. One way to improve your initial models is to use ground control points. For this test case I was able to match features along the rivers to the same rivers in a hillshaded version of the reference DEM. I used three sets of GCPs, one for each image individually and a joint set for both images. I then ran `bundle_adjust` individually for each camera using the GCPs.

```
bundle_adjust 5001.tif 5001.tsai gcp_5001.gcp -t nadirpinhole --inline-adjustments \
  --num-passes 1 --camera-weight 0 --ip-detect-method 1 -o bundle_5001/out       \
  --max-iterations 30 --fix-gcp-xyz

bundle_adjust 6001.tif 6001.tsai gcp_6001.gcp -t nadirpinhole --inline-adjustments \
  --num-passes 1 --camera-weight 0 --ip-detect-method 1 -o bundle_6001/out       \
  --max-iterations 30 --fix-gcp-xyz
```

At this point it is a good idea to experiment with downsampled copies of the input images before running processing with the full size images. You can generate these using `stereo_gui`. Also make copies of the camera model files and scale the image center and pitch to match the downsample amount.

```
stereo_gui 5001.tif 6001.tif --create-image-pyramids-only
ln -s 5001_sub16.tif  5001_small.tif
ln -s 6001_sub16.tif  6001_small.tif
cp 5001.tsai  5001_small.tsai
cp 6001.tsai  6001_small.tsai
```

Now we can run `bundle_adjust` and `stereo`. If you are using the GCPs from earlier, the pixel values will need to be scaled to match the downsampling applied to the input images.

```
bundle_adjust 5001_small.tif 6001_small.tif bundle_5001/out-5001_small.tsai \
  bundle_6001/out-6001_small.tsai gcp_small.gcp -t nadirpinhole               \
  -o bundle_small_new/out --force-reuse-match-files --max-iterations 30     \
  --camera-weight 0 --disable-tri-ip-filter --disable-pinhole-gcp-init      \
  --skip-rough-homography --inline-adjustments --ip-detect-method 1         \
  --datum WGS84 --num-passes 2

stereo --alignment-method homography --skip-rough-homography              \
  --disable-tri-ip-filter --ip-detect-method 1 --session-type nadirpinhole  \
   5001_small.tif 6001_small.tif bundle_small_new/out-out-5001_small.tsai \
  bundle_small_new/out-out-6001_small.tsai st_small_new/out

gdal_translate -b 4 st_small_new/out-PC.tif st_small_new/error.tif
```

Looking at the error result, it is clear that the simple pinhole model is not doing a good job modeling the KH7 camera. We can try to improve things by adding a distortion model to replace the NULL model in the .tsai files we are using.

```
BrownConrady
xp  = -1e-12
yp  = -1e-12
k1  = -1e-10
k2  = -1e-14
k3  = -1e-22
p1  = -1e-12
p2  = -1e-12
phi = -1e-12
```

Once the distortion model is added, you can use `bundle_adjust` to optimize them. See the section on solving for pinhole intrinsics in the KH4B example for details. We hope to provide a more rigorous method of modeling the KH7 camera in the future.

11.19 Declassified satellite images: KH-9

The KH-9 satellite contained one frame camera and two panoramic cameras, one pitched forwards and one aft. The frame camera is a normal pinhole model so this example describes how to set up the panoramic cameras for processing. Processing this data is similar to processing KH-4B data except that the images are much larger.

For this example we use the following images from the Earth Explorer declassified collection 3:

```
D3C1216-200548A041
D3C1216-200548F040
```

Make note of the lat/lon corners of the images listed in Earth Explorer, and note which image corners correspond to which compass locations.

After downloading and unpacking the images, we merge them with the `image_mosaic` tool.

```
image_mosaic D3C1216-200548F040_a.tif  D3C1216-200548F040_b.tif  D3C1216-200548F040_c.tif \
  D3C1216-200548F040_d.tif  D3C1216-200548F040_e.tif  D3C1216-200548F040_f.tif           \
  D3C1216-200548F040_g.tif  D3C1216-200548F040_h.tif  D3C1216-200548F040_i.tif           \
  D3C1216-200548F040_j.tif  D3C1216-200548F040_k.tif  D3C1216-200548F040_l.tif           \
  --ot byte --overlap-width 3000 -o D3C1216-200548F040.tif
image_mosaic D3C1216-200548A041_a.tif  D3C1216-200548A041_b.tif  D3C1216-200548A041_c.tif \
  D3C1216-200548A041_d.tif  D3C1216-200548A041_e.tif  D3C1216-200548A041_f.tif           \
  D3C1216-200548A041_g.tif  D3C1216-200548A041_h.tif  D3C1216-200548A041_i.tif           \
  D3C1216-200548A041_j.tif  D3C1216-200548A041_k.tif --overlap-width 1000               \
  --ot byte -o D3C1216-200548A041.tif  --rotate
```

These images also need to be cropped to remove most of the area around the images:

```
historical_helper.py rotate-crop --input-path D3C1216-200548F040.tif --output-path for.tif \
  --interest-points '2414 1190  346001 1714  345952 23960  2356 23174'
historical_helper.py rotate-crop --input-path D3C1216-200548A041.tif --output-path aft.tif \
  --interest-points '1624 1333  346183 1812  346212 24085  1538 23504'
```

For this example there are ASTER DEMs which can be used for reference. They can be downloaded from https://gdex.cr.usgs.gov/gdex/ as single GeoTIFF files. To cover the entire area of this image pair you may need to download two files separately and merge them using `dem_mosaic`.

As with KH-4B, this satellite contains a forward pointing and aft pointing camera that need to have different values for "forward_tilt" in the sample camera files. The suggested values are -0.174533 for the aft camera and 0.174533 for the forward camera. Note that some KH9 images have a much smaller field of view (horizontal size) than others!

```
VERSION_4
```

```
OPTICAL_BAR
image_size = 62546 36633
image_center = 31273 18315.5
pitch = 7.0e-06
f = 1.5
scan_time = 0.7
forward_tilt = 0.174533
iC = -1053926.8825477704 5528294.6575468015 3343882.1925249361
iR = -0.96592328992496967 -0.16255393156297787 0.20141603042941184 \
     -0.23867502833024612 0.25834753840712932 -0.93610404349651921 \
     0.10013205696518604 -0.95227767417513032 -0.28834146846321851
speed = 8000
mean_earth_radius = 6371000
mean_surface_elevation = 0
motion_compensation_factor = 1
scan_dir = right
```

Camera files are generated using `cam_gen` from a sample camera file as in the previous examples.

```
cam_gen --sample-file sample_kh9_for_optical_bar.tsai --camera-type opticalbar          \
  --lon-lat-values '-151.954 61.999  -145.237 61.186  -145.298 60.944  -152.149 61.771' \
  for.tif --reference-dem aster_dem.tif --refine-camera  -o for.tsai
cam_gen --sample-file sample_kh9_aft_optical_bar.tsai --camera-type opticalbar         \
  --lon-lat-values '-152.124 61.913  -145.211 61.156  -145.43 60.938  -152.117 61.667' \
  aft.tif --reference-dem aster_dem.tif --refine-camera  -o aft.tsai
```

As with KH-4B, it is best to first experiment with low resolution copies of the images. Don't forget to scale the image size, center location, and pixel size in the new camera files!

```
stereo_gui for.tif aft.tif --create-image-pyramids-only
ln -s for_sub32.tif  for_small.tif
ln -s aft_sub32.tif  aft_small.tif
cp for.tsai  for_small.tsai
cp aft.tsai  aft_small.tsai
```

From this point KH-9 data can be processed in a very similar manner to the KH-4B example. Once again, you may need to vary some of the camera parameters to find the settings that produce the best results. For this example we will demonstrate how to use `bundle_adjust` to solve for intrinsic parameters in optical bar models.

Using the DEM and the input images it is possible to collect rough ground control points which can be used to roughly align the initial camera models.

```
bundle_adjust for_small.tif for_small.tsai ground_control_points.gcp -t opticalbar \
  --inline-adjustments --num-passes 1 --camera-weight 0 --ip-detect-method 1       \
  -o bundle_for_small/out --max-iterations 30 --fix-gcp-xyz

bundle_adjust aft_small.tif aft_small.tsai ground_control_points.gcp -t opticalbar \
  --inline-adjustments --num-passes 1 --camera-weight 0 --ip-detect-method 1       \
  -o bundle_aft_small/out --max-iterations 30 --fix-gcp-xyz
```

Now we can do a joint bundle adjustment. While in this example we immediately attempt to solve for intrinsics, you can get better results using techniques such as the `--disparity-list` option described in 11.17 and 11.16 along with the reference DEM. We will try to solve for all intrinsics but will share the focal length and optical center since we expect them to be very similar. If we get good values for the other intrinsics we could do another pass where we don't share those values in order to find small difference between the two cameras. We specify intrinsic scaling limits here. The first three pairs are for the focal length and the two optical center values. For an optical bar camera, the next three values are for `speed`, `motion_compensation_factor`, and `scan_time`. We are fairly confident in the focal length and the optical center but we only have guesses for the other values so we allow them to vary in a wider range.

```
bundle_adjust left_small.tif right_small.tif bundle_for_small/out-for_small.tsai \
  bundle_aft_small/out-aft_small.tsai -t opticalbar -o bundle_small/out          \
  --force-reuse-match-files --max-iterations 30 --camera-weight 0                \
  --disable-tri-ip-filter  --skip-rough-homography --inline-adjustments          \
  --ip-detect-method 1 --datum WGS84 --num-passes 2 --solve-intrinsics           \
  --intrinsics-to-float "focal_length optical_center other_intrinsics"           \
  --intrinsics-to-share "focal_length optical_center" --ip-per-tile 1000         \
  --intrinsics-limits "0.95 1.05   0.90 1.10  0.90 1.10   0.5 1.5  -5.0 5.0      \
  0.3 2.0" --num-random-passes 2
```

These limits restrict our parameters to reasonable bounds but unfortunately they greatly increase the run time of `bundle_adjust`. Hopefully you can figure out the correct values for `scan_dir` doing long optimization runs using the limits. The `--intrinsic-limits` option is useful when used in conjunction with the `--num-random-passes` option because it also sets the numeric range in which the random initial parameter values are chosen from. Note that `--num-passes` is intended to filter out bad interest points while `--num-random-passes` tries out multiple random starting seeds to see which one leads to the result with the lowest error.

Part III

Appendices

Appendix A

Tools

This chapter provides a overview of the various tools that are provided as part of the Ames Stereo Pipeline, and a summary of their command line options.

A.1 stereo

The `stereo` program is the primary tool of the Ames Stereo Pipeline. It takes a stereo pair of images that overlap and creates an output point cloud image that can be processed into a visualizable mesh or a DEM using `point2mesh` (section A.7) and `point2dem` (section A.6), respectively.

Usage:

```
ISIS 3> stereo [options] <images> [<cameras>] output_file_prefix
```

Example (for ISIS):

```
stereo file1.cub file2.cub results/run
```

For ISIS, a .cub file has both image and camera information, as such no separate camera files are specified.

Example (for Digital Globe Earth images):

```
stereo file1.tif file2.tif file1.xml file2.xml results/run
```

Multiple input images are also supported (section 5.1.8).

This tool is is primarily designed to process USGS ISIS `.cub` files and Digital Globe data. However, Stereo Pipeline does have the capability to process other types of stereo image pairs (e.g., image files with a CAHVOR camera model from the NASA MER rovers). If you would like to experiment with these features, please contact us for more information.

The *output_file_prefix* is prepended to all output data files. For example, setting *output_file_prefix* to '`out`' will yield files with names like `out-L.tif` and `out-PC.tif`. To keep the Stereo Pipeline results organized in sub-directories, we recommend using an output prefix like '`results-10-12-09/out`' for *output_file_prefix*. The `stereo` program will create a directory called `results-10-12-09/` and place files named `out-L.tif`, `out-PC.tif`, etc. in that directory.

Table A.1: Command-line options for stereo

Option	Description
`--help|-h`	Display the help message
`--session-type|-t` *string*	Select the stereo session type to use for processing. Usually the program can select this automatically by the file extension. Options: pinhole isis dg rpc spot5 aster opticalbar pinholemappinhole isismapisis dgmaprpc rpcmaprpc astermaprpc.
`--stereo-file|-s` *filename(=./stereo.default)*	Define the stereo.default file to use.
`--entry-point|-e integer(=0 to 5)`	Stereo Pipeline entry point (start at this stage).
`--stop-point|-e integer(=1 to 6)`	Stereo Pipeline stop point (stop at the stage *right before* this value).
`--corr-seed-mode integer(=0 to 3)`	Correlation seed strategy (section B.2).
`--threads` *integer(=0)*	Set the number of threads to use. 0 means use as many threads as there are cores.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.

More information about additional options that can be passed to `stereo` via the command line or via the `stereo.default` configuration file can be found in Appendix B on page 213. `stereo` creates a set of intermediate files, they are described in Appendix C on page 223.

A.1.1 Entry Points

The `stereo -e` *number* option can be used to restart a `stereo` job partway through the stereo correlation process. Restarting can be useful when debugging while iterating on `stereo.default` settings.

Stage 0 (Preprocessing) normalizes the two images and aligns them by locating interest points and matching them in both images. The program is designed to reject outlying interest points. This stage writes out the pre-aligned images and the image masks.

Stage 1 (Disparity Map Initialization) performs pyramid correlation and builds a rough disparity map that is used to seed the sub-pixel refinement phase.

Stage 2 (Blend) blend the borders of adjacent tiles. Only needed for parallel stereo with the SGM/MGM algorithms. Skipped otherwise.

Stage 3 (Sub-pixel Refinement) performs sub-pixel correlation that refines the disparity map.

Stage 4 (Outlier Rejection and Hole Filling) performs filtering of the disparity map and (optionally) fills in holes using an inpainting algorithm. This phase also creates a "good pixel" map.

Stage 5 (Triangulation) generates a 3D point cloud from the disparity map.

A.1.2 Decomposition of Stereo

The `stereo` executable is a python script that makes calls to separate C++ executables for each entry point.

Stage 0 (Preprocessing) calls `stereo_pprc`. Multi-threaded.

Stage 1 (Disparity Map Initialization) calls `stereo_corr`. Multi-threaded.

Stage 2 (Blend) class `stereo_blend`. Multi-threaded.

Stage 3 (Sub-pixel Refinement) class `stereo_rfne`. Multi-threaded.

Stage 4 (Outlier Rejection and Hole Filling) calls `stereo_fltr`. Multi-threaded.

Stage 5 (Triangulation) calls `stereo_tri`. Multi-threaded, except for ISIS input data.

All of the sub-programs have the same interface as `stereo`. Users processing a large number of stereo pairs on a cluster may find it advantageous to call these executables in their own manner. An example would be to run stages 0-4 in order for each stereo pair. Then run several sessions of `stereo_tri` since it is single-threaded for ISIS.

It is important to note that each of the C++ stereo executables invoked by `stereo` have their own command-line options. Those options can be passed to `stereo` which will in turn pass them to the appropriate executable. By invoking each executable with no options, it will display the list of options it accepts.

As explained in more detail in section 5.1.3, each such option has the same syntax as used in `stereo.default`, while being prepended by a double hyphen (`--`). A command line option takes precedence over the same option specified in `stereo.default`. Chapter B documents all options for the individual sub-programs.

Note that the stereo tools operate only on single channel (grayscale) images. If you need to run stereo on multi-channel images you must first convert them to grayscale or extract a single channel to operate on.

A.2 stereo_gui

The `stereo_gui` program is a GUI frontend to `stereo`, and has the same command-line options. It can display the input images side-by-side (and in other ways, as detailed later). One can zoom in by dragging the mouse from upper-left to lower-right, and zoom out via the reverse motion.

By pressing the `Control` key while dragging the mouse, regions can be selected in the input images, and then stereo can be run on these regions from the menu via Run→Stereo. The `stereo` command that is invoked (with appropriately populated parameter values for `--left-image-crop-win` and `--right-image-crop-win` for the selected regions) will be displayed on screen, and can be re-run on a more powerful machine/cluster without GUI access.

Additional navigation options are using the mouse wheel or the +/- keys to zoom, and the arrow keys to pan (one should first click to bring into focus the desired image before using any keys).

Usage:

```
ISIS 3> stereo_gui [options] <images> [<cameras>] output_file_prefix
```

A.2.1 Use as an Image Viewer

This program can be also used as a general-purpose image viewer, case in which no stereo options or camera information is necessary. It can display arbitrarily large images with integer, floating-point, or RGB pixels, including ISIS .cub files and DEMs. It handles large images by building on disk pyramids of increasingly coarser subsampled images and displaying the subsampled versions that are appropriate for the current level of zoom.

The images can be shown either side-by-side, as tiles on a grid (using `--grid-cols integer`), or on top of each other (using `--single-window`), with a dialog to choose among them. In the last usage scenario, the

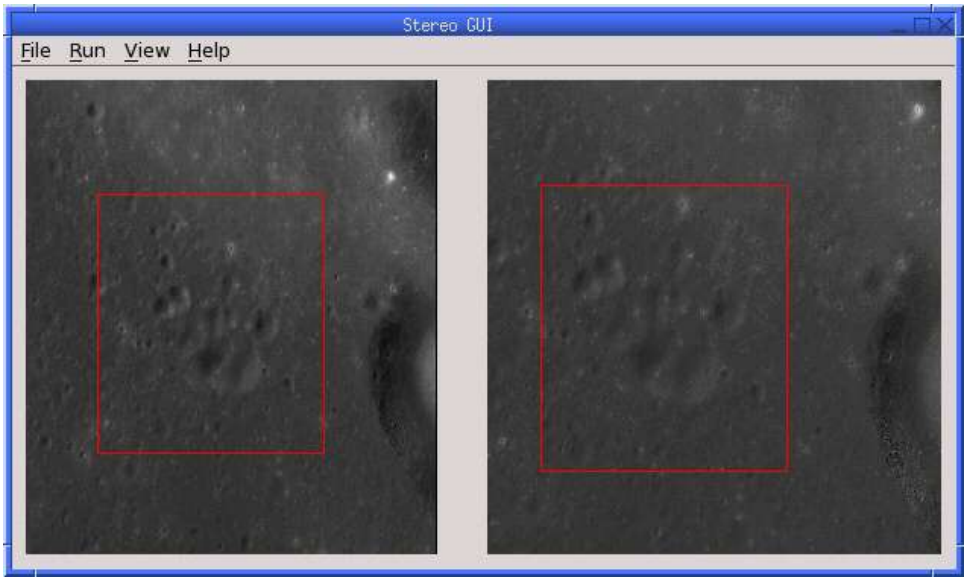


Figure A.1: An illustration of stereo_gui. The `stereo` command will be run on the regions selected by red rectangles.

option `--use-georef` will overlay the images correctly if georeference information is present. It is possible to switch among these modes once the GUI has been open, from the GUI View menu.

When the images are shown side-by-side, the GUI can zoom in all images to the same region, for easier comparison among them.

When the images are in a single window, an individual image can be turned on or off via a checkbox. Clicking on an image's name will zoom to it and display it on top of other images. By right-clicking the list of images, other operations can be performed, such as deleting an image from the view, etc.

`stereo_gui` can show hillshaded DEMs, either via the `--hillshade` option, or by choosing from the GUI View menu the `Hillshaded images` option.

This program can also display the output of the ASP `colormap` tool (section A.29).

When clicking on a pixel, the pixel indices and value will be printed on screen. When selecting a region by pressing the `Control` key while dragging the mouse, its bounds will be displayed on screen. If the image is geo-referenced, the extent of the region in projected coordinates and in the longitude-latitude domain will be shown as well.

The program can also save a screenshot to disk in the BMP or XPM format.

A.2.2 Other Functionality

View/create/move/delete/save interest point matches, GCP, and VWIP files

`stereo_gui` can be used to view interest point matches (`*.match` files), such as generated by `ipmatch`, `bundle_adjust`, or `stereo`. It can also manually create and delete matches (useful in situations when automatic interest point matching is unreliable due to large changes in illumination). Interest point matches can be created or deleted with the right-mouse click. To move interest points, right click on a panel and check "Move match point". While this is checked you can move interest points by clicking and dragging them within the panel. Uncheck "Move match point" to stop moving interest points.

The match file to load can be specified via `--match-file`. It may also be auto-detected if `stereo_gui` was invoked like `stereo`, with an output prefix (auto-detection works only when images are not map-projected and alignment is homography or affine epipolar). Match files can be created with the `ipmatch` tool or by using `stereo_pprc`.

When working with N images, N-1 match files are needed to describe all of the interest points. For image i, the match file must contain the matches from image i-1 or from image 0. You can provide these match files to `stereo_gui` by conforming to its naming convention (prefix-fname1__fname2.match) or by selecting them from the GUI when prompted. All match files must describe the same set of interest points, the tool will check the positions of loaded points and discard any that do not correspond to the already loaded points. If one of the match files fails to load or does not contain enough matching points, the missing points will be added to an arbitrary position and flagged as invalid. You must either validate these points by manually moving them to the correct position or else delete them.

This tool can show the interest points from a GCP file (but cannot edit them with this interface, creating such points is described later in this section). The `--gcp-file` option is used.

The `stereo_gui` program can also display `.vwip` files. Those are interest points created by `ipfind`, `bundle_adjust`, or `stereo`, before they are matched across images. One should specify as many such files as images when launching this program.

Creating GCP with a georeferenced image and a DEM

There exist situations when one has one or more images for which the camera files are either inaccurate or, for Pinhole camera models, just the intrinsics may be known. Given a DEM of the area, and optionally a georeferenced image, it is possible to create GCP files (section A.4.1) that can later be used with `bundle_adjust` to either improve the alignment of these cameras to the DEM, or create new Pinhole cameras from scratch (the latter is shown in section 9.4).

One starts by opening these desired images and the georeferenced image in the GUI, in this order (hence the georeferenced image is the last). If no georeferenced image exists, one can use the given DEM instead (and it can be hillshaded after loading to easier identify features).

Next, a feature is identified and manually added as an interest point in all open images, using the right-click menu, and this process is repeated a few times. Thus created interest points can also be moved around by right-clicking to turn on this mode, and then dragging them with the mouse (this can be slow).

If the input images and the georeferenced image are very similar visually, one can also try to automatically detect interest point matches in them using `ipfind`/`ipmatch` and load the .match files as described in the earlier section on creating interest points.

When you are finished creating interest points, use the "IP matches"->"Write GCP file" menu item to generate a ground control point file containing the selected points. You will be prompted for the reference DEM and for the desired output file name, unless this DEM was already specified via `--dem-file` upon launch and the GCP file was already specified via `--gcp-file`. The last image, that is the reference, is only used to find the positions on the ground, which in turn are used to find the heights for the GCPs from the DEM. The selected interest points from the reference image are not saved to the GCP file.

Creating manual interest point matches using map-projected images

To make it easier to pick interest point matches in situations when the images are very different or taken from very diverse perspectives, it is easier to first mapproject them onto a DEM, as then the images look a lot more similar. Once interest points are created among the map-projected images in the GUI, some functionality can be invoked to transfer them to the original images.

Here is an example. Given three images A.tif, B.tif, and C.tif, and a DEM named dem.tif, map-project them onto the DEM, obtaining the images A.map.tif, B.map.tif, and C.map.tif. Then one can invoke `stereo_gui` as

```
stereo_gui A.map.tif B.map.tif C.map.tif run/run
```

pick interest points by right-clicking, save them, and quit the GUI. Then one runs:

```
bundle_adjust A.tif B.tif C.tif <cameras> run/run \
--mapprojected-data 'A.map.tif B.map.tif C.map.tif dem.tif' --min-matches 0
```

which will create match files among the original images and run bundle adjustment.

One can then invoke

```
stereo_gui A.tif B.tif C.tif run/run
```

and turn on viewing of interest point matches to study if they were "unmapped" to the right locations.

Polygon editing

`stereo_gui` is able to draw and edit polygonal shapes on top of georeferenced images, and load/save them as shape files (*.shp). This functionality can be accessed by turning on polygon editing from the *Vector Layer* menu, and then right-clicking with the mouse to access the various functions. Subsequently, one can use `gdal_rasterize` to keep or exclude the portion of a given image/DEM that is within or outside the selected polygon.

Shadow threshold

`stereo_gui` can be used to find the shadow threshold for each of a given set of images (useful for shape-from-shading, see chapter 10). This can be done by turning on from the menu the `Shadow threshold detection` mode, and then clicking on pixels in the shadow. The largest of the chosen pixel values will be set to the shadow threshold for each image and printed to the screen. To see the images with the pixels below the shadow threshold highlighted, select from the menu the `View shadow-thresholded images` option.

Somewhat related to this, if the viewer is invoked with `--nodata-value` *`double`*, it will display any pixels (presumably in the background) with values less than or equal to this as transparent.

Listed below are the options specific to `stereo_gui`. It will accept all other `stereo` options as well.

Table A.2: Command-line options for stereo_gui

Option	Description
`-h | --help`	Display this help message.
`--grid-cols arg`	Display images as tiles on a grid with this many columns. Default: Use one row.
`--window-size arg (=1200 800)`	The width and height of the GUI window in pixels.
`-w | --single-window`	Show all images in the same window (with a dialog to choose among them) rather than next to each other.
`--use-georef`	Plot the images in the projected coordinate system given by image georeferences.
`--nodata-value` *double(=NaN)*	Pixels with values less than or equal to this number are treated as no-data and displayed as transparent. This overrides the no-data values from input images.
`--hillshade`	Interpret the input images as DEMs and hillshade them.
`--hillshade-azimuth`	The azimuth value when showing hillshaded images.
`--hillshade-elevation`	The elevation value when showing hillshaded images.
`--view-matches`	Locate and display the interest point matches.
`--match-file`	Display this match file instead of looking one up based on existing conventions (implies `--view-matches`).
`--gcp-file`	Display the GCP pixel coordinates for this GCP file (implies `--view-matches`). Also save here GCP if created from the GUI.
`--dem-file`	Use this DEM when creating GCP from images.
`--delete-temporary-files-on-exit`	Delete any subsampled and other files created by the GUI when exiting.
`--create-image-pyramids-only`	Without starting the GUI, build multi-resolution pyramids for the inputs, to be able to load them fast later.

A.3 parallel_stereo

The `parallel_stereo` program is a modification of `stereo` designed to distribute the stereo processing over multiple computing nodes. It uses GNU Parallel to manage the jobs, a tool which is distributed along with Stereo Pipeline. It expects that all nodes can connect to each other using ssh without password and that they share the same storage space. `parallel_stereo` can also be useful when processing extraterrestrial data on a single computer. This is because ISIS camera models are restricted to a single thread, but `parallel_stereo` can run multiple processes in parallel to reduce computation times.

At the simplest, `parallel_stereo` can be invoked exactly like `stereo`, with the addition of the list of nodes to use (if using multiple nodes).

```
parallel_stereo --nodes-list machines.txt <other stereo options>
```

It will create the same output files as `stereo`. Internally some of them will be GDAL VRT files, that is, plain text virtual mosaics of files created by individual processes, with the actual files in subdirectories; ASP and GDAL tools are able to use these virtual files in the same way as regular binary TIF files.

If your jobs are launched on a cluster or supercomputer, the name of the file containing the list of nodes may exist as an environmental variable. For example, on NASA's Pleiades Supercomputer, which uses the Portable Batch System (PBS), the list of nodes can be retrieved as $PBS_NODEFILE.

It is important to note that when invoking this tool only the correlation, blending, subpixel refinement, and triangulation stages of stereo (section A.1.2) are spread over multiple machines, with the preprocessing and filtering stages using just one node, as they require global knowledge of the data. In addition, not all stages of stereo benefit equally from parallelization. Most likely to gain are stages 1 and 2 (correlation and refinement) which are the most computationally expensive.

For these reasons, while `parallel_stereo` can be called to do all stages of stereo generation from start to finish in one command, it may be more resource-efficient to invoke it using a single node for stages 0 and 3, many nodes for stages 1 and 2, and just a handful of nodes for stage 4 (triangulation). For example, to invoke the tool only for stage 2, one uses the options:

```
--entry-point 2 --stop-point 3
```

By default, stages 1, 2, and 4 of `parallel_stereo` use as many processes as there are cores on each node, and one thread per process. These can be customized as shown below.

Table A.3: Command-line options for parallel_stereo

Options	Description
`--help|-h`	Display the help message.
`--nodes-list` *`filename`*	The list of computing nodes, one per line. If not provided, run on the local machine.
`--ssh` *`filename`*	Specify the path to an alternate version of the ssh tool to use.
`--entry-point|-e integer(=0 to 4)`	Stereo Pipeline entry point (start at this stage).
`--stop-point|-e integer(=1 to 5)`	Stereo Pipeline stop point (stop at the stage *right before* this value).
`--corr-seed-mode integer(=0 to 3)`	Correlation seed strategy (section B.2).
`--sparse-disp-options` *`string`*	Options to pass directly to sparse_disp (section 4.5).
`--verbose`	Display the commands being executed.
`--job-size-w` *`integer(=2048)`*	Pixel width of input image tile for a single process.
`--job-size-h` *`integer(=2048)`*	Pixel height of input image tile for a single process.
`--processes` *`integer`*	The number of processes to use per node.
`--threads-multiprocess` *`integer`*	The number of threads to use per process.

`--threads-singleprocess` *integer*	The number of threads to use when running a single process (for pre-processing and filtering).

A.4 bundle_adjust

The `bundle_adjust` program performs bundle adjustment on a given set of images and cameras. An introduction to bundle adjustment, and some advanced usage, including solving for intrinsics, can be found in chapter 8.

This tool can use several underlying least-squares minimization algorithms, the default is Google's Ceres Solver (http://ceres-solver.org/).

Usage:

```
bundle_adjust <images> <cameras> <optional ground control points> \
  -o <output prefix> [options]
```

Example (for ISIS):

```
bundle_adjust file1.cub file2.cub file3.cub -o run_ba/run
```

Example (for Digital Globe Earth data, using ground control points):

```
bundle_adjust file1.tif file2.tif file1.xml file2.xml gcp_file.gcp \
  --datum WGS_1984 -o run_ba/run --num-passes 2
```

Here, we invoked the tool with two passes, which also enables removal of outliers by reprojection error and disparity (the options below have more detail).

Example (for generic pinhole camera data, using estimated camera positions):

```
bundle_adjust file1.JPG file2.JPG file1.tsai file2.tsai -o run_ba/run  \
   -t nadirpinhole --inline-adjustments --datum WGS_1984               \
   --camera-positions nav_data.csv                                     \
   --csv-format "1:file 6:lat 7:lon 9:height_above_datum"
```

Here we assumed that the cameras point towards some planet's surface and used the `nadirpinhole` session. If this assumption is not true one should use the `pinhole` session, though this one often does not perform as well when finding interest points in planetary context.

This tool will write the adjustments to the cameras as `*.adjust` files starting with the specified output prefix. In order for `stereo` to use the adjusted cameras, it should be passed this output prefix via the option `--bundle-adjust-prefix`. For example,

```
stereo file1.cub file2.cub run_stereo/run --bundle-adjust-prefix run_ba/run
```

If the `--inline-adjustments` option is used, no separate adjustments will be written, rather, the tool will save to disk copies of the input cameras with adjustments already applied to them. These output cameras can then be passed directly to stereo:

```
stereo file1.JPG file2.JPG run_ba/run-file1.tsai run_ba/run-file2.tsai run_stereo/run
```

The `bundle_adjust` program can read camera adjustments from a previous run, via `--input-adjustments-prefix` *`string`*. It can also apply to the input cameras a transform as output by `pc_align`, via `--initial-transform` *`string`*. This is useful if a DEM produced by ASP was aligned to a ground truth, and it is desired to apply

the same alignment to the cameras that were used to create that DEM. The initial transform can have a rotation, translation, and scale, and it is applied after the input adjustments are read, if those are present.

If the `--datum` option is specified, `bundle_adjust` will write the mean absolute residuals (reprojection errors) for each triangulated point, before and after optimization. The files are named

```
{output-prefix}-initial_residuals_no_loss_function_pointmap_point_log.csv
```

and

```
{output-prefix}-final_residuals_no_loss_function_pointmap_point_log.csv
```

(there are also versions of these files incorporating the Ceres loss function, which attenuates large residuals, those have in their names `loss` rather than `no_loss`). Such files can be inspected to see at which pixels the residual error is large. One can also invoke `point2dem` with the `--csv-format` option to grid these files for visualization in the GUI. Here is a sample file:

```
# lon, lat, height_above_datum, mean_residual, num_observations
-55.1169093561696002, -69.3430771656333178, 4.82452381754674064, 0.114133363354161105, 2
```

The field `num_observations` counts how many images each point gets projected into.

A.4.1 Ground Control Points

A number of plain-text files containing ground control points (GCP) can be passed as inputs to `bundle_adjust`.

These can either be created by hand, or using `stereo_gui` (section A.2.2).

A GCP file must end with a .gcp extension, and contain one ground control point per line. Each line must have the following fields:

- ground control point id (integer)
- latitude (in degrees)
- longitude (in degrees)
- height above datum (in meters), with the datum itself specified separately
- x, y, z standard deviations (three positive floating point numbers, smaller values suggest more reliable measurements)

On the same line, for each image in which the ground control point is visible there should be:

- image file name
- column index in image (float)
- row index in image (float)
- column and row standard deviations (two positive floating point numbers, smaller values suggest more reliable measurements)

The fields can be separated by spaces or commas. Here is a sample representation of a ground control point measurement:

```
5 23.7 160.1 427.1 1.0 1.0 1.0 image1.tif 124.5 19.7 1.0 1.0 image2.tif 254.3 73.9 1.0 1.0
```

When the `--use-lon-lat-height-gcp-error` flag is used, the three standard deviations are interpreted as applying not to x, y, z but to latitude, longitude, and height above datum (in this order). Hence, if the latitude and longitude are known accurately, while the height less so, the third standard deviation can be set to something larger.

Table A.4: Command-line options for bundle_adjust

Option	Description
`--help|-h`	Display the help message.
`--output-prefix|-o` *filename*	Prefix for output filenames.
`--cost-function` *string*	Choose a cost function from: Cauchy, PseudoHuber, Huber, L1, L2. Default: Cauchy.
`--robust-threshold` *double(=0.5)*	Set the threshold for robust cost functions. Increasing this makes the solver focus harder on the larger errors.
`--datum` *string*	Use this datum. Needed only for ground control points, a camera position file, or for RPC sessions. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).
`--semi-major-axis` *double*	Explicitly set the datum semi-major axis in meters.
`--semi-minor-axis` *double*	Explicitly set the datum semi-minor axis in meters.
`--session-type|-t` *string*	Select the stereo session type to use for processing. Usually the program can select this automatically by the file extension. Options: pinhole nadirpinhole isis dg rpc spot5 aster opticalbar.
`--min-matches` *integer(=30)*	Set the minimum number of matches between images that will be considered.
`--num-iterations` *integer(=100)*	Set the maximum number of iterations.
`--parameter-tolerance` *double(=1e-8)*	Stop when the relative error in the variables being optimized is less than this.
`--overlap-limit` *integer(=0)*	Limit the number of subsequent images to search for matches to the current image to this value. By default try to match all images.
`--overlap-list` *string*	A file containing a list of image pairs, one pair per line, separated by a space, which are expected to overlap. Matches are then computed only among the images in each pair.
`--auto-overlap-buffer` *double*	Try to automatically determine which images overlap. Only supports Worldview style XML camera files.
`--rotation-weight` *double(=0.0)*	A higher weight will penalize more rotation deviations from the original configuration.
`--translation-weight` *double(=0.0)*	A higher weight will penalize more translation deviations from the original configuration.

`--camera-weight` *double(=1.0)*	The weight to give to the constraint that the camera positions/orientations stay close to the original values (only for the Ceres solver). A higher weight means that the values will change less. The options --rotation-weight and --translation-weight can be used for finer-grained control and a stronger response.
`--ip-per-tile` *integer*	How many interest points to detect in each 1024^2 image tile (default: automatic determination).
`--ip-detect-method` *integer(=0)*	Choose an interest point detection method from: 0=OBAloG, 1=SIFT, 2=ORB.
`--epipolar-threshold` *double(=-1)*	Maximum distance from the epipolar line to search for IP matches. Default: automatic calculation.
`--ip-inlier-factor` *double(=1.0/15)*	A higher factor will result in more interest points, but perhaps also more outliers.
`--ip-uniqueness-threshold` *double(=0.7)*	A higher threshold will result in more interest points, but perhaps less unique ones.
`--nodata-value` *double(=NaN)*	Pixels with values less than or equal to this number are treated as no-data. This overrides the no-data values from input images.
`--individually-normalize`	Individually normalize the input images instead of using common values.
`--inline-adjustments`	If this is set, and the input cameras are of the pinhole or panoramic type, apply the adjustments directly to the cameras, rather than saving them separately as .adjust files.
`--input-adjustments-prefix` *string*	Prefix to read initial adjustments from, written by a previous invocation of this program.
`--initial-transform` *string*	Before optimizing the cameras, apply to them the 4x4 rotation + translation transform from this file. The transform is in respect to the planet center, such as written by pc_align's source-to-reference or reference-to-source alignment transform. Set the number of iterations to 0 to stop at this step. If –input-adjustments-prefix is specified, the transform gets applied after the adjustments are read.
`--fixed-camera-indices` *string*	A list of indices, in quotes and starting from 0, with space as separator, corresponding to cameras to keep fixed during the optimization process.
`--fix-gcp-xyz`	If the GCP are highly accurate, use this option to not float them during the optimization.
`--use-lon-lat-height-gcp-error`	When having GCP, interpret the three standard deviations in the GCP file as applying not to x, y, and z, but rather to latitude, longitude, and height.
`--solve-intrinsics`	Optimize intrinsic camera parameters. Only used for pinhole cameras.
`--intrinsics-to-float arg`	If solving for intrinsics and desired to float only a few of them, specify here, in quotes, one or more of: focal_length, optical_center, other_intrinsics.

`--intrinsics-to-share arg`	If solving for intrinsics and desired to share only a few of them, specify here, in quotes, one or more of: focal_length, optical_center, other_intrinsics. By default all of the intrinsics are shared so to not share any of them pass in a blank string.
`--intrinsics-limits arg`	Set a string in quotes that contains min max ratio pairs for intrinsic parameters. For example, "0.8 1.2" limits the parameter to changing by no more than 20 percent. The first pair is for focal length, the next two are for the center pixel, and the remaining pairs are for other intrinsic parameters. If too many pairs are passed in the program will throw an exception and print the number of intrinsic parameters the cameras use. Cameras adjust all of the parameters in the order they are specified in the camera model unless it is specified otherwise in D.1. Unfortunately, setting limits can greatly slow down the solver.
`--num-passes` *integer(=2)*	How many passes of bundle adjustment to do. If more than one, outliers will be removed between passes using --remove-outliers-params and --remove-outliers-by-disparity-params, and re-optimization will take place. Residual files and a copy of the match files with the outliers removed will be written to disk.
`--num-random-passes` *integer(=0)*	After performing the normal bundle adjustment passes, do this many more passes using the same matches but adding random offsets to the initial parameter values with the goal of avoiding local minima that the optimizer may be getting stuck in. Only the results for the optimization pass with the lowest error are kept.
`--remove-outliers-params` *'pct factor err1 err2'*	Outlier removal based on percentage, when more than one bundle adjustment pass is used. Triangulated points with reprojection error in pixels larger than min(max('pct'-th percentile * 'factor', err1), err2) will be removed as outliers. Hence, never remove errors smaller than err1 but always remove those bigger than err2. Specify as a list in quotes. Default: '75.0 3.0 2.0 3.0'.
`--remove-outliers-by-disparity-params` *pct factor*	Outlier removal based on the disparity of interest points (difference between right and left pixel), when more than one bundle adjustment pass is used. For example, the 10% and 90% percentiles of disparity are computed, and this interval is made three times bigger. Interest points whose disparity fall outside the expanded interval are removed as outliers. Instead of the default 90 and 3 one can specify pct and factor, without quotes.

`--elevation-limit` *double double*	Remove as outliers interest points for which the elevation of the triangulated position (after cameras are optimized) is outside of this range. Specify as two values: min max.
`-lon-lat-limit` *double double double double*	Remove as outliers interest points for which the longitude and latitude of the triangulated position (after cameras are optimized) are outside of this range. Specify as: min_lon min_lat max_lon max_lat.
`--reference-terrain arg`	An externally provided trustworthy 3D terrain, either as a DEM or as a lidar file, very close (after alignment) to the stereo result from the given images and cameras that can be used as a reference, instead of GCP, to optimize the intrinsics of the cameras.
`--max-num-reference-points` *integer(=100000000)*	Maximum number of (randomly picked) points from the reference terrain to use.
`--disparity-list arg`	The unaligned disparity files to use when optimizing the intrinsics based on a reference terrain. Specify them as a list in quotes separated by spaces. First file is for the first two images, second is for the second and third images, etc. If an image pair has no disparity file, use 'none'.
`--max-disp-error` *double(=-1)*	When using a reference terrain as an external control, ignore as outliers xyz points which projected in the left image and transported by disparity to the right image differ by the projection of xyz in the right image by more than this value in pixels.
`--reference-terrain-weight` *double(=1)*	How much weight to give to the cost function terms involving the reference terrain.
`--heights-from-dem` *string*	If the cameras have already been bundle-adjusted and aligned to a known high-quality DEM, in the triangulated xyz points replace the heights with the ones from this DEM, and fix those points unless --heights-from-dem-weight is positive.
`--heights-from-dem-weight` *double(=-1)*	How much weight to give to keep the triangulated points close to the DEM if specified via --heights-from-dem. If the weight is not positive, keep the triangulated points fixed.

`--csv-format` *string*	Specify the format of input CSV files as a list of entries column_index:column_type (indices start from 1). Examples: '1:x 2:y 3:z 4:file' (a Cartesian coordinate system with origin at planet center is assumed, with the units being in meters), '5:lon 6:lat 7:radius_m 2:file' (longitude and latitude are in degrees, the radius is measured in meters from planet center), '6:file 3:lat 2:lon 1:height_above_datum', '1:easting 2:northing 3:height_above_datum' (need to set `--csv-proj4`; the height above datum is in meters). Can also use radius_km for column_type, when it is again measured from planet center.
`--csv-proj4` *string*	The PROJ.4 string to use to interpret the entries in input CSV files, if those files contain Easting and Northing fields.
`--min-triangulation-angle` *double(=0.1)*	The minimum angle, in degrees, at which rays must meet at a triangulated point to accept this point as valid.
`--ip-triangulation-max-error` *double*	When matching IP, filter out any pairs with a triangulation error higher than this.
`--forced-triangulation-distance` *double*	When triangulation fails, for example, when input cameras are inaccurate, artificially create a triangulation point this far ahead of the camera, in units of meter.
`--ip-num-ransac-iterations` *int(=1000)*	How many RANSAC iterations to do in interest point matching.
`--save-cnet-as-csv`	Save the initial control network containing all interest points in the format used by ground control points, so it can be inspected.
`--camera-positions` *filename*	CSV file containing estimated positions of each camera. Only used with the inline-adjustments setting to initialize global camera coordinates. If used, the csv-format setting must also be set. The "file" field is searched for strings that are found in the input image files to match locations to cameras.
`--disable-pinhole-gcp-init`	Don't try to initialize pinhole camera coordinates using provided GCP coordinates. Set this if you only have one image per GCP or if the pinhole initialization process is not producing good results.
`--transform-cameras-using-gcp`	Use GCP, even those that show up in just an image, to transform cameras to ground coordinates. Need at least two images to have at least 3 GCP each. If at least three GCP each show up in at least two images, the transform will happen even without this option using a more robust algorithm.

`--position-filter-dist` *double(=-1.0)*	If estimated camera positions are used, this option can be used to set a threshold distance in meters between the cameras. If any pair of cameras is farther apart than this distance, the tool will not attempt to find matching interest points between those two cameras.
`--force-reuse-match-files`	Force reusing the match files even if older than the images or cameras.
`--enable-rough-homography`	Enable the step of performing datum-based rough homography for interest point matching. This is best used with reasonably reliable input cameras and a wide footprint on the ground.
`--skip-rough-homography`	Skip the step of performing datum-based rough homography. This obsolete option is ignored as is the default.
`--enable-tri-ip-filter`	Enable triangulation-based interest points filtering. This is best used with reasonably reliable input cameras.
`--disable-tri-ip-filter`	Disable triangulation-based interest points filtering. This obsolete option is ignored as is the default.
`--no-datum`	Do not assume a reliable datum exists, such as for irregularly shaped bodies.
`--mapprojected-data` *string*	Given map-projected versions of the input images, the DEM they were mapprojected onto, and IP matches among the mapprojected images, create IP matches among the un-projected images before doing bundle adjustment. Specify the mapprojected images and the DEM as a string in quotes, separated by spaces. An example is in the documentation.
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default defined in the program or in the .vwrc file. Note that when using more than one thread and the Ceres option the results will vary slightly each time the tool is run.
`--report-level|-r` *integer=(10)*	Use a value >= 20 to get increasingly more verbose output.

A.5 parallel_bundle_adjust

The `parallel_bundle_adjust` program is a modification of `bundle_adjust` designed to distribute some of the preprocessing steps over multiple processes and multiple computing nodes. It uses GNU Parallel to manage the jobs in the same manner as `parallel_stereo`. For information on how to set up and use the node list see A.3.

The `parallel_bundle_adjust` has three processing steps: statistics, matching, and optimization. Only the first two steps can be done in parallel and in fact after you have run steps 0 and 1 in a folder with `parallel_bundle_adjust` you could just call regular `bundle_adjust` to complete processing in the folder. Steps 0 and 1 produce the -stats.tif and .match files that are used in the last step.

Table A.5: Command-line options for parallel_bundle_adjust

Options	Description
`--help|-h`	Display the help message.
`--nodes-list` *filename*	The list of computing nodes, one per line. If not provided, run on the local machine.
`--entry-point|-e integer(=0 to 4)`	Stereo Pipeline entry point (start at this stage).
`--stop-point|-e integer(=1 to 5)`	Stereo Pipeline stop point (stop at the stage *right before* this value).
`--verbose`	Display the commands being executed.
`--processes` *integer*	The number of processes to use per node.
`--threads-multiprocess` *integer*	The number of threads to use per process.
`--threads-singleprocess` *integer*	The number of threads to use when running a single process (for pre-processing and filtering).

A.6 point2dem

The `point2dem` program produces a GeoTIFF terrain model and/or an orthographic image from a set of point clouds. The clouds can be created by the `stereo` command, or be in LAS or CSV format.

Example:

```
point2dem output-prefix-PC.tif -o stereo/filename \
    --nodata-value -10000 -n
```

This produces a digital elevation model. The program will infer the spheroid (datum) and the projection to use from the input images, if that information is present. Otherwise these can be set with `-r` and `--t_srs`.

Here, pixels with no data will be set to a value of -10000. Unless the input images have projection information, the resulting DEM will be saved in a simple cylindrical map-projection. The DEM is stored by default as a one channel, 32-bit floating point GeoTIFF file.

The `-n` option creates an 8-bit, normalized version of the DEM that can be easily loaded into a standard image viewing application for debugging.

Another example:

```
point2dem output-prefix-PC.tif -o stereo/filename -r moon \
    --orthoimage output-prefix-L.tif
```

This command takes the left input image and orthographically projects it onto the 3D terrain produced by the Stereo Pipeline. The resulting `*-DRG.tif` file will be saved as a GeoTIFF image with the same geoheader as the DEM.

Here we have explicitly specified the spheroid (`-r moon`), rather than have it inferred automatically. The Moon spheroid will have a radius of 1737.4 km.

In the following example the point cloud is very close to the South Pole of the Moon, and for that reason we use the stereographic projection:

```
point2dem --stereographic --proj-lon 0 --proj-lat -90 output-prefix-PC.tif
```

Multiple point clouds can be passed as inputs, to be combined into a single DEM. If it is desired to use the `--orthoimage` option as above, the clouds need to be specified first, followed by the `L.tif` images. Here is an example, which combines together LAS and CSV point clouds together with an output file from `stereo`:

```
point2dem in1.las in2.csv output-prefix-PC.tif -o combined \
  --dem-spacing 0.001 --nodata-value -32768
```

A.6.1 Comparing with MOLA Data

When comparing the output of `point2dem` to laser altimeter data, like MOLA, it is important to understand the different kinds of data that are being discussed. By default, `point2dem` returns planetary radius values in meters. These are often large numbers that are difficult to deal with. If you use the `-r mars` option, the output terrain model will be in meters of elevation with reference to the IAU reference spheroid for Mars: 3,396,190 m. So if a post would have a radius value of 3,396,195 m, in the model returned with the `-r mars` option, that pixel would just be 5 m.

You may want to compare the output to MOLA data. MOLA data is released in three 'flavors,' namely: Topography, Radius, and Areoid. The MOLA Topography data product that most people use is just the MOLA Radius product with the MOLA Areoid product subtracted. Additionally, it is important to note that all of these data products have a reference value subtracted from them. The MOLA reference value is NOT the IAU reference value, but 3,396,000 m.

In order to compare with the MOLA data, you can do one of two different things. You could operate purely in radius space, and have `point2dem` create radius values that are directly comparable to the MOLA radius data. You can do this by having `point2dem` subtract the MOLA reference value, by using either `-r mola` or setting `--semi-major-axis 3396000` and `--semi-minor-axis 3396000`.

Alternatively, to get values that are directly comparable to MOLA *Topography* data, you'll need to run `point2dem` with either `-r mars` or `-r mola`, then run the ASP tool `dem_geoid` (section A.10). This program will convert the DEM height values from being relative to the IAU reference spheroid or the MOLA spheroid to being relative to the MOLA Areoid.

The newly obtained DEM will inherit the datum from the unadjusted DEM, so it could be either of the two earlier encountered radii, but of course the heights in it will be in respect to the areoid, not to this datum. It is important to note that one cannot tell from inspecting a DEM if it was adjusted to be in respect to the areoid or not, so there is the potential of mixing up adjusted and unadjusted terrain models.

A.6.2 Post Spacing

Recall that `stereo` creates a point cloud file as its output and that you need to use `point2dem` on to create a GeoTIFF that you can use in other tools. The point cloud file is the result of taking the image-to-image matches (which were created from the kernel sizes you specified, and the subpixel versions of the same, if used) and projecting them out into space from the cameras, and arriving at a point in real world coordinates. Since `stereo` does this for every pixel in the input images, the *default* value that `point2dem` uses (if you don't specify anything explicitly) is the input image scale, because there's an 'answer' in the point cloud file for each pixel in the original image.

However, as you may suspect, this is probably not the best value to use because there really isn't that much 'information' in the data. The true 'resolution' of the output model is dependent on a whole bunch of things (like the kernel sizes you choose to use) but also can vary from place to place in the image depending on the texture.

The general 'rule of thumb' is to produce a terrain model that has a post spacing of about 3x the input image ground scale. This is based on the fact that it is nearly impossible to uniquely identify a single pixel correspondence between two images, but a 3x3 patch of pixels provides improved matching reliability. As you go to numerically larger post-spacings on output, you're averaging more point data (that is probably spatially correlated anyway) together.

So you can either use the `--dem-spacing` argument to `point2dem` to do that directly, or you can use your favorite averaging algorithm to reduce the `point2dem`-created model down to the scale you want.

If you attempt to derive science results from an ASP-produced terrain model with the default DEM spacing, expect serious questions from reviewers.

A.6.3 Using with LAS or CSV Clouds

The `point2dem` program can take as inputs point clouds in LAS and CSV formats. These differ from point clouds created by stereo by being, in general, not uniformly distributed. It is suggested that the user pick carefully the output resolution for such files (`--dem-spacing`). If the output DEM turns out to be sparse, the spacing could be increased, or one could experiment with increasing the value of `--search-radius-factor`, which will fill in small gaps in the output DEM by searching further for points in the input clouds.

It is expected that the input LAS files have spatial reference information such as WKT data. Otherwise it is assumed that the points are raw x, y, z values in meters in reference to the planet center.

Unless the output projection is explicitly set when invoking `point2dem`, the one from the first LAS file will be used.

For LAS or CSV clouds it is not possible to generate intersection error maps or ortho images.

For CSV point clouds, the option `--csv-format` must be set. If such a cloud contains easting, northing, and height above datum, the option `--csv-proj4` containing a PROJ.4 string needs to be specified to interpret this data (if the PROJ.4 string is set, it will be also used for output DEMs, unless `--t_srs` is specified).

Table A.6: Command-line options for point2dem

Options	Description
`--help|-h`	Display the help message.
`--nodata-value` *float(=-3.40282347e+38)*	Set the nodata value.
`--use-alpha`	Create images that have an alpha channel.
`--normalized|-n`	Also write a normalized version of the DEM (for debugging).
`--orthoimage`	Write an orthoimage based on the texture files passed in as inputs (after the point clouds).
`--errorimage`	Write an additional image whose values represent the triangulation ray intersection error in meters (the closest distance between the rays emanating from the two cameras corresponding to the same point on the ground).
`--output-prefix|-o` *output-prefix*	Specify the output prefix.
`--output-filetype|-t` *type(=tif)*	Specify the output file type.
`--x-offset` *float(=0)*	Add a horizontal offset to the DEM.
`--y-offset` *float(=0)*	Add a horizontal offset to the DEM.
`--z-offset` *float(=0)*	Add a vertical offset to the DEM.
`--rotation-order` *order(=xyz)*	Set the order of an Euler angle rotation applied to the 3D points prior to DEM rasterization.
`--phi-rotation` *float(=0)*	Set a rotation angle phi.
`--omega-rotation` *float(=0)*	Set a rotation angle omega.
`--kappa-rotation` *float(=0)*	Set a rotation angle kappa.

`--t_srs` *string*	Specify the output projection (PROJ.4 string). Can also be an URL or in WKT format, as for GDAL.
`--t_projwin` *xmin ymin xmax ymax*	The output DEM will have corners with these georeferenced coordinates.
`--datum` *string*	Set the datum. This will override the datum from the input images and also --t_srs, --semi-major-axis, and --semi-minor-axis. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).
`--reference-spheroid` *string*	This is identical to the datum option.
`--semi-major-axis` *float(=0)*	Explicitly set the datum semi-major axis in meters.
`--semi-minor-axis` *float(=0)*	Explicitly set the datum semi-minor axis in meters.
`--sinusoidal`	Save using a sinusoidal projection.
`--mercator`	Save using a Mercator projection.
`--transverse-mercator`	Save using a transverse Mercator projection.
`--orthographic`	Save using an orthographic projection.
`--stereographic`	Save using a stereographic projection.
`--oblique-stereographic`	Save using an oblique stereographic projection.
`--gnomonic`	Save using a gnomonic projection.
`--lambert-azimuthal`	Save using a Lambert azimuthal projection.
`--utm` *zone*	Save using a UTM projection with the given zone.
`--proj-lat` *float*	The center of projection latitude (if applicable).
`--proj-lon` *float*	The center of projection longitude (if applicable).
`--proj-scale` *float*	The projection scale (if applicable).
`--false-northing` *float*	The projection false northing (if applicable).
`--false-easting` *float*	The projection false easting (if applicable).
`--dem-spacing|-s` *float(=0)*	Set output DEM resolution (in target georeferenced units per pixel). If not specified, it will be computed automatically (except for LAS and CSV files). Multiple spacings can be set (in quotes) to generate multiple output files. This is the same as the --tr option.
`--search-radius-factor` *float(=0)*	Multiply this factor by `dem-spacing` to get the search radius. The DEM height at a given grid point is obtained as a weighted average of heights of all points in the cloud within search radius of the grid point, with the weights given by a Gaussian. Default search radius: max(`dem-spacing`, default_dem_spacing), so the default factor is about 1.
`--gaussian-sigma-factor` *float(=0)*	The value s to be used in the Gaussian $exp(-s * (x/grid\_size)^2)$ when computing the DEM. The default is -log(0.25) = 1.3863. A smaller value will result in a smoother terrain.

`--csv-format` *string*	Specify the format of input CSV files as a list of entries column_index:column_type (indices start from 1). Examples: '1:x 2:y 3:z' (a Cartesian coordinate system with origin at planet center is assumed, with the units being in meters), '5:lon 6:lat 7:radius_m' (longitude and latitude are in degrees, the radius is measured in meters from planet center), '3:lat 2:lon 1:height_above_datum', '1:easting 2:northing 3:height_above_datum' (need to set `--csv-proj4`; the height above datum is in meters). Can also use radius_km for column_type, when it is again measured from planet center.
`--csv-proj4` *string*	The PROJ.4 string to use to interpret the entries in input CSV files, if those files contain Easting and Northing fields. If not specified, --t_srs will be used.
`--rounding-error` *float(*=$1/2^{10}$=0.0009765625*)*	How much to round the output DEM and errors, in meters (more rounding means less precision but potentially smaller size on disk). The inverse of a power of 2 is suggested.
`--dem-hole-fill-len` *int(=0)*	Maximum dimensions of a hole in the output DEM to fill in, in pixels.
`--orthoimage-hole-fill-len` *int(=0)*	Maximum dimensions of a hole in the output orthoimage to fill in, in pixels. See also --orthoimage-hole-fill-extra-len.
`--orthoimage-hole-fill-extra-len` *int(=0)*	This value, in pixels, will make orthoimage hole filling more aggressive by first extrapolating the point cloud. A small value is suggested to avoid artifacts. Hole-filling also works better when less strict with outlier removal, such as in --remove-outliers-params, etc.
`--remove-outliers-params` *pct (float) factor (float) [default: 75.0 3.0]*	Outlier removal based on percentage. Points with triangulation error larger than pct-th percentile times factor will be removed as outliers.
`--max-valid-triangulation-error` *float(=0)*	Outlier removal based on threshold. Points with triangulation error larger than this (in meters) will be removed from the cloud.
`--max-output-size` *columns rows*	Creating of the DEM will be aborted if it is calculated to exceed this size in pixels.
`--median-filter-params` *window_size (int) threshold (double)*	If the point cloud height at the current point differs by more than the given threshold from the median of heights in the window of given size centered at the point, remove it as an outlier. Use for example 11 and 40.0.
`--erode-length` *length (int)*	Erode input point clouds by this many pixels at boundary (after outliers are removed, but before filling in holes).

`--filter` *string(=weighted_average)*	The filter to apply to the heights of the cloud points within a given circular neighborhood when gridding (its radius is controlled via --search-radius-factor). Options: weighted_average (default), min, max, mean, median, stddev, count (number of points), nmad (= 1.4826 * median(abs(X - median(X)))), n-pct (where n is a real value between 0 and 100, for example, 80-pct, meaning, 80th percentile). Except for the default, the name of the filter will be added to the obtained DEM file name, e.g., output-min-DEM.tif.
`--use-surface-sampling` *[default: false]*	Use the older algorithm, interpret the point cloud as a surface made up of triangles and sample it (prone to aliasing).
`--fsaa`	Oversampling amount to perform antialiasing. Obsolete, can be used only in conjunction with `--use-surface-sampling`.
`--threads` *int(=0)*	Select the number of processors (threads) to use.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.

A.7 point2mesh

The `point2mesh` tool produces a mesh surface that can be visualized in `osgviewer`, which is a standard 3D viewing application that is part of the open source OpenSceneGraph package. This viewer is bundled with Stereo Pipeline. [1]

Unlike DEMs, the 3D mesh is not meant to be used as a finished scientific product. Rather, it can be used for fast visualization to create a 3D view of the generated terrain.

The `point2mesh` program requires a point cloud file or a DEM, and an optional texture file. For example, it can be used with *output-prefix*`-PC.tif` and *output-prefix*`-L.tif`, as output by `stereo`, or otherwise with *output-prefix*`-DEM.tif` and *output-prefix*`-DRG.tif`, with the latter two output by `point2dem`.

When a texture file is not provided, a 1D texture is applied in the local Z direction that produces a rough rendition of a contour map. In either case, `point2mesh` will produce a *output-prefix*`.osgb` file that contains the 3D model in OpenSceneGraph format.

Two options for `osgviewer` bear pointing out: the `-l` flag indicates that synthetic lighting should be activated for the model, which can make it easier to see fine detail in the model by providing some real-time, interactive hillshading. The `-s` flag sets the sub-sampling rate, and dictates the degree to which the 3D model should be simplified. For 3D reconstructions, this can be essential for producing a model that can fit in memory. The default value is 10, meaning every 10th point is used in the X and Y directions. In other words that mean only $1/10^2$ of the points are being used to create the model. Adjust this sampling rate according to how much detail is desired, but remember that large models will impact the frame rate of the 3D viewer and affect performance.

Examples:

```
point2mesh -s 2 -l output-prefix-PC.tif output-prefix-L.tif
point2mesh -s 2 -l output-prefix-DEM.tif output-prefix-DRG.tif
```

[1] The full OpenSceneGraph package can be installed separately from http://www.openscenegraph.org/.

To view the resulting *output-prefix*.osgb file use `osgviewer`.

Fullscreen:
```
> osgviewer output-prefix.osgb
```

In a window:
```
> osgviewer output-prefix.osgb --window 50 50 1000 1000
```

Be sure to turn on lightning as soon as the model is loaded, by pressing on "L". In addition, the keys T, W, and F can be used to toggle on and off texture, wireframe, and full-screen modes. The left, middle, and right mouse buttons control rotation, panning, and zooming of the model.

The `-t` output file type option can also take the *obj* value, when it will write the mesh in the Wavefront OBJ format file (*output-prefix.obj*) that can be read into various 3D graphics programs.

Table A.7: Command-line options for point2mesh

Options	Description
`--help|-h`	Display the help message.
`--simplify-mesh` *float*	Run OSG Simplifier on mesh, 1.0 = 100%.
`--smooth-mesh`	Run OSG Smoother on mesh
`--use-delaunay`	Uses the delaunay triangulator to create a surface from the point cloud. This is not recommended for point clouds with noise issues.
`--step|-s` *integer(=10)*	Sampling step size for the mesher.
`--input-file` *pointcloud-file*	Explicitly specify the input file.
`--output-prefix|-o` *output-prefix*	Specify the output prefix.
`--texture-file` *texture-file*	Explicitly specify the texture file.
`--output-filetype|-t` *type(=osgb)*	Specify the output file type.
`--enable-lighting|-l`	Enables shades and lighting on the mesh.
`--center`	Center the model around the origin. Use this option if you are experiencing numerical precision issues.

A.8 dem_mosaic

The program `dem_mosaic` takes as input a list of DEM files, optionally erodes pixels at the DEM boundaries, and creates a mosaic. By default, it blends the DEMs where they overlap.

Usage:

```
dem_mosaic [options] <dem files or -l dem_files_list.txt> -o output_file_prefix
```

The input DEMs can either be set on the command line, or if too many, they can be listed in a text file (one per line) and that file can be passed to the tool.

The output mosaic is written as non-overlapping tiles with desired tile size, with the size set either in pixels or in georeferenced (projected) units. The default tile size is large enough that normally the entire mosaic is saved as one tile, in the format output_file_prefix-tile-0.tif. Alternatively, one can pass to the `-o` option an output file name ending in .tif. Then the mosaic will be written with this exact name, without appending tile-0.tif. (This will fail if the tool decides there is a need for more than one tile.)

Individual tiles can be saved via the `--tile-index` option (the tool displays the total number of tiles when it is being run). As such, separate processes can be invoked for individual tiles for increased robustness and perhaps speed.

The output mosaic tiles will be named <output prefix>-tile-<tile index>.tif, where <output prefix> is an arbitrary string. For example, if it is set to `results/output`, all the tiles will be in the `results` directory.

By the default, the output mosaicked DEM will use the same grid size and projection as the first input DEM. These can be changed via the `--tr` and `--t_srs` options.

The default behavior is to blend the DEMs everywhere. If the option `--priority-blending-length` *`integer`* is invoked, the blending behavior will be different. At any location, the pixel value of the DEM earliest in the list present at this location will be kept, unless closer to the boundary of that DEM than this blending length (measured in input DEM pixels), only in the latter case blending will happen. This mode is useful when blending several high-resolution "foreground" DEMs covering small regions with larger "background" DEMs covering a larger extent. Then, the pixels from the high-resolution DEMs are more desirable, yet at their boundary these DEMs should blend into the background.

To obtain smoother blending when the input DEMs are quite different at the boundary, one can increase `--weights-blur-sigma` and `--weights-exponent`. The latter will result in weights growing slower earlier and faster later. Some experimentation may be necessary, helped for example by examining the weights used in blending; they can be written out with `--save-dem-weight` *`integer`*.

Instead of blending, `dem_mosaic` can compute the image of first, last, minimum, maximum, mean, standard deviation, median, and count of all encountered valid DEM heights at output grid points. For the "first" and "last" operations, the order in which DEMs were passed in is used. With any of these options, the tile names will be adjusted accordingly. It is important to note that with these options blending will not happen, since it is explicitly requested that particular values of the input DEMs be used.

If the number of input DEMs is very large, the tool can fail as the operating system may refuse to load all DEMs. In that case, it is suggested to use the parameter `--tile-size` to break up the output DEM into several large tiles, and to invoke the tool for each of the output tiles with the option `--tile-index`. Later, `dem_mosaic` can be invoked again to merge these tiles into a single DEM.

If the DEMs have reasonably regular boundaries and no holes, smoother blending may be obtained by using `--use-centerline-weights`.

Example 1 (erode 3 pixels from input DEMs and blend them):

```
dem_mosaic --erode-length 3 dem1.tif dem2.tif -o blended
```

Example 2 (read the DEMs from a list, and apply priority blending):

```
echo dem1.tif dem2.tif > imagelist.txt
dem_mosaic -l imagelist.txt --priority-blending-length 14 -o priority_blended
```

Example 3 (Find the mean DEM, no blending is used):

```
dem_mosaic -l imagelist.txt --mean -o mosaic
```

Example 4 (write with the exact output name, without using the tile-0.tif extension):

```
dem_mosaic dem1.tif dem2.tif -o blended.tif
```

Table A.8: Command-line options for dem_mosaic

Options	Description
`--help|-h`	Display the help message.
`-l | --dem-list-file` *string*	Text file listing the DEM files to mosaic, one per line.
`-o | --output-prefix` *string*	Specify the output prefix. One or more tiles will be written with this prefix. Alternatively, an exact output file can be specified, with a .tif extension.
`--tile-size` *integer(=1000000)*	The maximum size of output DEM tile files to write, in pixels.
`--tile-index` *integer*	The index of the tile to save (starting from zero). When this program is invoked, it will print out how many tiles are there. Default: save all tiles.
`--tile-list` *string*	List of tile indices (in quotes) to save. A tile index starts from 0.
`--erode-length` *integer(=0)*	Maximum dimensions of a hole in the output DEM to fill in, in pixels.
`--priority-blending-length` *integer(=0)*	If positive, keep unmodified values from the earliest available DEM at the current location except a band this wide measured in pixels around its boundary where blending will happen.
`--hole-fill-length` *integer(=0)*	Erode input DEMs by this many pixels at boundary before mosaicking them.
`--tr` *double*	Output DEM resolution in target georeferenced units per pixel. Default: use the same resolution as the first DEM to be mosaicked.
`--t_srs` *string*	Specify the output projection (PROJ.4 string). Default: use the one from the first DEM to be mosaicked.
`--t_projwin` *xmin ymin xmax ymax*	Limit the mosaic to this region, with the corners given in georeferenced coordinates (xmin ymin xmax ymax). Max is exclusive.
`--first`	Keep the first encountered DEM value (in the input order).
`--last`	Keep the last encountered DEM value (in the input order).
`--min`	Keep the smallest encountered DEM value.

`--max`	Keep the largest encountered DEM value.
`--mean`	Find the mean DEM value.
`--stddev`	Find the standard deviation of DEM values.
`--median`	Find the median DEM value (this can be memory-intensive, fewer threads are suggested).
`--nmad`	Find the normalized median absolute deviation DEM value (this can be memory-intensive, fewer threads are suggested).
`--count`	Each pixel is set to the number of valid DEM heights at that pixel.
`--georef-tile-size` *double*	Set the tile size in georeferenced (projected) units (e.g., degrees or meters).
`--output-nodata-value` *double*	No-data value to use on output. Default: use the one from the first DEM to be mosaicked.
`--ot` *string(=Float32)*	Output data type. Supported types: Byte, UInt16, Int16, UInt32, Int32, Float32. If the output type is a kind of integer, values are rounded and then clamped to the limits of that type.
`--weights-blur-sigma` *integer (=5)*	The standard deviation of the Gaussian used to blur the weights. Higher value results in smoother weights and blending. Set to 0 to not use blurring.
`--weights-exponent` *float (=2.0)*	The weights used to blend the DEMs should increase away from the boundary as a power with this exponent. Higher values will result in smoother but faster-growing weights.
`--use-centerline-weights`	Compute weights based on a DEM centerline algorithm. Produces smoother weights if the input DEMs don't have holes or complicated boundary.
`--dem-blur-sigma` *integer (=0)*	Blur the final DEM using a Gaussian with this value of sigma. Default: No blur.
`--extra-crop-length` *integer(=200)*	Crop the DEMs this far from the current tile (measured in pixels) before blending them (a small value may result in artifacts).
`--nodata-threshold` *float*	Values no larger than this number will be interpreted as no-data.
`--force-projwin`	Make the output mosaic fill precisely the specified projwin, by padding it if necessary and aligning the output grid to the region.
`--save-dem-weight` *integer*	Save the weight image that tracks how much the input DEM with given index contributed to the output mosaic at each pixel (smallest index is 0).
`--save-index-map`	For each output pixel, save the index of the input DEM it came from (applicable only for --first, --last, --min, --max, --median, and --nmad). A text file with the index assigned to each input DEM is saved as well.
`--threads` *integer(=4)*	Set the number of threads to use.

A.9 image_mosaic

The program `image_mosaic` aligns multiple input images into a single output image. Currently it only supports a horizontal sequence of images such as scanned Corona images. An example of using this tool is in section 11.17.

Usage:

```
image_mosaic [options] <images> -o output_file_path [options]
```

Table A.9: Command-line options for image_mosaic

Options	Description
`--t_orientation` *string(=horizontal)*	Specify the image layout. Currently only supports horizontal.
`--reverse`	Mosaic the images in reverse order.
`--rotate`	After mosaicking, rotate the image by 180 degrees around its center.
`--rotate-90`	After mosaicking, rotate the image by 90 degrees clockwise around its center.
`--rotate-90-ccw`	After mosaicking, rotate the image by 90 degrees counter-clockwise around its center.
`--use-affine-transform`	Solve for full affine transforms between segments instead of a simpler rotate+translate transform.
`-o | --output-image` *string*	Specify the output file path. Required.
`--overlap-width` *integer(=2000)*	The width of the expected overlap region in the images, in pixels.
`--blend-radius` *integer(=0)*	The width in pixels over which blending is performed. Default is calculated based on the overlap width.
`--band` *integer(=0)*	Specify a band to use for multi-channel images.
`--ot` *string(=Float32)*	Output data type. Supported types: Byte, UInt16, Int16, UInt32, Int32, Float32. If the output type is a kind of integer, values are rounded and then clamped to the limits of that type.
`--input-nodata-value` *double*	Override the input nodata value.
`--output-nodata-value` *double*	Specify the output nodata value.
`--ip-per-tile` *integer*	How many interest points to detect in each 1024^2 image tile (default: automatic determination).
`--output-prefix` *string*	If specified, save here the interest point matches used in mosaicking.
`--tile-size` *integer(=256, 256)*	The size of image tiles used for processing. The amount of image blending is limited by the tile size, so this will be increased automatically if it is too small for the overlap width.
`--help|-h`	Display the help message.

A.10 dem_geoid

This tool takes as input a DEM whose height values are relative to the datum ellipsoid, and adjusts those values to be relative to the equipotential surface of the planet (geoid on Earth, and areoid on Mars). The program can also apply the reverse of this adjustment. The adjustment simply subtracts from the DEM height the geoid height (correcting, if need be, for differences in dimensions between the DEM and geoid datum ellipsoids).

Three geoids and one areoid are supported. The Earth geoids are: EGM96 and EGM2008, relative to the WGS84 datum ellipsoid (http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm96/egm96.html, http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm2008/egm08_wgs84.html) and NAVD88, relative to the NAD83 datum ellipsoid (http://www.ngs.noaa.gov/GEOID/GEOID09/).

The Mars areoid is MOLA MEGDR (http://geo.pds.nasa.gov/missions/mgs/megdr.html). When importing it into ASP, we adjusted the areoid height values to be relative to the IAU reference spheroid for Mars of radius 3,396,190 m. The areoid at that source was relative to the Mars radius of 3,396,000 m. Yet `dem_geoid` can adjust correctly Mars DEMs created in respect to either spheroid.

Example: Go from a DEM in respect to the WGS84 datum to one in respect to the EGM2008 geoid:

```
dem_geoid input-DEM.tif --geoid egm2008
```

This program will write a new image file with the suffix `*-adj.tif`.

Table A.10: Command-line options for dem_geoid

Options	Description
`--help|-h`	Display the help message.
`--nodata-value` *float(=-32768)*	The value of no-data pixels, unless specified in the DEM.
`--geoid` *string*	Specify the geoid to use for the given datum. For WGS84 use EGM96 or EGM2008. Default: EGM96. For Mars use MOLA or leave blank. For NAD83 use NAVD88 or leave blank. When not specified it will be auto-detected.
`--output-prefix|-o` *filename*	Specify the output file prefix.
`--double`	Output using double precision (64 bit) instead of float (32 bit).
`--reverse-adjustment`	Go from DEM relative to the geoid/areoid to DEM relative to the datum ellipsoid.

A.11 dg_mosaic

This tool can be used when processing Digital Globe Imagery (chapter 4). A Digital Globe satellite may take a picture, and then split it into several images and corresponding camera XML files. `dg_mosaic` will mosaic these images into a single file, and create the appropriate combined camera XML file.

Digital Globe camera files contain, in addition to the original camera models, their RPC approximations (section 11.12). `dg_mosaic` outputs both types of combined models. The combined RPC model can be used to map-project the mosaicked images with the goal of computing stereo from them (section 5.1.7).

The tool needs to be applied twice, for both the left and right image sets.

`dg_mosaic` can also reduce the image resolution while creating the mosaics (with the camera files modified

accordingly).

Some older (2009 or earlier) Digital Globe images may exhibit seams upon mosaicking due to inconsistent image and camera information. The `--fix-seams` switch can be used to rectify this problem. Its effect should be minimal if such inconsistencies are not present.

Table A.11: Command-line options for dg_mosaic

Options	Description
`--help|-h`	Display the help message.
`--target-resolution`	Choose the output resolution in meters per pixel on the ground (note that a coarse resolution may result in aliasing).
`--reduce-percent` *integer(=100)*	Render a reduced resolution image and XML based on this percentage. This can result in aliasing artifacts.
`--skip-rpc-gen` *[default: false]*	Skip RPC model generation.
`--rpc-penalty-weight` *float(=0.1)*	The weight to use to penalize higher order RPC coefficients when generating the combined RPC model. Higher penalty weight results in smaller such coefficients.
`--output-prefix` *string*	The prefix for the output .tif and .xml files.
`--band` *integer*	Which band to use (for multi-spectral images).
`--input-nodata-value` *float*	Nodata value to use on input; input pixel values less than or equal to this are considered invalid.
`--output-nodata-value` *float*	Nodata value to use on output.
`--ot` *string(=Float32)*	Output data type. Supported types: Byte, UInt16, Int16, UInt32, Int32, Float32. If the output type is a kind of integer, values are rounded and then clamped to the limits of that type.
`--fix-seams`	Fix seams in the output mosaic due to inconsistencies between image and camera data using interest point matching.
`--ignore-inconsistencies`	Ignore the fact that some of the files to be mosaicked have inconsistent EPH/ATT values. Do this at your own risk.
`--preview`	Render a small 8 bit png of the input for preview.
`--` *dry-run|-n*	Make calculations, but just print out the commands.

A.12 mapproject

The tool `mapproject` is used to orthorectify (map-project) a camera image onto a DEM or datum. (ASP is able to use map-projected images to run stereo, see section 5.1.7.)

The `mapproject` program can be run using multiple processes and can be distributed over multiple machines. This is particularly useful for ISIS cameras, as in that case any single process must use only one thread due to the limitations of ISIS. The tool splits the image up into tiles, farms the tiles out to sub-processes, and then merges the tiles into the requested output image. If your image is small, smaller tiles can be used as well to start more simultaneous processes (parameter `--tile-size`).

Examples:

Map-project a .cub file (it has both image and camera information):

```
mapproject -t isis DEM.tif image.cub output.tif --ppd 256
```

Map-project an image file with associated .xml camera file:

```
mapproject -t rpc DEM.tif image.tif image.xml output.tif --mpp 20
```

Mapproject onto a datum rather than a DEM:

```
mapproject WGS84 image.tif image.xml output.tif --mpp 10
```

The first argument can either be a path to a DEM file or the name of a standard datum. Valid datum names include WGS84, NAD83, NAD27, D_MOON, D_MARS, and MOLA.

It is very important to pick a good value for the grid size parameter, given by `--mpp`, `--ppd`, or `--tr`. Ideally it should be very close to the known image resolution as measured on the ground (in degree or meter units, depending on the projection).

If the imagery is from Digital Globe, both the exact DG model from the XML file can be used for map-projection (`-t dg`) and its RPC approximation (`-t rpc`). The former is more accurate but much smaller.

Usage:

```
mapproject [options] <dem> <camera-image> <camera-model> <output-image>
```

Table A.12: Command-line options for mapproject

Options	Description
`--help|-h`	Display the help message.
`--nodata-value` *float(=-32768)*	No-data value to use unless specified in the input image.
`--t_srs`	Specify the output projection (PROJ.4 string). If not provided, use the one from the DEM.
`--mpp` *float*	Set the output file resolution in meters per pixel.
`--ppd` *float*	Set the output file resolution in pixels per degree.
`--tr` *float*	Set the output file resolution in target georeferenced units per pixel.
`--datum-offset` *float*	When projecting to a datum instead of a DEM, add this elevation offset to the datum.

`--session-type|-t pinhole|isis|rpc`	Select the stereo session type to use for processing. Choose 'rpc' if it is desired to later do stereo with the 'dg' session.
`--t_projwin` *xmin ymin xmax ymax*	Limit the map-projected image to this region, with the corners given in georeferenced coordinates (xmin ymin xmax ymax). Max is exclusive.
`--t_pixelwin` *xmin ymin xmax ymax*	Limit the map-projected image to this region, with the corners given in pixels (xmin ymin xmax ymax). Max is exclusive.
`--bundle-adjust-prefix` *string*	Use the camera adjustment obtained by previously running bundle_adjust with this output prefix.
`--ot` *string(=Float32)*	Output data type, when the input is single channel. Supported types: Byte, UInt16, Int16, UInt32, Int32, Float32. If the output type is a kind of integer, values are rounded and then clamped to the limits of that type. This option will be ignored for multi-channel images, when the output type is set to be the same as the input type.
`--nearest-neighbor`	Use nearest neighbor interpolation instead of bicubic interpolation.
`--mo` *string*	Write metadata to the output file. Provide as a string in quotes if more than one item, separated by a space, such as 'VAR1=VALUE1 VAR2=VALUE2'. Neither the variable names nor the values should contain spaces.
`--num-processes`	Number of parallel processes to use (default program chooses).
`--nodes-list`	List of available computing nodes.
`--tile-size`	Size of square tiles to break processing up into.
`--suppress-output`	Suppress output from sub-processes.
`--threads` *int(=0)*	Select the number of processors (threads) to use.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.

A.13 cam2rpc

This tool is used to generate an approximate RPC model for any camera model supported by ASP, in a given longitude-latitude-height region for a given datum, or for a terrain covered by a given DEM. If `--save-tif-image` is specified, the image portion corresponding to the RPC model will be saved in the TIF format.

The obtained RPC models and images can be used with `stereo` (when the latter is invoked with `--session-type rpc` and the correct datum is specified via `--datum`). These can also be passed to the third-party `S2P` and `SETSM` stereo software, though both of these packages work for Earth only.

The accuracy of RPC models generally degrades if expected to cover very large regions. Hence, they can be used piecewise, and the obtained terrain models from ASP can be then mosaicked together using `dem_mosaic`.

Example for ISIS cub cameras for Mars:

```
cam2rpc input.cub output.xml --session-type isis --datum D_MARS --save-tif-image \
  --height-range -10000 -9000 --lon-lat-range 141.50 34.43 141.61 34.15          \
  --num-samples 40 --penalty-weight 0.03 --gsd 1
```

Example for pinhole cameras, where instead of sampling a lon-lat-height box, values from a DEM are used.

```
cam2rpc input.tif input.tsai output.xml --session-type nadirpinhole             \
  --dem-file DEM.tif --save-tif-image --image-crop-box 90 70 5511 3675
```

Here we have constrained the RPC camera model and output image to not go beyond a given bounding box.

Usage:

```
cam2rpc [options] <camera-image> <camera-model> <output-rpc>
```

Table A.13: Command-line options for cam2rpc

Option	Description
`--datum` *string*	Use this datum to interpret the heights. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).
`--semi-major-axis` *double*	Explicitly set the datum semi-major axis in meters.
`--semi-minor-axis` *double*	Explicitly set the datum semi-minor axis in meters.
`--t_srs` *string*	Specify a projection (PROJ.4 string) instead of the datum. Can also be an URL or in WKT format, as for GDAL.

`--dem-file` *string*	Instead of using a datum and a longitude-latitude-height box, sample the surface of this DEM.
`--lon-lat-range arg (=0 0 0 0)`	The longitude-latitude range in which to compute the RPC model. Specify in the format: lon_min lat_min lon_max lat_max.
`--height-range arg (=0 0)`	Minimum and maximum heights above the datum in which to compute the RPC model.
`--num-samples arg (=40)`	How many samples to use in each direction in the longitude-latitude-height range.
`--penalty-weight arg (=0.03)`	A higher penalty weight will result in smaller higher-order RPC coefficients.
`--save-tif-image`	Save a TIF version of the input image that approximately corresponds to the input longitude-latitude-height range and which can be used for stereo together with the RPC model.
`--input-nodata-value arg`	Set the image input nodata value.
`--output-nodata-value arg`	Set the image output nodata value.
`-t | --session-type` *string*	Select the input camera model type. Normally this is auto-detected, but may need to be specified if the input camera model is in XML format. Options: pinhole isis rpc dg spot5 aster opticalbar.
`--bundle-adjust-prefix` *string*	Use the camera adjustment obtained by previously running bundle_adjust with this output prefix.
`--image-crop-box arg (=0 0 0 0)`	The output image and RPC model should not exceed this box, specified in input image pixels as minx miny widx widy.
`--no-crop`	Try to create an RPC model over the entire input image, even if the input longitude-latitude-height box covers just a small portion of it. Not recommended.
`--skip-computing-rpc`	Skip computing the RPC model.
`--gsd arg (=-1)`	Expected resolution on the ground, in meters. This is needed for SETSM.
`--threads arg (=0)`	Select the number of processors (threads) to use.
`--tile-size arg (=256, 256)`	Image tile size used for multi-threaded processing.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress` *string* `(=LZW)`	TIFF Compression method. [None, LZW, Deflate, Packbits]
`-v | --version`	Display the version of software.
`-h | --help`	Display this help message.

A.14 disparitydebug

The `disparitydebug` program produces output images for debugging disparity images created from `stereo`. The `stereo` tool produces several different versions of the disparity map; the most important ending with extensions `*-D.tif` and `*-F.tif`. (see Appendix C for more information.) These raw disparity map files can be useful for debugging because they contain raw disparity values as measured by the correlator; however they cannot be directly visualized or opened in a conventional image browser. The `disparitydebug` tool converts a single disparity map file into two normalized TIFF image files (`*-H.tif` and `*-V.tif`, containing the horizontal and vertical, or line and sample, components of disparity, respectively) that can be viewed using any image display program.

The `disparitydebug` program will also print out the range of disparity values in a disparity map, that can serve as useful summary statistics when tuning the search range settings in the `stereo.default` file.

If the input images are map-projected (georeferenced), the outputs of `disparitydebug` will also be georeferenced.

Table A.14: Command-line options for disparitydebug

Options	Description
`--help|-h`	Display the help message
`--input-file` *filename*	Explicitly specify the input file
`--output-prefix|-o` *filename*	Specify the output file prefix
`--output-filetype|-t` *type(=tif)*	Specify the output file type
`--float-pixels`	Save the resulting debug images as 32 bit floating point files (if supported by the selected file type)

A.15 orbitviz

Produces a Google Earth Keyhole Markup Language (KML) file useful for visualizing camera positions. The input for this tool is one or more images and camera files.

Usage:

```
orbitviz [options] <input images and cameras>
```

Table A.15: Command-line options for orbitviz

Options	Description
`--help|-h`	Display the help message
`--output|-o` *filename(=orbit.kml)*	The output kml file that will be written.
`--linescan-line` *integer(=1)*	Get the camera position at this pixel line.
`--linescan-sample` *integer(=1)*	Get the camera position at this pixel sample.
`--model-scale | -s` *float(=1)*	Scale the size of the coordinate axes by this amount. Ex: To scale axis sizes up to Earth size, use 3.66.
`--use-path-to-dae-model|-u` *fullpath*	Use this dae model to represent camera location. *Google Sketch up can create these.*

`-r | --reference-spheroid` *string=WGS_1984*	Use this reference spheroid (datum). Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).
`-t | --session-type` *string*	Select the stereo session type to use for processing. Options: pinhole isis spot5 dg aster opticalbar.
`--load-camera-solve`	Use a specialized display for showing the results of the `camera_solve` tool. When using this option, only pass in the path to the `camera_solve` output folder as a positional argument. Green lines drawn between the camera positions indicate a successful interest point match between those two images.
`--hide-labels`	Hide image names unless the camera is highlighted.
`--bundle-adjust-prefix` *string*	Use the camera adjustment obtained by previously running bundle_adjust with this output prefix.
`--write-csv`	Write a csv file with the orbital data.

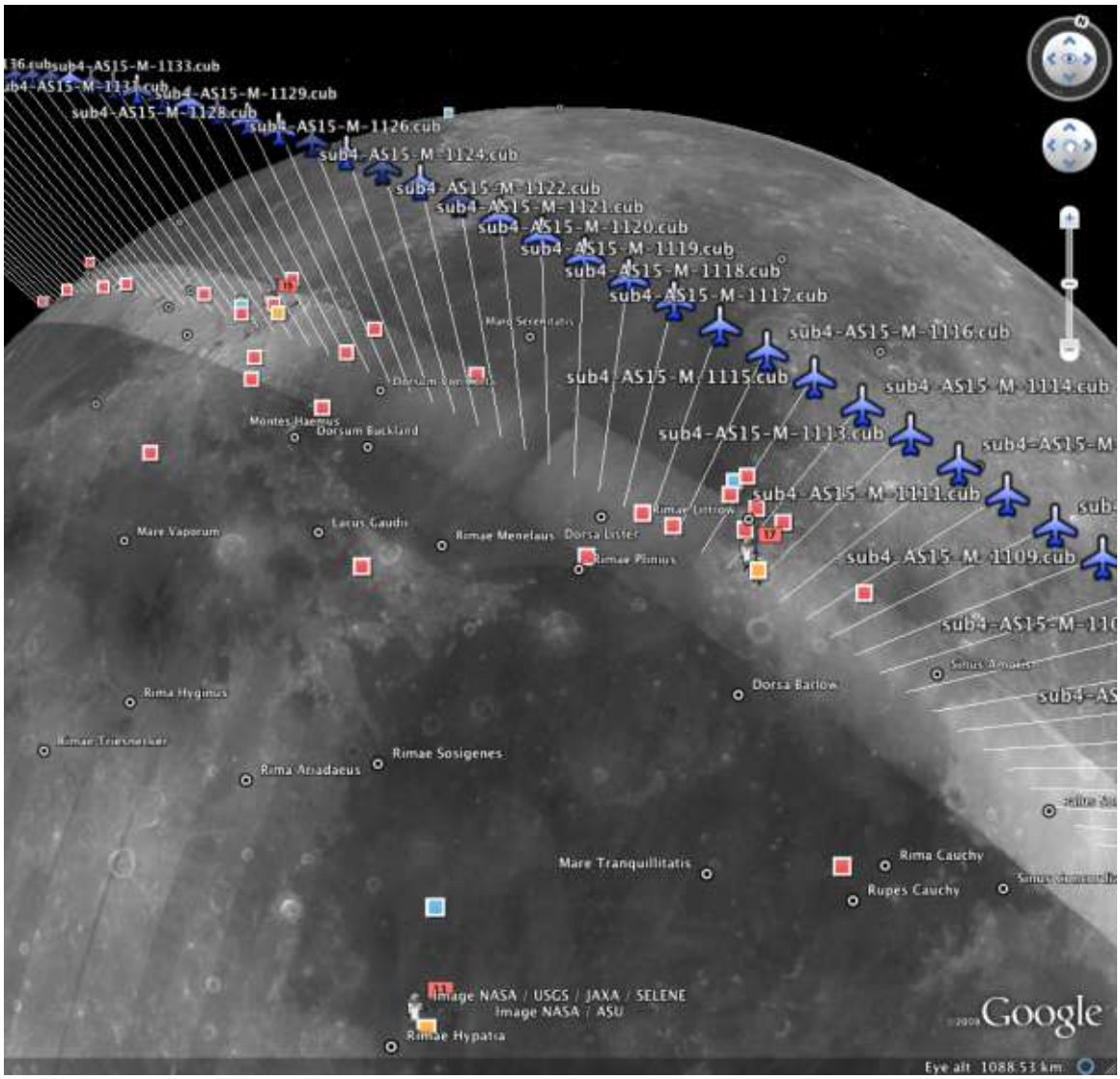


Figure A.2: Example of a KML visualization produced with `orbitviz` depicting camera locations for the Apollo 15 Metric Camera during orbit 33 of the Apollo command module.

A.16 camera_footprint

The tool `camera_footprint` computes what the footprint of in image would be if map projected on to a provided datum or DEM and prints it to the screen. If a KML output path is provided it will also create a KML file containing the footprint. The KML will show a box with an X pattern showing the points ASP used to compute the footprint. This tool can be useful for debugging camera orientations or getting a quick overview of where images are located.

Usage:

```
camera_footprint [options] <camera-image> <camera-model>
```

Table A.16: Command-line options for camera_footprint

Options	Description
`--help|-h`	Display the help message.
`--dem-file` *string*	Intersect with this DEM instead of a datum.
`--datum` *string*	Use this datum to interpret the heights. Options are: WGS_1984, D_MOON, D_MARS, and MOLA.
`--t_srs`	Specify the georeference projection (PROJ.4 string).
`--session-type|-t`	Select the stereo session type to use for processing. Normally this is autodetected.
`--bundle-adjust-prefix` *string*	Use the camera adjustment obtained by previously running bundle_adjust with this output prefix.
`--output-kml` *string*	Write an output KML file at this location.
`--quick`	Use a faster but less accurate computation.

A.17 cam2map4stereo.py

This program takes similar arguments as the ISIS3 `cam2map` program, but takes two input images. With no arguments, the program determines the minimum overlap of the two images, and the worst common resolution, and then map-projects the two images to this identical area and resolution.

The detailed reasons for doing this, and a manual step-by-step walkthrough of what `cam2map4stereo.py` does is provided in the discussion on aligning images on page 16.

The `cam2map4stereo.py` is also useful for selecting a subsection and/or reduced resolution portion of the full image. You can inspect a raw camera geometry image in qview after you have run `spiceinit` on it, select the latitude and longitude ranges, and then use `cam2map4stereo.py`'s `--lat`, `--lon`, and optionally `--resolution` options to pick out just the part you want.

Use the `--dry-run` option the first few times to get an idea of what `cam2map4stereo.py` does for you.

Table A.17: Command-line options for cam2map4stereo.py

Options	Description
`--help|-h`	Display the help message.
`--manual`	Read the manual.
`--map=`*`MAP`*`|-m `*`MAP`*	The mapfile to use for `cam2map`.
`--pixres=`*`PIXRES`*`|-p `*`PIXRES`*	The pixel resolution mode to use for `cam2map`.
`--resolution=`*`RESOLUTION`*`|-r `*`RESOLUTION`*	Resolution of the final map for `cam2map`.
`--interp=`*`INTERP`*`|-i `*`INTERP`*	Pixel interpolation scheme for `cam2map`.
`--lat=`*`LAT`*`|-a `*`LAT`*	Latitude range for `cam2map`, where `LAT` is of the form *min:max*. So to specify a latitude range between -5 and 10 degrees, it would look like `--lat=-5:10`.
`--lon=`*`LON`*`|-o `*`LON`*	Longitude range for `cam2map`, where `LON` is of the form *min:max*. So to specify a longitude range between 45 and 47 degrees, it would look like `--lon=40:47`.
`--dry-run|-n`	Make calculations, and print the `cam2map` command that would be executed, but don't actually run it.
`--prefix`	Make all output files use this prefix. Default: no prefix.
`--suffix|-s`	Suffix that gets inserted in the output file names, defaults to 'map'.

A.18 pansharp

This tool reads in a high resolution grayscale file and a low resolution RGB file and produces a high resolution RGB file. The output image will be at the resolution of the grayscale image and will cover the region where the two images overlap. Both images must have georeferencing information. This can either be projection information in the image metadata or it can be a separate Worldview format XML camera file containing four ground control points (if using the tool with Digital Globe images).

Usage:

```
pansharp [options] <grayscale image file> <color image file> <output image file>
```

Table A.18: Command-line options for pansharp

Options	Description
`--help`	Display the help message.
`--min-value`	Manually specify the bottom of the input data range.
`--max-value`	Manually specify the top of the input data range.
`--gray-xml`	Look for georeference data here if not present in the grayscale image.
`--color-xml`	Look for georeference data here if not present in the RGB image.
`--nodata-value`	The nodata value to use for the output RGB file.

A.19 datum_convert

This tool is used to convert a DEM from one datum to another. For example, a UTM zone 10 DEM with an NAD27 datum can be converted to a UTM zone 10 DEM with a WGS84 datum. This tool does not convert between projections, another program such as `gdalwarp` (included with ASP) or ASP's `dem_mosaic` should be used for that. `datum_convert` performs horizontal conversion; vertical conversion is only provided for the limited case of conversions between datums defined only by the +a and +b terms (such as our D_MARS and MOLA datums). The underlying Proj.4 library does have some support for vertical datums (see https://github.com/OSGeo/proj.4/wiki/VerticalDatums) so a motivated user may be able to apply them successfully. If you do, let us know what steps you took so we can add them to the manual! ASP ships with some vertical datum grid files in the ASP/share/proj folder but more can be found on the Internet. Whenever you perform datum conversions be careful; the Proj.4 library tends to fail silently by performing an identity transform on the input data. If your output data exactly matches your input data this means that something has probably gone wrong.

The tool will try to automatically find datum information from the input file but the input datum can be manually specified if the information in the file is missing or incorrect. Be aware that if the `--keep-bounds` option is not set there may be noticeable changes in the image data just from re-interpolating to the new projected space grid. In the case of sparsely sampled input images this effect can be much larger than the changes resulting from the actual datum transformation.

Intuitively, the input and output DEMs should correspond to the same point cloud in 3D space up to the interpolation errors required to perform the conversion. In practice datum conversion is a complex task which may need to account for things like shifting tectonic plates over time. ASP's implementation is based on Proj.4 and the HTDPGrids extension (https://github.com/OSGeo/proj.4/wiki/HTDPGrids). Datum support in Proj.4 is not robust even with the extension so if it is critical that you have a very accurate conversion we recommend that you attempt to verify results obtained using `datum_convert` with another

conversion method.

This tool requires the gdal and numpy Python packages to run. One way to get these is to install the ASP Python tools, described at the end of section 4.5.

Usage:

```
datum_convert [options] <input dem> <output dem>
```

Table A.19: Command-line options for datum_convert

Options	Description
`--help`	Display the help message.
`--show-all-datums`	Print out all the datum names which are recognized.
`--output-datum` *string*	The datum to convert to. Supported options include: WGS_1984, NAD83, WGS72, and NAD27.
`--input-datum` *string*	Override the datum of the input file. Supports the same options as –output-datum.
`--output-datum-year` *default: 2000.0*	Specify the exact date of the output datum in floating point format ex: 2003.4.
`--input-datum-year` *default: 2000.0*	As –output-datum-year, but for the input file.
`--t_srs` *string*	Specify the output datum via the PROJ.4 string.
`--keep-bounds`	Don't recompute the projected space boundary. This can help reduce changes caused by interpolation.
`--nodata-value`	The value of no-data pixels, unless specified in the DEM.
`--double`	Create float64 instead of float32 output files.
`--show-grid-calc`	Don't hide the shift grid creation output.
`--debug-mode`	Print the converted lon/lat/alt coordinates for each pixel. Only useful for investigating exact change that is happening.
`--grid-size-lon`	Specify the number of columns in the grid shift file.
`--grid-size-lat`	Specify the number of rows in the grid shift file.
`--keep-working-files`	Don't delete intermediate files.

A.20 point2las

This tool can be used to convert point clouds generated by ASP to the public LAS format for interchange of 3-dimensional point cloud data.

If the input cloud has a datum, or the `--datum` option is specified, then the output LAS file will be created in respect to this datum. Otherwise raw x, y, z values will be saved.

Table A.20: Command-line options for point2las

Options	Description
`--help|-h`	Display the help message.
`--datum`	Create a geo-referenced LAS file in respect to this datum. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).
`--reference-spheroid` *string*	This is identical to the datum option.

`--t_srs` *string*	Specify the output projection (PROJ.4 string).
`--compressed`	Compress using laszip.
`--output-prefix|-o` *filename*	Specify the output file prefix.
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default defined in the program or in the .vwrc file.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.

A.21 pc_align

This tool can be used to align two point clouds. The algorithms employed are one of the several flavors of Iterative Closest Point (ICP), based on the `libpointmatcher` library [124]

https://github.com/ethz-asl/libpointmatcher

It also implements the Fast Global Registration algorithm

https://github.com/IntelVCL/FastGlobalRegistration

In addition, it supports feature-based alignment (terrains are hillshaded and interest point matches are found among them), and alignment using least squares. It can handle a scale change in addition to rotations and translations. For joint alignment of more than two clouds, the related tool `n_align` can be used (section A.22).

Usage:

```
pc_align --max-displacement <float> [other options] <reference cloud> <source cloud> \
  -o <output prefix>}
```

An example of using this tool is in section 5.2.5.

Several important things need to be kept in mind if `pc_align` is to be used successfully and give accurate results, as described below.

A.21.1 The input point clouds

Due to the nature of ICP, the first input point cloud, that is, the reference (fixed) cloud, should be denser than the second, source (movable) point cloud, to get the most accurate results. This is not a serious restriction, as one can perform the alignment this way and then simply invert the obtained transform if desired (`pc_align` outputs both the direct and inverse transform, and can output the reference point cloud transformed to match the source and vice-versa).

In many typical applications, the source and reference point clouds are already roughly aligned, but the source point cloud may cover a larger area than the reference. The user should provide to `pc_align` the expected maximum distance (displacement) source points may move by as result of alignment, using the option `--max-displacement`. This number will help remove source points too far from the reference point cloud which may not match successfully and may degrade the accuracy. If in doubt, this value can be set to something large but still reasonable, as the tool is able to throw away a certain number of unmatched outliers. At the end of alignment, `pc_align` will display the *observed* maximum displacement, a multiple of which can be used to seed the tool in a subsequent run. If an initial transform is applied to the source cloud

(section A.21.5), the outliers are thrown out *after* this operation. The observed maximum displacement is also between the source points with this transform applied and the source points after alignment to the reference.

The user can choose how many points to pick from the reference and source point clouds to perform the alignment. The amount of memory and processing time used by `pc_align` is directly proportional to these numbers, ideally the more points the better. Pre-cropping to judiciously chosen regions may improve the accuracy and/or run-time.

A.21.2 Alignment method

The default alignment method is Point-to-Plane ICP, which may be more robust to large translations than Point-to-Point ICP, though the latter can be good enough if the input point clouds have small alignment errors and it is faster and uses less memory as well. The tool also accepts an option named `--highest-accuracy` which will compute the normals for Point-to-Plane ICP at all points rather than about a tenth of them. This option is not necessary most of the time, but may result in better alignment at the expense of using more memory and processing time.

The default alignment transform is rigid, that is, a combination of rotation and translation. With Point-to-Point ICP, it is also possible to solve for a scale change (to obtain a so-called `similarity transform`). It is suggested this approach be used only when a scale change is expected. It can be turned on by setting `--alignment-method similarity-point-to-point`. (This method works best if an initial alignment is first performed with, for example, the Point-to-Plane approach, to determine the rotation and translation part of the transform, and then that one can be used as an initial guess in order to solve for the scale as well.)

For very large scale difference or translation among the two clouds, both of these algorithms may fail. If the clouds are DEMs, one may specify the option `--initial-transform-from-hillshading` *`string`* which will hillshade the two DEMs, find interest point matches among them, and use that to compute an initial transform between the clouds (section A.21.5), which may or may not contain scale, after which the earlier algorithms will be applied to refine the transform. This functionality is implemented with ASP's `hillshade`, `ipfind`, and `ipmatch` tools, and `pc_align` has options to pass flags to these programs, such as to increase the number interest points being found, if the defaults are not sufficient. If the two clouds look too different for interest point matching to work, they perhaps can be re-gridded to use the same (coarser) grid, as described in section A.21.10, to obtain the initial transform which can then be applied to the original clouds.

A non-ICP algorithm supported by ASP is *Fast Global Registration*, accessible with `--alignment-method fgr`, and customizable using the `--fgr-options` field (see the table below for more details). This approach can perform better than ICP when the clouds are close enough to each other but there is a large number of outliers, since it does a cross-check, so it can function with very large `--max-displacement`. It does worse if the clouds need a big shift to align.

This one is being advertised as less sensitive to outliers, hence it should give good results with a larger value of the maximum displacement.

Another option is to use least squares (with outlier handling using a robust cost function) to find the transform, if the reference cloud is a DEM. For this, one should specify the alignment method as `least-squares` or `similarity-least-squares` (the latter also solves for scale). It is suggested that the input clouds be very close or otherwise the `--initial-transform` option be used, for the method to converge, and use perhaps on the order of 10-20 iterations and a smaller value for `--max-num-source-points` (perhaps a few thousand) for this approach to converge reasonably fast.

A.21.3 File formats

The input point clouds can be in one of several formats: ASP's point cloud format (the output of `stereo`), DEMs as GeoTIFF or ISIS cub files, LAS files, or plain-text CSV files (with .csv or .txt extension).

By default, CSV files are expected to have on each line the latitude and longitude (in degrees), and the height above the datum (in meters), separated by commas or spaces. Alternatively, the user can specify the format of the CSV file via the `--csv-format` option. Entries in the CSV file can then be (in any order) (a) longitude, latitude (in degrees), height above datum (in meters), (b) longitude, latitude, distance from planet center (in meters or km), (c) easting, northing and height above datum (in meters), in this case a PROJ.4 string must be set via `--csv-proj4`, (d) Cartesian coordinates (x, y, z) measured from planet center (in meters). The precise syntax is described in the table below. The tool can also auto-detect the LOLA RDR PointPerRow format.

Any line in a CSV file starting with the pound character (#) is ignored.

If none of the input files have a geoheader with datum information, and the input files are not in Cartesian coordinates, the datum needs to be specified via the `--datum` option, or by setting `--semi-major-axis` and `--semi-minor-axis`.

A.21.4 The alignment transform

The transform obtained by `pc_align` is output to a text file as a 4×4 matrix with the upper-left 3×3 submatrix being the rotation (and potentially also a scale, per section A.21.2) and the top three elements of the right-most column being the translation. This transform, if applied to the source point cloud, will bring it in alignment with the reference point cloud. The transform assumes the 3D Cartesian coordinate system with the origin at the planet center (known as ECEF). This matrix can be supplied back to the tool as an initial guess (section A.21.5). The inverse transform is saved to a file as well.

A.21.5 Applying an initial transform

The transform output by `pc_align` can be supplied back to the tool as an initial guess via the `--initial-transform` option, with the same or different clouds. If it is desired to simply apply this transform to the clouds without further work, one can specify `--num-iterations 0`. This may be useful, for example, in first finding the alignment transform over a smaller, more reliable region (e.g., over rock, excluding moving ice), then applying it over the entire available dataset.

Alternatively, one can apply to the source cloud an initial shift, expressed in the North-East-Down coordinate system at the centroid of the source points, before the alignment algorithm is invoked. Hence, if it is desired to move the source cloud North by 5 m, East by 10 m, and down by 15 m relative to the point on planet surface which is the centroid of the source points, one can invoke `pc_align` with `--initial-ned-translation '5 10 15'` (notice the quotes).

If an initial transform is used, the alignment transform output by the program will be from the source points *before* the initial transform, hence the output alignment transform will incorporate the initial transform.

If a good initial alignment is found, it is suggested to use a smaller value for `--max-displacement`, as the clouds will already be mostly on top of each other after the initial transform is applied.

A.21.6 Interpreting the transform

The alignment transform, with its origin at the center of the planet, can result in large movements on the planet surface even for small angles of rotation. Because of this it may be difficult to interpret both its

rotation and translation components.

The `pc_align` program outputs the translation component of this transform, defined as the vector from the centroid of the original source points (before any initial transform applied to them) to the centroid of the source points with the computed alignment transform applied to them. This translation component is displayed in three ways (a) Cartesian coordinates with the origin at the planet center, (b) Local North-East-Down coordinates at the centroid of the source points (before any initial transform), and (c) Latitude-Longitude-Height differences between the two centroids. If the effect of the transform is small (e.g., the points moved by at most several hundred meters) then the representation in the form (b) above is most amenable to interpretation as it is in respect to cardinal directions and height above ground if standing at a point on the planet surface.

This program prints to screen the Euler angles of the rotation transform, and also the axis of rotation and the angle measured against that axis. It can be convenient to interpret the rotation as being around the center of gravity of the reference cloud, even though it was computed as a rotation around the planet center, since changing the point around which a rigid transform is applied will only affect its translation component, which is relative to that point, but not the rotation matrix.

A.21.7 Error metrics and outliers

The tool outputs to CSV files the lists of errors together with their locations in the source point cloud, before the alignment of the source points (but after applying any initial transform), and also after the alignment computed by the tool. They are named `<output prefix>-beg_errors.csv` and `<output prefix>-end_errors.csv`. An error is defined as the distance from a source point used in alignment to the closest reference point. The format of output CSV files is the same as of input CSV files, or as given by `--csv-format`, although any columns of extraneous data in the input files are not saved on output.

The program prints to screen and saves to a log file the 16th, 50th, and 84th error percentiles as well as the means of the smallest 25%, 50%, 75%, and 100% of the errors.

When the reference point cloud is a DEM, a more accurate computation of the errors from source points to the reference cloud is used. A source point is projected onto the datum of the reference DEM, its longitude and latitude are found, then the DEM height at that position is interpolated. That way we determine a "closest" point on the reference DEM that interprets the DEM not just as a collection of points but rather as a polyhedral surface going through those points. These errors are what is printed in the statistics. To instead compute errors as done for other type of point clouds, use the option `--no-dem-distances`.

By default, when `pc_align` discards outliers during the computation of the alignment transform, it keeps the 75% of the points with the smallest errors. As such, a way of judging the effectiveness of the tool is to look at the mean of the smallest 75% of the errors before and after alignment.

A.21.8 Output point clouds and convergence history

The transformed input point clouds (the source transformed to match the reference, and the reference transformed to match the source) can also be saved to disk if desired. If an input point cloud is in CSV or ASP point cloud format, the output transformed cloud will be in the same format. If the input is a DEM, the output will be an ASP point cloud, since a gridded point cloud may not stay so after a 3D transform. The `point2dem` program can be used to re-grid the obtained point cloud back to a DEM.

The convergence history for `pc_align` (the translation and rotation change at each iteration) is saved to disk and can be used to fine-tune the stopping criteria.

A.21.9 Manual alignment

If automatic alignment fails, for example, if the clouds are too different, or they differ by a scale factor, a manual alignment can be computed as an initial guess transform (and one can stop there if `pc_align` is invoked with 0 iterations). For that, the input point clouds should be first converted to DEMs using `point2dem`, unless in that format already. Then, `stereo_gui` can be called to create manual point correspondences (interest point matches) from the reference to the source DEM (hence they should be displayed in the GUI in this order, from left to right, and one can hillshade them to see features better). Once the match file is saved to disk, it can be passed to `pc_align` via the `--match-file` option, which will compute an initial transform before continuing with alignment. This transform can also be used for non-DEM clouds once it is found using DEMs obtained from those clouds.

A.21.10 Creating a point cloud from a DEM

Given a DEM, if one invokes `pc_align` as follows:

```
pc_align dem.tif dem.tif --max-displacement -1 --num-iterations 0 \
   --save-transformed-source-points -o run/run
```

this will create a point cloud out of the DEM. This cloud can then be re-gridded using `point2dem` at a lower resolution or with a different projection.

A.21.11 Troubleshooting

Remember that filtering is applied only to the source point cloud. If you have an input cloud with a lot of noise, make sure it is being used as the source cloud.

If you are not getting good results with `pc_align`, something that you can try is to convert an input point cloud into a smoothed DEM. Use `point2dem` to do this and set `--search-radius-factor` if needed to fill in holes in the DEM. For some input data this can significantly improve alignment accuracy.

Table A.21: Command-line options for pc_align

Options	Description
`--num-iterations` *default: 1000*	Maximum number of iterations.
`--max-displacement` *float*	Maximum expected displacement of source points as result of alignment, in meters (after the initial guess transform is applied to the source points). Used for removing gross outliers in the source (movable) point cloud.
`--output-prefix|-o` *filename*	Specify the output file prefix.
`--outlier-ratio` *default: 0.75*	Fraction of source (movable) points considered inliers (after gross outliers further than max-displacement from reference points are removed).
`--max-num-reference-points` *default:* 10^8	Maximum number of (randomly picked) reference points to use.
`--max-num-source-points` *default:* 10^5	Maximum number of (randomly picked) source points to use (after discarding gross outliers).

`--alignment-method` *default: point-to-plane*	The type of iterative closest point method to use. [point-to-plane, point-to-point, similarity-point-to-point, fgr, least-squares, similarity-least-squares]
`--highest-accuracy`	Compute with highest accuracy for point-to-plane (can be much slower).
`--datum` *string*	Use this datum for CSV files. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), and Moon (=D_MOON).
`--semi-major-axis` *float*	Explicitly set the datum semi-major axis in meters.
`--semi-minor-axis` *float*	Explicitly set the datum semi-minor axis in meters.
`--csv-format` *string*	Specify the format of input CSV files as a list of entries column_index:column_type (indices start from 1). Examples: '1:x 2:y 3:z' (a Cartesian coordinate system with origin at planet center is assumed, with the units being in meters), '5:lon 6:lat 7:radius_m' (longitude and latitude are in degrees, the radius is measured in meters from planet center), '3:lat 2:lon 1:height_above_datum', '1:easting 2:northing 3:height_above_datum' (need to set `--csv-proj4`; the height above datum is in meters). Can also use radius_km for column_type, when it is again measured from planet center.
`--csv-proj4` *string*	The PROJ.4 string to use to interpret the entries in input CSV files, if those files contain Easting and Northing fields.
`--compute-translation-only`	Compute the transform from source to reference point cloud as a translation only (no rotation).
`--save-transformed-source-points`	Apply the obtained transform to the source points so they match the reference points and save them.
`--save-inv-transformed-reference-points`	Apply the inverse of the obtained transform to the reference points so they match the source points and save them.
`--initial-transform` *string*	The file containing the transform to be used as an initial guess. It can come from a previous run of the tool.
`--initial-ned-translation` *string*	Initialize the alignment transform based on a translation with this vector in the North-East-Down coordinate system around the centroid of the reference points. Specify it in quotes, separated by spaces or commas.
`--initial-transform-from-hillshading` *string*	If both input clouds are DEMs, find interest point matches among their hillshaded versions, and use them to compute an initial transform to apply to the source cloud before proceeding with alignment. Specify here the type of transform, as one of: 'similarity' (rotation + translation + scale), 'rigid' (rotation + translation) or 'translation'.

`--hillshade-options`	Options to pass to the `hillshade` program when computing the transform from hillshading. Default: `--azimuth 300 --elevation 20 --align-to-georef`.
`--ipfind-options`	Options to pass to the `ipfind` program when computing the transform from hillshading. Default: `--ip-per-image 500000 --interest-operator sift --descriptor-generator sift`.
`--ipmatch-options`	Options to pass to the `ipmatch` program when computing the transform from hillshading. Default: `--inlier-threshold 100 --ransac-iterations 10000 --ransac-constraint similarity`.
`--match-file`	Compute an initial transform from the source to the reference point cloud using manually selected point correspondences (obtained for example using stereo_gui). The type of transform can be set via `--initial-transform-from-hillshading` *string*.
`--fgr-options`	Options to pass to the Fast Global Registration algorithm, if used. Default: 'div_factor: 1.4 use_absolute_scale: 0 max_corr_dist: 0.025 iteration_number: 100 tuple_scale: 0.95 tuple_max_cnt: 10000'.
`--diff-rotation-error` *default:* 10^{-8}	Change in rotation amount below which the algorithm will stop (if translation error is also below bound), in degrees.
`--diff-translation-error` *default:* 10^{-3}	Change in translation amount below which the algorithm will stop (if rotation error is also below bound), in meters.
`--no-dem-distances`	For reference point clouds that are DEMs, don't take advantage of the fact that it is possible to interpolate into this DEM when finding the closest distance to it from a point in the source cloud (the text above has more detailed information).
`--config-file` *file.yaml*	This is an advanced option. Read the alignment parameters from a configuration file, in the format expected by libpointmatcher, over-riding the command-line options.
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default as set by OpenMP. Only some parts of the algorithm are multi-threaded.
`--help|-h`	Display the help message.

A.22 n_align

This tool can be used to jointly align a set of two or more point clouds, hence it extends the functionality of `pc_align` (section A.21). It implements the ICP flavor from [147], more exactly, the MATLAB algorithm at

https://searchcode.com/file/13619767/Code/matlab/GlobalProcrustesICP/globalProcrustes.m

It is hoped that joint alignment will give less biased results than pairwise alignment for the clouds.

Usage:

```
n_align <cloud files> -o <output prefix>
```

This tool supports the same types of data on input and output as `pc_align`.

Even for two clouds this algorithm is not the same as the ones that are part of `pc_align`. This algorithm is expected to be more robust to outliers than the regular ICP in `pc_align` since it uses a cross-check. Yet, it may not handle a large translation difference between the clouds as well. In that case, given a set of clouds, one can first use `pc_align` to align all other clouds to the first one, then invoke this algorithm for joint alignment while passing the obtained alignment transforms as an argument to this tool, to be used as initial guesses. The option to use for this, as shown below for simplicity for three clouds, is:

```
--initial-transforms 'identity.txt run_12/run-transform.txt run_13/run-transform.txt'
```

where the file 'identity.txt' contains the 4×4 identity matrix (the transform from the first cloud to itself), and 'run_12/run' is the output prefix for `pc_align` when invoked on the first and second clouds, etc. The final transforms output by this tool will incorporate the initial guesses.

This tool should be less sensitive than `pc_align` to the order of the clouds since any two of them are compared against each other. The number of iterations and number of input points used will dramatically affect its performance, and likely the accuracy. Cropping all clouds to the same region is likely to to improve both run-time and the results.

Table A.22: Command-line options for n_align

Option	Description
`--num-iterations arg (=100)`	Maximum number of iterations.
`--max-num-points arg (=1000000)`	Maximum number of (randomly picked) points from each cloud to use.
`--csv-format arg`	Specify the format of input CSV files as a list of entries column_index:column_type (indices start from 1). Examples: '1:x 2:y 3:z', '2:file 5:lon 6:lat 7:radius_m', '3:lat 2:lon 1:height_above_datum 5:file', '1:easting 2:northing 3:height_above_datum' (need to set –csv-proj4). Can also use radius_km for column_type.
`--csv-proj4 arg`	The PROJ.4 string to use to interpret the entries in input CSV files.
`--datum arg`	Use this datum for CSV files instead of auto-detecting it. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).
`--semi-major-axis arg (=0)`	Explicitly set the datum semi-major axis in meters.

`--semi-minor-axis arg (=0)`	Explicitly set the datum semi-minor axis in meters.
`-o | --output-prefix arg`	Specify the output prefix. The computed alignment transforms and, if desired, the transformed clouds, will be saved to names starting with this prefix.
`--save-transformed-clouds`	Apply the obtained alignment transforms to the input clouds and save them.
`--initial-transforms-prefix arg`	The prefix of the transforms to be used as initial guesses. The naming convention is the same as for the transforms written on output.
`--initial-transforms arg`	Specify the initial transforms as a list of files separated by spaces and in quotes, that is, as 'trans1.txt ... trans_n.txt'.
`--relative-error-tolerance (=1e-10)`	Stop when the change in the error divided by the error itself is less than this.
`--align-to-first-cloud`	Align the other clouds to the first one, rather than to their common centroid.
`-v | --verbose`	Print the alignment error after each iteration.
`--threads arg (=0)`	Select the number of processors (threads) to use.
`--tile-size arg (=256, 256)`	Image tile size used for multi-threaded processing.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress arg (=LZW)`	TIFF Compression method. [None, LZW, Deflate, Packbits]
`-v | --version`	Display the version of software.
`-h | --help`	Display this help message.

A.23 pc_merge

This is a simple tool for combining multiple ASP-generated point cloud files into a single concatenated file. The output file will be float32 unless the input images are float64 or the user has specified the float64 option.

`pc_merge` can merge clouds with 1, 3, 4, and 6 bands. In particular, it can merge *output-prefix*-L.tif images created by `stereo`. This is useful if it is desired to create an ortho-image from a merged cloud with `point2dem`. In that case, one can invoke `pc_merge` on individual "L" files to create a merged texture file to pass to `point2dem` together with the merged point cloud tile.

Usage:

```
pc_merge [options] [required output file option] <multiple point cloud files>
```

Table A.23: Command-line options for pc_merge

Options	Description
`--help`	Display the help message.
`--write-double|-d`	Force output file to be float64 instead of float32.
`--output-file|-o`	Specify the output file (required).

A.24 wv_correct

An image taken by one of Digital Globe's World View satellite cameras is formed of several blocks as tall as the image, mosaicked from left to right, with each block coming from an individual CCD sensor [43]. Either due to imperfections in the camera or in the subsequent processing the image blocks are offset in respect to each other in both row and column directions by a subpixel amount. These so-called *CCD boundary artifacts* are not visible in the images but manifest themselves as discontinuities in the the DEMs obtained with ASP.

The tool named `wv_correct` is able to significantly attenuate these artifacts (see Figure 4.2 in the Digital Globe tutorial for an example). This tool should be used on raw Digital Globe images before calling `dg_mosaic` and `mapproject`.

It is important to note that both the positions of the CCD offsets and the offset amounts were determined empirically without knowledge of Digital Globe's mosaicking process; this is why we are not able to remove these artifacts completely.

Presently, `wv_correct` works for WV01 images for TDI of 8, 16, 32, 48, 56 and 64, and for WV02 images for TDI of 8, 16, 48, and 64 (both the forward and reverse scan directions for both cameras). In addition, the WV02 TDI 32 forward scan direction is supported. These are by far the most often encountered TDI. We plan to extend the tool to support other TDI when we get more such data to be able to compute the corrections. For WV03 images, CCD artifacts appear to not be significant, hence no corrections are planned for the near future.

Usage:

```
wv_correct [options] <input image> <input camera model> <output image>
```

Table A.24: Command-line options for wv_correct

Options	Description
`--ot` *string(=Float32)*	Output data type. Supported types: Byte, UInt16, Int16, UInt32, Int32, Float32. If the output type is a kind of integer, values are rounded and then clamped to the limits of that type.
`--help|-h`	Display the help message.
`--threads` *integer(=0)*	Set the number threads to use. 0 means use the default defined in the program or in the .vwrc file.

A.25 hiedr2mosaic.py

Assemble a collection of HiRISE EDR files into a single image. This runs the sequence of ISIS preprocessing commands, followed by hijitreg, to assemble the input images into a single output image. You can either download the input files yourself and pass them all in or specify a download folder and pass in only a URL such as http://hirise-pds.lpl.arizona.edu/PDS/EDR/ESP/ORB_029400_029499/ESP_029421_2300/. If you use a URL, the program will attempt to download all of the HiRISE images found at that location and then run the processing script. See the "Mars Reconnaissance Orbiter HiRISE" section in the examples chapter for a more detailed explanation.

Usage:

```
hiedr2mosaic.py [options] <input files OR a URL>
```

Table A.25: Command-line options for hiedr2mosaic.py

Options	Description
`--manual`	Display the help message.
`--match|-m`	The CCD number passed as the match argument to noproj (default 5). .
`--stop-at-no-proj`	Stops processing after the noproj steps are complete.
`--resume-at-no-proj`	Restarts processing using the results from 'stop-at-no-proj.
`--download-folder`	Download input files to this folder. Must pass in a URL instead of files.
`--threads|-t`	Specify the number of threads to use.
`--keep|-k`	Keep all intermediate files.

A.26 lronac2mosaic.py

This tool takes in two LRONAC files (M*LE.IMG and M*RE.IMG) and produces a single noproj mosaic composed of the two inputs. It performs the following operations in this process: `lronac2isis`, `lronaccal`, `lronacecho`, `spiceinit`, `noproj`, and `handmos`. The offsets used in `handmos` are calculated using an ASP internal tool called `lronacjitreg` and is similar in functionality to the ISIS command `hijitreg`. Offsets need to be calculated via feature measurements in image to correct for imperfections in camera pointing. The angle between LE and RE optics changes slightly with spacecraft temperature.

Optionally, `lronac2mosiac.py` can be given many IMG files all at once. The tool will then look at image names to determine which should be paired and mosaicked. The tool will also spawn multiple processes of ISIS commands were possible to finish the task faster. The max number of simultaneous processes is limited by the `--threads` option.

Usage:

```
lronac2mosaic.py [options] <IMG file 1> <IMG file 2>
```

Table A.26: Command-line options for lronac2mosaic.py

Options	Description
`--manual`	Display the help message.
`--output-dir|-o`	Set the output folder (default is input folder).
`--stop-at-no-proj`	Stops processing after the noproj steps are complete.
`--resume-at-no-proj`	Restarts processing using the results from 'stop-at-no-proj.
`--threads|-t`	Specify the number of threads to use.
`--keep|-k`	Keep all intermediate files.

A.27 image_calc

This tool can be used to perform simple, per-pixel arithmetic on one or more input images. An arithmetic operation specified on the command line is parsed and applied to each pixel, then the result is written to disk. The tool supports multiple input images but each must be the same size and data type. Input images are restricted to one channel.

The following symbols are allowed in the arithmetic string: +, -, *, /, (), min(), max(), pow(), abs(), and var_N where N is the index of one of the input images. An example arithmetic string is: "-abs(var_2)

+ min(58, var_1, var_3) / 2". The tool respects the normal PEMDAS order of operations *except* that it parses equal priority operations from right to left, *not* the expected left to right. Parentheses can be used to enforce any preferred order of evaluation.

Usage:

```
image_calc [options] -c <arithmetic formula> <inputs> -o <output>
```

Example:

```
image_calc -c "pow(var_0/3.0, 1.1)" input_image.tif -o output_image.tif -d float32
```

Table A.27: Command-line options for image_calc

Options	Description
`--help`	Display the help message.
`--calc|-c`	The arithmetic string in quotes (required).
`--output-data-type|-d`	The data type of the output file (default is float64).
`--input-nodata-value`	Set an override nodata value for the input images.
`--output-nodata-value`	Manually specify a nodata value for the output image (default is data type min).
`--output-file|-o`	Specify the output file instead of using a default.
`--mo` *string*	Write metadata to the output file. Provide as a string in quotes if more than one item, separated by a space, such as 'VAR1=VALUE1 VAR2=VALUE2'. Neither the variable names nor the values should contain spaces.

A.28 hsv_merge

Replaces the intensity information in an RGB image with the provided grayscale image by temporarily converting to HSV. Both input image must be the same size.

Mimics `hsv_merge.py` by Frank Warmerdam and Trent Hare. Use it to combine results from gdaldem.

Usage:

```
hsv_merge [options] <rgb_image> <gray_image>
```

Table A.28: Command-line options for hsv_merge

Options	Description
`--help`	Display the help message.
`--output-file|-o`	Specify the output file. Required!

A.29 colormap

The `colormap` tool reads a DEM and writes a corresponding color-coded height image that can be used for visualization.

Usage:

```
colormap [options] <input DEM>
```

Table A.29: Command-line options for colormap

Option	Description
`--help`	Display a help message.
`-s [ --shaded-relief-file ] arg`	Specify a shaded relief image (grayscale) to apply to the colorized image.
`-o [ --output-file ] arg`	Specify the output file.
`--colormap-style arg`	Specify the colormap style. Options: binary-red-blue (default), jet, or the name of a file having the colormap, similar to the file used by gdaldem.
`--nodata-value arg`	Remap the DEM default value to the min altitude value.
`--min arg`	Minimum height of the color map.
`--max arg`	Maximum height of the color map.
`--moon`	Set the min and max height to good values for the Moon.
`--mars`	Set the min and max height to good values for Mars.
`--legend`	Generate an unlabeled legend, saved as "legend.png".

A.30 hillshade

The `hillshade` tool reads in a DEM and outputs an image of that DEM as though it were a three-dimensional surface, with every pixel shaded as though it were illuminated by a light from a specified location.

Table A.30: Command-line options for hillshade

Option	Description
`--help`	Display a help message
`--input-file arg`	Explicitly specify the input file
`-o [ --output-file ] arg`	Specify the output file
`--align-to-georef`	Azimuth is relative to geographic East, not +x in the image.
`-a [ --azimuth ] arg (=300)`	Sets the direction that the light source is coming from (in degrees). Zero degrees is to the right, with positive degree counter-clockwise.
`-e [ --elevation ] arg (=20)`	Set the elevation of the light source (in degrees)
`-s [ --scale ] arg (=0)`	Set the scale of a pixel (in the same units as the DTM height values
`--nodata-value arg`	Remap the DEM default value to the min altitude value
`--blur arg`	Pre-blur the DEM with the specified sigma

A.31 image2qtree

`image2qtree` turns a georeferenced image (or images) into a quadtree with geographical metadata. For example, it can output a kml file for viewing in Google Earth.

Table A.31: Command-line options for image2qtree

Option	Description
General Options	
`--help`	Display a help message
`-o [ --output-name ] arg`	Specify the base output directory
`-q [ --quiet ]`	Quiet output
`-v [ --verbose ]`	Verbose output
`--cache arg (=1024)`	Cache size, in megabytes
Input Options	
`--force-wgs84`	Use WGS84 as the input images' geographic coordinate systems, even if they're not (old behavior)
`--pixel-scale arg (=1)`	Scale factor to apply to pixels
`--pixel-offset arg (=0)`	Offset to apply to pixels
`--normalize`	Normalize input images so that their full dynamic range falls in between [0,255]
Output Options	
`-m [ --output-metadata ] arg (=none)`	Specify the output metadata type. One of [kml, tms, uniview, gmap, celestia, none]
`--file-type arg (=png)`	Output file type
`--channel-type arg (=uint8)`	Output (and input) channel type. One of [uint8, uint16, int16, float]
`--module-name arg (=marsds)`	The module where the output will be placed. Ex: marsds for Uniview, or Sol/Mars for Celestia
`--terrain`	Outputs image files suitable for a Uniview terrain view. Implies output format as PNG, channel type uint16. Uniview only
`--jpeg-quality arg (=0.75)`	JPEG quality factor (0.0 to 1.0)
`--png-compression arg (=3)`	PNG compression level (0 to 9)
`--palette-file arg`	Apply a palette from the given file
`--palette-scale arg`	Apply a scale factor before applying the palette
`--palette-offset arg`	Apply an offset before applying the palette
`--tile-size arg (=256)`	Tile size, in pixels
`--max-lod-pixels arg (=1024)`	Max LoD in pixels, or -1 for none (kml only)
`--draw-order-offset arg (=0)`	Offset for the <drawOrder> tag for this overlay (kml only)
`--composite-multiband`	Composite images using multi-band blending
`--aspect-ratio arg (=1)`	Pixel aspect ratio (for polar overlays; should be a power of two)
Projection Options	
`--north arg`	The northernmost latitude in degrees

`--south arg`	The southernmost latitude in degrees
`--east arg`	The easternmost longitude in degrees
`--west arg`	The westernmost longitude in degrees
`--force-wgs84`	Assume the input images' geographic coordinate systems are WGS84, even if they're not (old behavior)
`--sinusoidal`	Assume a sinusoidal projection
`--mercator`	Assume a Mercator projection
`--transverse-mercator`	Assume a transverse Mercator projection
`--orthographic`	Assume an orthographic projection
`--stereographic`	Assume a stereographic projection
`--lambert-azimuthal`	Assume a Lambert azimuthal projection
`--lambert-conformal-conic`	Assume a Lambert Conformal Conic projection
`--utm arg`	Assume UTM projection with the given zone
`--proj-lat arg`	The center of projection latitude (if applicable)
`--proj-lon arg`	The center of projection longitude (if applicable)
`--proj-scale arg`	The projection scale (if applicable)
`--std-parallel1 arg`	Standard parallels for Lambert Conformal Conic projection
`--std-parallel2 arg`	Standard parallels for Lambert Conformal Conic projection
`--nudge-x arg`	Nudge the image, in projected coordinates
`--nudge-y arg`	Nudge the image, in projected coordinates

A.32 geodiff

The `geodiff` program takes as input two DEMs (or a DEM and a CSV file, with the latter in the same format as used for `pc_align` and `point2dem`), and subtracts the second from the first. The grid used is the one from the first DEM, so the second one is interpolated into it (when one file is a CSV, the grid from the other one, the DEM, is used). The tool can also take the absolute difference of the two DEMs.

It is important to note that the tool is very sensitive to the order of the two DEMs, due to the fact that the grid comes from the first one. Ideally the grid of the first DEM would be denser than the one of the second.

Usage:

```
> geodiff [options] <dem1> <dem2> [ -o output_file_prefix ]
```

Table A.32: Command-line options for geodiff

Option	Description
`--help|-h`	Display the help message.
`--output-prefix|-o` *filename*	Specify the output prefix.
`--absolute`	Output the absolute difference as opposed to just the difference.

`--float`	Output using float (32 bit) instead of using doubles (64 bit).
`--csv-format` *string*	Specify the format of input CSV files as a list of entries column_index:column_type (indices start from 1). Examples: '1:x 2:y 3:z' (a Cartesian coordinate system with origin at planet center is assumed, with the units being in meters), '5:lon 6:lat 7:radius_m' (longitude and latitude are in degrees, the radius is measured in meters from planet center), '3:lat 2:lon 1:height_above_datum', '1:easting 2:northing 3:height_above_datum' (need to set `--csv-proj4`; the height above datum is in meters). Can also use radius_km for column_type, when it is again measured from planet center.
`--csv-proj4` *string*	The PROJ.4 string to use to interpret the entries in input CSV files, if those files contain Easting and Northing fields. If not specified, it will be borrowed from the DEM.
`--nodata-value` *float(=-32768)*	The no-data value to use, unless present in the DEM geoheaders.
`--threads` *integer(=0)*	Set the number of threads to use. 0 means use as many threads as there are cores.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.

A.33 aster2asp

The `aster2asp` tool takes as input a directory containing ASTER images and associated metadata, and creates TIF and XML files that can then be passed to `stereo` to create a point cloud.

An example for how to use this tool is given in section 11.15.

The tool can only process Level 1A ASTER images. The input should be a directory containing visible and near-infrared (VNIR) nadir (Band3N) and backward (Band3B) images, together with plain text files containing values for the satellite positions, sight vectors, longitudes, latitudes, lattice points, and radiometric correction tables. These files are described in [1].

Usage:

```
aster2asp <input directory> -o <output prefix>
```

The tool will apply the existing radiometric corrections to the the images, and save two images with Float32 pixels with names like `out-Band3N.tif` and `out-Band3B.tif`. Based on the metadata mentioned earlier, it will create approximate RPC camera models in XML format (section 11.12) for the left and right cameras, following [42], with names of the form `out-Band3N.xml` and `out-Band3B.xml` (we do not perform yet any jitter corrections as described in that paper).

These can then be passed to `stereo` as:

```
stereo -t rpc out-Band3N.tif out-Band3B.tif out-Band3N.xml out-Band3B.xml \
   out_stereo/run
```

It is important to note that the tool expects the minimum and maximum simulation box heights (in meters, above the datum) in which to compute the RPC approximation. The defaults are 0 and 8000, corresponding to sea level and the highest location on Earth. Narrowing down these numbers (if it is known what range of terrain heights is expected) may result in slightly more accurate models.

Table A.33: Command-line options for aster2asp

Option	Description
`-o | --output-prefix arg`	Specify the output prefix.
`--min-height arg (=0)`	The minimum height (in meters) above the WGS84 datum of the simulation box in which to compute the RPC approximation.
`--max-height arg (=8000)`	The maximum height (in meters) above the WGS84 datum of the simulation box in which to compute the RPC approximation.
`--num-samples arg (=100)`	How many samples to use between the minimum and maximum heights.
`--penalty-weight arg (=0.1)`	Penalty weight to use to keep the higher-order RPC coefficients small. Higher penalty weight results in smaller such coefficients.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress arg (=LZW)`	TIFF Compression method. [None, LZW, Deflate, Packbits]
`-v | --version`	Display the version of software.
`-h | --help`	Display this help message.

A.34 add_spot_rpc

The `add_spot_rpc` tool creates an RPC model to approximate a SPOT5 sensor model. The RPC model can be appended to the end of a SPOT5 metadata file, allowing it to be used with the RPC session type in other ASP tools. The most important application is to map project SPOT5 images, then to perform stereo on the map projected images with the `spot5maprpc` session type.

If the output file does not exist, a new file is created containing the RPC model. Otherwise the RPC model is appended to an existing file. When an existing SPOT5 metadata file is the output file, the new RPC model is properly inserted into the file so that it is ready to use.

An example for how to use this tool is given in section 11.13.

Usage:

```
add_spot_rpc <input metadata file> -o <output file>
```

It is important to note that the tool expects the minimum and maximum simulation box heights (in meters, above the datum) in which to compute the RPC approximation. The defaults are 0 and 8000, corresponding to sea level and the highest location on Earth. Narrowing down these numbers (if it is known what range of terrain heights is expected) may result in slightly more accurate models.

Table A.34: Command-line options for add_spot_rpc

Option	Description
`-o | --output-prefix arg`	Specify the output prefix.
`--min-height arg (=0)`	The minimum height (in meters) above the WGS84 datum of the simulation box in which to compute the RPC approximation.
`--max-height arg (=8000)`	The maximum height (in meters) above the WGS84 datum of the simulation box in which to compute the RPC approximation.
`--num-samples arg (=100)`	How many samples to use between the minimum and maximum heights.
`--penalty-weight arg (=0.1)`	Penalty weight to use to keep the higher-order RPC coefficients small. Higher penalty weight results in smaller such coefficients.
`-v | --version`	Display the version of software.
`-h | --help`	Display this help message.

A.35 sfs

The `sfs` tool can improve a DEM using shape-from-shading. Examples for how to use it are in chapter 10. The tool `parallel_sfs` (next section) extends `sfs` to run using multiple processes and potentially on multiple machines.

Usage:

```
sfs -i <input DEM> -n <max iterations> -o <output prefix> <images> [other options]
```

The tool outputs at each iteration the current DEM and a slew of other auxiliary and appropriately-named datasets.

Table A.35: Command-line options for sfs

Option	Description
`-i | --input-dem arg`	The input DEM to refine using SfS.
`-o | --output-prefix arg`	Prefix for output filenames.
`-n | --max-iterations arg (=100)`	Set the maximum number of iterations.
`--reflectance-type arg (=1)`	Reflectance type (0 = Lambertian, 1 = Lunar-Lambert, 2 = Hapke, 3 = Experimental extension of Lunar-Lambert, 4 = Charon model (a variation of Lunar-Lambert)).
`--smoothness-weight arg (=0.04)`	A larger value will result in a smoother solution.
`--initial-dem-constraint-weight arg (=0)`	A larger value will try harder to keep the SfS-optimized DEM closer to the initial guess DEM.
`--albedo-constraint-weight arg (=0)`	If floating the albedo, a larger value will try harder to keep the optimized albedo close to the nominal value of 1.

`--bundle-adjust-prefix arg`	Use the camera adjustments obtained by previously running bundle_adjust with this output prefix.
`--float-albedo`	Float the albedo for each pixel. Will give incorrect results if only one image is present.
`--float-exposure`	Float the exposure for each image. Will give incorrect results if only one image is present.
`--float-cameras`	Float the camera pose for each image except the first one.
`--float-all-cameras`	Float the camera pose for each image, including the first one. Experimental.
`--model-shadows`	Model the fact that some points on the DEM are in the shadow (occluded from the Sun).
`--shadow-thresholds arg`	Optional shadow thresholds for the input images (a list of real values in quotes, one per image).
`--save-dem-with-nodata`	Save a copy of the DEM while using a no-data value at a DEM grid point where all images show shadows. To be used if shadow thresholds are set.
`--use-approx-camera-models`	Use approximate camera models for speed.
`--use-rpc-approximation`	Use RPC approximations for the camera models instead of approximate tabulated camera models (invoke with –use-approx-camera-models).
`--rpc-penalty-weight arg (=0.1)`	The RPC penalty weight to use to keep the higher-order RPC coefficients small, if the RPC model approximation is used. Higher penalty weight results in smaller such coefficients.
`--coarse-levels arg (=0)`	Solve the problem on a grid coarser than the original by a factor of 2 to this power, then refine the solution on finer grids. Experimental.
`--max-coarse-iterations arg (=50)`	How many iterations to do at levels of resolution coarser than the final result.
`--crop-input-images`	Crop the images to a region that was computed to be large enough and keep them fully in memory, for speed.
`--image-exposures-prefix arg`	Use this prefix to optionally read initial exposures (filename is <prefix>-exposures.txt).
`--model-coeffs-prefix arg`	Use this prefix to optionally read model coefficients from a file (filename is <prefix>-model_coeffs.txt) .

`--model-coeffs arg`	Use the model coefficients specified as a list of numbers in quotes. Lunar-Lambertian: O, A, B, C, e.g., '1 0.019 0.000242 -0.00000146'. Hapke: omega, b, c, B0, h, e.g., '0.68 0.17 0.62 0.52 0.52'. Charon: A, f(alpha), e.g., '0.7 0.63'.
`--crop-win arg (=xoff yoff xsize ysize)`	Crop the input DEM to this region before continuing.
`--init-dem-height arg (=nan)`	Use this value for initial DEM heights. An input DEM still needs to be provided for georeference information.
`--nodata-value arg (=nan)`	Use this as the DEM no-data value, overriding what is in the initial guess DEM.
`--float-dem-at-boundary`	Allow the DEM values at the boundary of the region to also float (not advised).
`--fix-dem`	Do not float the DEM at all. Useful when floating the model params.
`--float-reflectance-model`	Allow the coefficients of the reflectance model to float (not recommended).
`--integrability-constraint-weight arg (=0.0)`	Use the integrability constraint from Horn 1990 with this value of its weight (experimental).
`--smoothness-weight-pq (=0.0)`	Smoothness weight for p and q, when the integrability constraint is used. A larger value will result in a smoother solution (experimental).
`--query`	Print some info and exit. Invoked from parallel_sfs.
`--camera-position-step-size arg (=1)`	Larger step size will result in more aggressiveness in varying the camera position if it is being floated (which may result in a better solution or in divergence).
`--threads arg (=0)`	Select the number of processors (threads) to use.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress arg (=LZW)`	TIFF Compression method. [None, LZW, Deflate, Packbits]
`-v | --version`	Display the version of software.
`-h | --help`	Display this help message.

A.36 parallel_sfs

The program `parallel_sfs` is a wrapper around `sfs` meant to divide the input DEM into tiles with overlap, run `sfs` on each tile as multiple processes, potentially on multiple machines, and then merge the results into a single output DEM. It has the same options as `sfs`, and a few additional ones, as outlined below.

Usage:

```
parallel_sfs -i <input DEM> -n <max iterations> -o <output prefix> <images> [other options]
```

Table A.36: Command-line options for parallel_sfs

Option	Description
`--tile-size (integer=300)`	Size of approximately square tiles to break up processing into (not counting the padding).
`--padding (integer=50)`	How much to expand a tile in each direction. This helps with reducing artifacts in the final mosaicked SfS output.
`--num-processes integer`	Number of processes to use (the default program tries to choose best).
`--nodes-list string`	A file containing the list of computing nodes, one per line. If not provided, run on the local machine.
`--threads (integer=1)`	How many threads each process should use. The sfs executable is single-threaded in most of its execution, so a large number will not help here.
`--suppress-output`	Suppress output of sub-calls.

A.37 undistort_image

The `undistort_image` program takes as input an image and a pinhole model .tsai file describing the image. The tool will generate a copy of the input image with the lens distortion specified in the pinhole model file removed. It will also save the corresponding pinhole camera model file without the distortion.

Usage:

```
> undistort_image [options] <input image> <camera model> -o <output image>
```

Table A.37: Command-line options for undistort_image

Option	Description
`--help|-h`	Display the help message.
`--output-file|-o` *filename*	Specify the output file.
`--output-nodata-value` *double(=smallest float)*	Set the output nodata value. Only applicable if the output is a single-channel image with pixels that are float or double.
`--preserve-pixel-type`	Save the undistorted image with integer pixels if so is the input. This may result in reduced accuracy.
`--interpolation-method` *string*	Interpolation method. Options: bilinear, bicubic. Default: bilinear.

A.38 camera_calibrate

The `camera_calibrate` tool can generate camera models suitable for use by `camera_solve` and other ASP tools. This tool only solves for intrinsic camera parameters; to obtain the camera pose you should use the `camera_solve` tool. This tool is a wrapper around the OpenCV (http://opencv.org/) checker-

board calibration tool which takes care of converting the output into readily usable formats. When you run the tool, three camera model files will be created in the output folder: `solve_cam_params.txt`, `vw_cam_params.tsai`, and `ocv_cam_params.yml`. The first file can be used as a camera calibration file for the `camera_solve` tool. The second file is a pinhole camera format that is recognized by ASP but remember that the extrinsic parameters were not solved for so ASP is limited in what it can do with the camera file. The last file contains the camera information as formatted by the OpenCV calibration tool. If you use the first file as an input to `camera_solve` you must remember to replace the wildcard image path in the file with the one to the images you want to use solve for (as opposed to the checkerboard images).

In order to use this tool you must provide multiple images of the same checkerboard pattern acquired with the camera you wish to calibrate. When calling the tool you must specify the number of internal square corners contained in your checkerboard pattern (width and height can be swapped) so that OpenCV knows what to look for. You must also specify an image wildcard path such as `"checkers/image_*.jpg"`. You may need to enclose this parameter in quotes so that your command line does not expand the wildcard before it is passed to the tool. If you do not provide the `-box-size` parameter the output calibration numbers will be unitless.

Usage:

```
> camera_calibrate [options] <output folder> <Board Height> <Board Width> <Image Wildcard> ...
```

Table A.38: Command-line options for camera_calibrate

Option	Description
`-h | --help`	Display this help message.
`--overwrite`	Recompute any intermediate steps already completed on disk.
`--suppress-output`	Reduce the amount of program console output.
`--box-size-cm` *float*	The size of the checkerboard squares in centimeters.
`--duplicate-files`	Make a copy of the vw param file for each input camera.

A.39 camera_solve

The `camera_solve` tool generates pinhole sensor models (frame cameras), including camera poses, for input images lacking metadata. See chapter 9 for an overview and examples of using the tool.

The camera calibration passed with the `--calib-file` option should be a .tsai pinhole camera model file in one of the formats compatible with ASP. Our supported pinhole camera models are described in section D.1.

You can use a set of estimated camera positions to register camera models in world coordinates. This method is not as accurate as using ground control points but it may be easier to use. To do this, use the `--camera-positions` parameter to `bundle-adjust` via the `--bundle-adjust-params` option similar to the example line below. If you see the camera models shifting too far from their starting positions try using the `--camera-weight` option to restrain their movement.

```
--bundle-adjust-params '--camera-positions nav.csv \
 --csv-format "1:file 12:lat 13:lon 14:height_above_datum" --camera-weight 1.0'
```

This tool will generate two .tsai camera model files in the output folder per input image. The first file, appended with .tsai, is in a local coordinate system and does not include optimizations for intrinsic parameters but it may be useful for debugging purposes. The second file, appended with .final.tsai, contains the final solver results. If ground control points or estimated camera positions were provided then the second file will be in a global coordinate system.

Usage:

```
> camera_solve [options] <output folder> <Input Image 1> <Input Image 2> ...
```

Table A.39: Command-line options for camera_solve

Option	Description
`-h | --help`	Display this help message.
`--datum` *string*	The datum to use when calibrating. Default is WGS84.
`--calib-file` *string*	Path to an ASP compatible pinhole model file containing camera model information. The position and pose information will be ignored. If you want to use a unique file for each input image, pass a space separated list of files surrounded by quotes.
`--gcp-file` *string*	Path to a ground control point file. This allows the tool to generate cameras in a global coordinate system.
`--bundle-adjust-params` *string*	Additional parameters (in single quotes) to pass to the `bundle_adjust` tool.
`--theia-overrides` *string*	Override any option in the auto-generated Theia flag file. Set as "–option1=val1 –option2=val2 ...".
`--theia-flagfile` *string*	Path to a custom Theia flagfile to use settings from. File paths specified in this file are ignored.
`--overwrite`	Recompute any intermediate steps already completed on disk.
`--reuse-theia-matches`	Pass Theia's IP find results into ASP instead of recomputing them to reduce total processing time.
`--suppress-output`	Reduce the amount of program console output.

This tool is a wrapper that relies on on two other tools to operate. The first of these is THEIA, as mentioned earlier, for computing the relative poses of the cameras. ASP's `bundle_adjust` tool is used to register the cameras in world coordinates using the ground control points. If the tool does not provide good results you can customize the parameters being passed to the underlying tools in order to improve the results. For `bundle_adjust` options, see the description in this document. For more information about THEIA flagfile options see their website or edit a copy of the default flagfile generated in the output folder

A.40 convert_pinhole_model

This tool can be used to approximately convert a pinhole model from one of the types listed in section D.1 or an optical bar model (section D.3) to any other pinhole model type. This can be convenient, for example, because the Brown-Conrady and Photometrix models provide a fast formula to undistort pixels, while the distortion operation is very slow, requiring a solver with multiple iterations using the undistortion formula at each step, which can make it time-consuming to run bundle adjustment and epipolar alignment during stereo. For other models, such as Tsai and Adjustable Tsai, the reverse is true, hence converting from the former to the latter models can be very convenient for performance reasons.

This program can also be used to convert a pinhole or optical bar model to a pinhole model with RPC lens distortion, which is a model where distortion is expressed as a ratio of polynomials. The RPC lens distortion model has the advantage that both the forward and reverse distortion calculation are approximated using RPC, hence both of these operations are fast, which can provide a large speedup when running stereo and bundle adjustment.

The degree of the RPC lens distortion can be specified via `--rpc-degree`. A smaller value is suggested to start with, as lower-degree polynomials may be easier to interpret.

Usage:

```
convert_pinhole_model [options] <input image> <input camera> -o <output camera>
```

Example (convert a camera model to the RPC type):

```
convert_pinhole_model input.jpg input.tsai --output-type RPC \
  --rpc-degree 2 -o output_rpc.tsai
```

Table A.40: Command-line options for convert_pinhole_model

Option	Description
`-h | --help`	Display this help message.
`-o | --output-file` *string*	Specify the output file. It is expected to have the .tsai extension.
`--output-type` *string*	The output model type. Options: TsaiLensDistortion, BrownConradyDistortion, RPC. Default: TsaiLensDistortion.
`--rpc-degree` *int*	The degree of the polynomials, if the output distortion model is RPC. Default: 3.
`--sample-spacing` *int*	Pick one out of this many consecutive pixels to sample. If not specified, it will be auto-computed.

A.41 cam_gen

This tool will create a Pinhole or Optical Bar camera model given camera's optical center, focal length, pixel pitch, the longitude-latitude coordinates of the camera image corners (or some other pixels) projected onto a DEM, and the DEM itself. A datum (and a height above it) can be used instead of the DEM. Normally all these inputs are known only approximately, so the output camera model will not be quite precise either, yet it could be good enough to refine later with bundle adjustment, which can also make use of the GCP file that this tool creates.

This program can be used with historical images for which camera position and orientation is not known. If the corners of the image on the ground are not known, they could be guessed in Google Earth. A good DEM to infer the heights from, at least for Earth, is the SRTM dataset. Section 11.16 makes use of `cam_gen` for SkySat images.

Usage:

```
cam_gen [options] <image-file> -o <camera-file>
```

Example:

```
cam_gen --refine-camera --lon-lat-values                                  \
  '-122.389 37.6273,-122.354 37.626,-122.358 37.6125,-122.393 37.6138'    \
   --reference-dem dem.tif --focal-length 553846.153846                   \
   --optical-center 1280 540 --pixel-pitch 1                              \
   img.tif -o img.tsai --gcp-file img.gcp --gcp-std 1e-2                  \
```

Here we assume that the pixel pitch is 1, hence both the focal length and the optical center are in units of pixels. If the focal length and pixel pitch are given in meters, and one assumes the optical center to be the center of the image, then the optical center passed to this tool should be half of the image width and height, with both multiplied by the pixel pitch, to make them in meters as well.

Some other pixels can be used instead of corners, if using the `--pixel-values` option.

Note that for Optical Bar cameras the camera parameters must be passed in using the `--sample-file` option instead of specifying them all manually.

It is strongly suggested to mapproject the image onto the obtained camera to verify if it projects where expected:

```
mapproject dem.tif img.tif img.tsai img_map.tif
```

The output `img_map.tif` can be overlayed onto the hillshaded DEM in `stereo_gui`.

The camera obtained using this tool (whether with or without the `--refine-camera` option) can be further optimized in `bundle_adjust` using the GCP file written above as follows:

```
bundle_adjust img.tif img.tsai img.gcp -o run/run --datum WGS84 \
  --inline-adjustments --robust-threshold 10000
```

It is suggested that this is avoided by default. One has to be a bit careful when doing this optimization to ensure some corners are not optimized at the expense of others. This is discussed in section 9.4.

One can invoke `orbitviz` as:

```
orbitviz img.tif img.tsai -o orbit.kml
```

to visualize the computed camera above the ground in Google Earth.

This tool can also create a Pinhole camera approximating any camera supported by ASP, such as from ISIS cubes, RPC cameras, etc., as long as the intrinsics are known, as above. For that, it will shoot rays from the image corners (and also some inner points) using the provided camera that will intersect a reference DEM determining the footprint on the ground, and then the best-fit pinhole model will be created based on that. Here's an example for ISIS cameras:

```
cam_gen image.cub --input-camera image.cub --focal-length 1000       \
  --optical-center 500 300 --pixel-pitch 1 --height-above-datum 4000 \
  --gcp-std 1 --datum WGS84 --refine-camera --reference-dem dem.tif  \
  -o output.tsai --gcp-file output.gcp
```

Here we passed the image as the input camera, since for ISIS cubes (and also for some RPC cameras) the camera information is not stored in a separate camera file.

Table A.41: Command-line options for cam_gen.

Option	Description
`-o | --output-camera-file` *string*	Specify the output camera file with a .tsai extension.
`--camera-type` *string*	Specify the camera type. Options are: pinhole (default) and opticalbar.
`--lon-lat-values` *string*	A (quoted) string listing numbers, separated by commas or spaces, having the longitude and latitude (alternating and in this order) of each image corner. The corners are traversed in the order 0,0 w,0, w,h, 0,h where w and h are the image width and height.
`--pixel-values` *string*	A (quoted) string listing numbers, separated by commas or spaces, having the column and row (alternating and in this order) of each pixel in the raw image at which the longitude and latitude is known. By default this is empty, and will be populated by the image corners traversed as earlier.
`--reference-dem` *string*	Use this DEM to infer the heights above datum of the image corners.
`--datum` *string*	Use this datum to interpret the longitude and latitude, unless a DEM is given. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).
`--height-above-datum arg` *double(=0)*	Assume this height above datum in meters for the image corners unless read from the DEM.

`--sample-file` *string*	Instead of manually specifying all of the camera parameters, specify a sample camera model file on disk to read them from.
`--focal-length` *double(=0)*	The camera focal length.
`--optical-center` *double(=0 0)*	The camera optical center.
`--pixel-pitch` *double(=0)*	The camera pixel pitch.
`--refine-camera`	After a rough initial camera is obtained, refine it using least squares.
`--frame-index` *string*	A file used to look up the longitude and latitude of image corners based on the image name, in the format provided by the SkySat video product.
`--gcp-file` *string*	If provided, save the image corner coordinates and heights in the GCP format to this file.
`--gcp-std arg` *double(=1)*	The standard deviation for each GCP pixel, if saving a GCP file. A smaller value suggests a more reliable measurement, hence will be given more weight.
`--input-camera` *string*	Create the output pinhole camera approximating this camera.
`-t | --session-type` *string*	Select the input camera model type. Normally this is auto-detected, but may need to be specified if the input camera model is in XML format. Options: pinhole isis rpc dg spot5 aster opticalbar.
`--bundle-adjust-prefix` *string*	Use the camera adjustment obtained by previously running bundle_adjust when providing an input camera.
`--threads` *integer(=0)*	Set the number of threads to use. 0 means use as many threads as there are cores.
`--tile-size` *integer(=256, 256)*	Image tile size used for multi-threaded processing.
`--no-bigtiff`	Tell GDAL to not create bigtiffs.
`--tif-compress None|LZW|Deflate|Packbits`	TIFF compression method.
`-v | --version`	Display the version of software.
`-h | --help`	Display this help message.

A.42 ipfind

The `ipfind` tool detects interest points (IPs) in images and writes them out to .vwip files. ASP is able to read these files to recover the IPs.

This tool is useful in testing out different IP detection settings and visualizing them (using the option `--debug-image`).

One can pass multiple input images to the tool and they will be processed one after another.

This program works in conjunction with `ipmatch` (section A.43) to match interest points across images.

Usage:

```
ipfind [options] <images>
```

Table A.42: Command-line options for ipfind

Option	Description
`--interest-operator` *string* `(=sift)`	Choose an interest point detector from: sift (default), orb, OBALoG, LoG, Harris, IAGD.
`--descriptor-generator` *string* `(=sift)`	Choose a descriptor generator from: sift (default), orb, sgrad, sgrad2, patch, pca. Some descriptors work only with certain interest point operators.
`--ip-per-image` *integer*	Set the maximum number of IP to find in the whole image. If not specified, use instead –ip-per-tile.
`-t | --tile-size` *integer*	The tile size for processing interest points. Useful when working with large images. Default: 256.
`--ip-per-tile` *integer* `(=250)`	Set the maximum number of IP to find in each tile. Default: 250.
`-g | --gain` *float* `(=1)`	Increasing this number will increase the gain at which interest points are detected. Default: 1.
`--single-scale`	Turn off scale-invariant interest point detection. This option only searches for interest points in the first octave of the scale space. Harris and LoG only.
`--no-orientation`	Turn off rotational invariance.
`--normalize`	Normalize the input. Use for images that have non-standard values such as ISIS cube files.
`--per-tile-normalize`	Individually normalize each processing tile.
`--nodata-radius` *integer* `(=1)`	Don't detect IP within this many pixels of image borders or nodata. Default: 1.
`--output-folder` *string*	Write output files to this location.
`--num-threads` *integer* `(=0)`	Set the number of threads for interest point detection. If set to 0 (default), use the visionworkbench default number of threads.
`-h | --help`	Display this help message.
`-d | --debug-image` *integer* `(=0)`	Write out a low-resolution or full-resolution debug image with interest points on it if the value of this flag is respectively 1 or 2. Default: 0 (do nothing).
`--print-ip` *integer* `(=0)`	Print information for this many detected IP. Default: 0.
`--lowe`	Save the interest points in an ASCII data format that is compatible with the Lowe-SIFT toolchain.

A.43 ipmatch

The `ipmatch` reads in interest points (IPs) from .vwip files and attempts to match them, writing out .match files containing these results. Other ASP tools can read in these files. `ipmatch` also produces debug images which can be useful. Note that this tool does not implement many of the IP matching steps that are used in `stereo_pprc` and `stereo_corr` since it does not use any sensor model information.

If more than two image/vwip sets are passed in, each possible combination of images will be matched.

Usage:

```
ipmatch [options] <image 1> <vwip file 1> <image 2> <vwip file 2> ...
```

Table A.43: Command-line options for ipmatch

Options	Description
`--help|-h`	Display the help message.
`--output-prefix` *string*	Write output files using this prefix.
`--matcher-threshold` *float*	Threshold for the separation between closest and next closest interest points. Default 0.6
`--non-kdtree`	Use a non-KDTree version of the matching algorithm.
`--distance-metric` *string*	Distance metric to use. Choose one of: [L2 (default), Hamming (only for binary types like ORB)].
`--ransac-constraint` *string*	RANSAC constraint type. Choose one of: [similarity, homography, fundamental, or none].
`--inlier-threshold` *float*	RANSAC inlier threshold. Default 10.
`--ransac-iterations` *integer*	Number of RANSAC iterations. Default 100.
`--debug-image|-d`	Set to write out debug images.

A.44 icebridge_kmz_to_csv

A simple tool for use with data from the NASA IceBridge program. Google Earth compatible .kmz files are available at http://asapdata.arc.nasa.gov/dms/missions.html which display the aircraft position at the point when each DMS frame image was captured. This tool exports those positions into a csv file which can be passed into `bundle_adjust` using the following parameters:

```
--camera-positions ../camera_positions.csv --csv-format "1:file 2:lon 3:lat 4:height_above_datum"
```

This may be useful in conjunction with the `camera_solve` tool to allow conversion of camera positions from local to global coordinates.

Usage:

```
> icebridge_kmz_to_csv <input kmz file> <output csv file>
```

A.45 lvis2kml

A simple tool for use with LVIS (Land, Vegetation, and Ice Sensor) lidar data from the NASA IceBridge program. Generates a Google Earth compatible .kml files from either an LVIS data file (.TXT extension) or an LVIS boundary file (.xml extension). Using this tool makes it easy to visualize what region a given LVIS file covers and what the shape of its data looks like. If the output path is not passed to the tool it will generate an output path by appending ".kml" to the input path. This tool requires the simplekml Python package to run. One way to get this is to install the ASP Python tools, described at the end of section 4.5.

Usage:

```
> lvis2kml [options] <input path> [output path]
```

Table A.44: Command-line options for lvis2kml

Option	Description
`-h | --help`	Display this help message.
`--name` *string*	Assign a name to the KML file.
`--color` *string*	Draw plots in one of (red | green | blue)
`--skip` *int(=1)*	When loading a data file, plot only every N-th point. Has no effect on boundary files.

A.46 GDAL Tools

ASP distributes in the `bin` directory the following GDAL tools: `gdalinfo`, `gdal_translate`, `gdalbuildvrt`, `gdalwarp`, and `gdaldem`. These executables are compiled with JPEG2000 and BigTIFF support, and can handle NTF images in addition to most image formats. They can be used to see image statistics, crop and scale images, build virtual mosaics, reproject DEMs, etc. Detailed documentation is available on the GDAL web site, at http://www.gdal.org/.

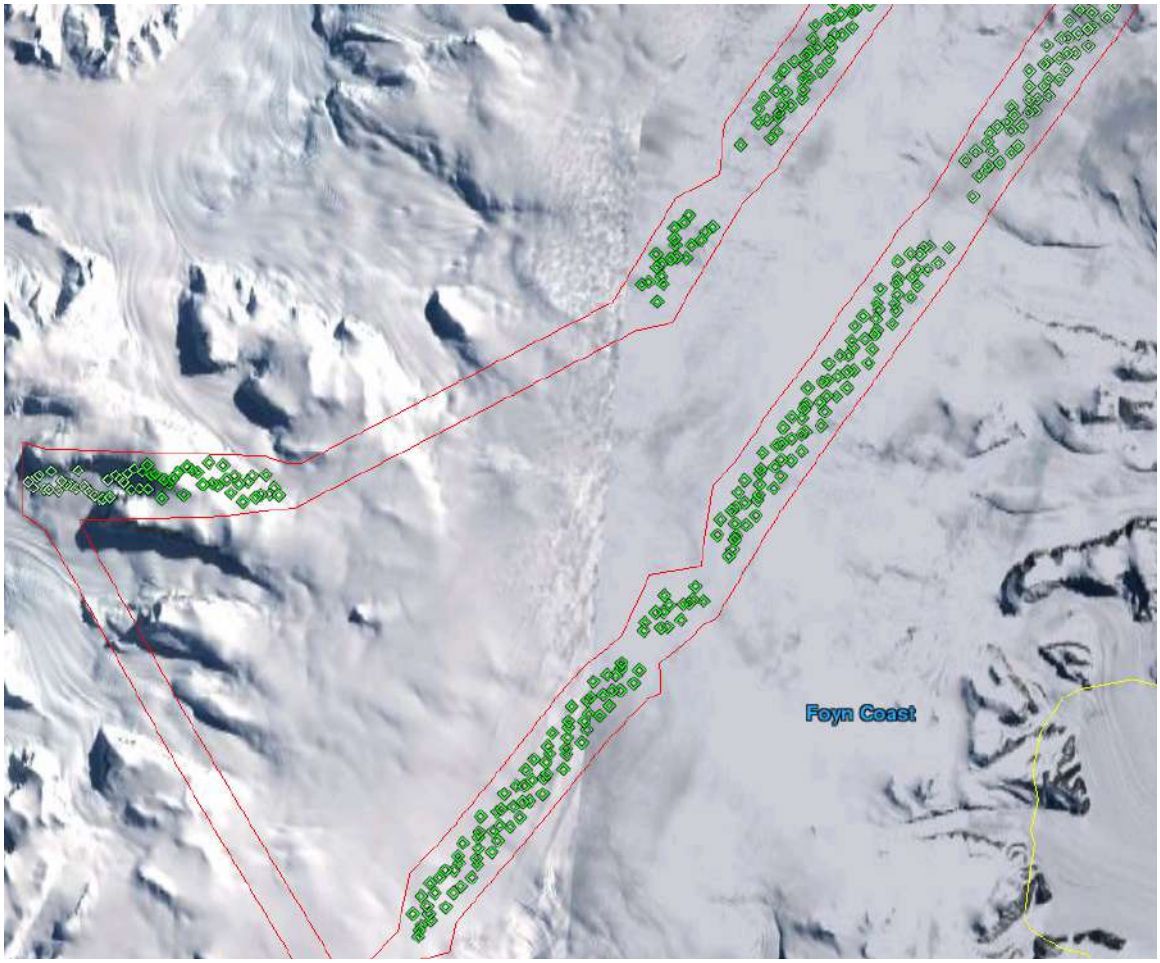


Figure A.3: Example of KML visualizations produced with `lvis2kml`. The output from both the boundary file (red) and the data file (green) with a point skip of 500 are shown in this image. The color saturation of data points is scaled with the elevation such that the points in the file with the least elevation show up as white and the highest points show up as the specified color.

Appendix B

The `stereo.default` File

The `stereo.default` file contains configuration parameters that the `stereo` program uses to process images. The `stereo.default` file is loaded from the current working directory when you run `stereo` unless you specify a different file using the `-s` option. Run `stereo --help` for more information. The extension is not important for this file.

A sample `stereo.default.example` file is included in the `examples/` directory of the Stereo Pipeline software distribution.

As mentioned in section 5.1.5, all the `stereo` parameters can also be specified on the command line.

Listed below are the parameters used by `stereo`, grouped by processing stage.

B.1 Preprocessing

---

alignment-method (= affineepipolar, homography, epipolar, none) **(default = affineepipolar)**

> When `alignment-method` is set to `homography`, `stereo` will attempt to pre-align the images by automatically detecting tie-points between images using a feature based image matching technique. Tie points are stored in a `*.match` file that is used to compute a linear homography transformation of the right image so that it closely matches the left image. Note: the user may exercise more control over this process by using the `ipfind` and `ipmatch` tools.
>
> When `alignment-method` is set to `affineepipolar`, `stereo` will attempt to pre-align the images by detecting tie-points, as earlier, and using those to transform the images such that pairs of conjugate epipolar lines become collinear and parallel to one of the image axes. The effect of this is equivalent to rotating the original cameras which took the pictures.
>
> When `alignment-method` is set to `epipolar`, `stereo` will apply a 3D transform to both images so that their epipolar lines will be horizontal. This speeds of stereo correlation as it greatly reduces the area required for searching.
>
> *Epipolar alignment is only available when performing stereo matches using the pinhole stereo session (i.e. when using* `stereo -t pinhole`*), and cannot be used when processing ISIS images at this time.*

left-image-crop-win xoff yoff xsize ysize
> Do stereo in a sub-region of the left image [default: use the entire image].

right-image-crop-win x**off yoff xsize ysize**
When combined with `left-image-crop-win`, do stereo in given subregions of left and right images. The crop windows can be determined using `stereo_gui`. It is important to note that when both of these are specified, we explicitly crop the input images to these regions, which does not happen when `left-image-crop-win` alone is specified. In that case we use the full images but only restrict the computation to the specified region.

force-use-entire-range (default = false)
By default, the Stereo Pipeline will normalize ISIS images so that their maximum and minimum channel values are ±2 standard deviations from a mean value of 1.0. Use this option if you want to *disable* normalization and force the raw values to pass directly to the stereo correlations algorithms.

For example, if ISIS's `histeq` has already been used to normalize the images, then use this option to disable normalization as a (redundant) pre-processing step.

individually-normalize (default = false)
By default, the maximum and minimum valid pixel value is determined by looking at both images. Normalized with the same "global" min and max guarantees that the two images will retain their brightness and contrast relative to each other.

This option forces each image to be normalized to its own maximum and minimum valid pixel value. This is useful in the event that images have different and non-overlapping dynamic ranges. You can sometimes tell when this option is needed: after a failed stereo attempt one of the rectified images (`*-L.tif` and `*-R.tif`) may be either mostly white or black. Activating this option may correct this problem.

Note: Photometric calibration and image normalization are steps that can and should be carried out beforehand using ISIS's own utilities. This provides the best possible input to the stereo pipeline and yields the best stereo matching results.

ip-per-tile
How many interest points to detect in each 1024^2 image tile (default: automatic determination).

ip-detect-method
What type of interest point detection algorithm to use for image alignment. 0 = Custom OBAloG implementation (default) 1 = SIFT implementation from OpenCV 2 = ORB implementation from OpenCV If the default method does not perform well, try out one of the other two methods.

nodata-value (default = none)
Pixels with values less than or equal to this number are treated as no-data. This overrides the nodata values from input images.

datum (default = WGS_1984)
Set the datum to use with RPC camera models. Options: WGS_1984, D_MOON (1,737,400 meters), D_MARS (3,396,190 meters), MOLA (3,396,000 meters), NAD83, WGS72, and NAD27. Also accepted: Earth (=WGS_1984), Mars (=D_MARS), Moon (=D_MOON).

no-datum
Do not assume a reliable datum exists, such as for irregularly shaped bodies.

epipolar-threshold
Maximum distance in pixels from the epipolar line to search for matches for each interest point. Due to the way ASP finds matches, reducing this value can actually increase the number of interest points detected. If image alignment seems to be working well but you are not getting enough interest points to get a good search range estimate, try setting this value to a small number, perhaps in the low double digits.

ip-inlier-factor (**default = 1.0/15**)
A higher factor will result in more interest points, but perhaps also more outliers. It is important to note that this parameter overlaps somewhat in scope and effect with `epipolar-threshold` and sometimes not both are active. It is suggested to experiment with both, as well as with `ip-uniqueness-threshold` below, which has a different justification but also somewhat similar effects.

ip-uniqueness-threshold (**default = 0.7**)
A higher threshold will result in more interest points, but perhaps less unique ones.

ip-triangulation-max-error ***double***
When matching IP, filter out any pairs with a triangulation error higher than this.

ip-num-ransac-iterations ***int(=100)***
How many RANSAC iterations to do in interest point matching.

force-reuse-match-files
Force reusing the match files even if older than the images or cameras.

skip-rough-homography
Skip the step of performing datum-based rough homography if it fails.

B.2 Correlation

prefilter-mode (= 0,1,2) **(default = 2)**
This selects the pre-processing filter to be used to prepare imagery before it is fed to the initialization stage of the pipeline.

0 - None

1 - Subtracted Mean - This takes a preferably large Gaussian kernel and subtracts its value from the input image. This effectively reduces low frequency content in the image. The result is correlation that is immune to translations in image intensity.

2 - LoG Filter - Takes the Laplacian of Gaussian of the image, This provides some immunity to differences in lighting conditions between a pair of images by isolating and matching on blob features in the image.

For all of the modes above, the size of the filter kernel is determined by the `prefilter-kernel-width` parameter below.

The choice of pre-processing filter must be made with thought to the cost function being used (see `cost-mode`, below). LoG filter preprocessing provides good immunity to variations in lighting conditions and is usually the recommended choice.

prefilter-kernel-width (*float*) **(default = 1.4)**
This defines the diameter of the Gaussian convolution kernel used for the preprocessing modes 1 and 2 above. A value of 1.4 works well for LoG and 25-30 works well for Subtracted Mean.

corr-seed-mode (=0,1,2,3) (default = 1)
This integer parameter selects a strategy for how to solve for the low-resolution integer correlation disparity, which is used to seed the full-resolution disparity later on.

0 - None - Don't calculate a low-resolution variant of the disparity image. The search range provided by `corr-search` is used directly in computing the full-resolution disparity.

1 - Low-resolution disparity from stereo - Calculate a low-resolution version of the disparity from the integer correlation of subsampled left and right images. The low-resolution disparity will be used to narrow down the search range for the full-resolution disparity.

This is a useful option despite the fact that our integer correlation implementation does indeed use a pyramid approach. Our implementation cannot search infinitely into lower resolutions due to its independent and tiled nature. This low-resolution disparity seed is a good hybrid approach.

2 - Low-resolution disparity from an input DEM - Use a lower-resolution DEM together with an estimated value for its error to compute the low-resolution disparity, which will then be used to find the full-resolution disparity as above. These quantities can be specified via the options `disparity-estimation-dem` and `disparity-estimation-dem-error` respectively. This option is not compatible with map projected input images.

3 - Disparity from full-resolution images at a sparse number of points. This is an advanced option for terrain having snow and no large-scale features. It is described in section 4.5.

For large images, bigger than MOC-NA, using the low-resolution disparity seed is a definitive plus. Smaller images such as Cassini ISS or MER images should just shut this option off to save storage space.

corr-sub-seed-percent (*float*) **(default=0.25)**
When using `corr-seed-mode 1`, the solved-for or user-provided search range is grown by this factor for the purpose of computing the low-resolution disparity.

min-num-ip (*integer*) **(default = 20)**
Automatic search range estimation will quit if at least this many interest points are not detected.

cost-mode (= 0,1,2,3,4) (default = 2)
This defines the cost function used during integer correlation. Squared difference is the fastest cost function. However it comes at the price of not being resilient against noise. Absolute difference is the next fastest and is a better choice. Normalized cross correlation is the slowest but is designed to be more robust against image intensity changes and slight lighting differences. Normalized cross correlation is about 2x slower than absolute difference and about 3x slower than squared difference. The census transform [159] and ternary census transform [60] can only be used with the SGM correlator. See section 7.2.4 for details.

0 - absolute difference

1 - squared difference

2 - normalized cross correlation

3 - census transform

4 - ternary census transform

corr-kernel (*integer integer*) **(default = 25 25)**
These option determine the size (in pixels) of the correlation kernel used in the initialization step. A different size can be set in the horizontal and vertical directions, but square correlation kernels are almost always used in practice.

corr-search (*integer integer integer integer*)
These parameters determine the size of the initial correlation search range. The ideal search range depends on a variety of factors ranging from how the images were pre-aligned to the resolution and

range of disparities seen in a given image pair. This search range is successively refined during initialization, so it is often acceptable to set a large search range that is guaranteed to contain all of the disparities in a given image. However, setting tighter bounds on the search can sometimes reduce the number of erroneous matches, so it can be advantageous to tune the search range for a particular data set.

If this option is not provided, `stereo` will make an attempt to guess its search range using interest points.

These four integers define the minimum horizontal and vertical disparity and then the maximum horizontal and vertical disparity.

corr-search-limit (*integer integer integer integer*)
Set these parameters to constrain the search range that `stereo` automatically computes when `corr-search` is not set. This setting is useful when you have a good idea of the alignment quality in the vertical direction but not in the horizontal direction. For example, when using pinhole frame cameras with epipolar alignment the actual vertical search range may be much smaller than the automatically computed search range.

elevation-limit (*float float*)
Notify ASP that all elevations are expected to fall in this range relative to the datum. Currently only used to restrict the search range estimate in nadir epipolar alignment cases.

corr-max-levels (*integer*) **(default = 5)**
The maximum number of additional (lower) resolution levels to use when performing integer correlation. Setting this value to zero just performs correlation at the native resolution.

xcorr-threshold (*integer*) **(default = 2)**
Integer correlation to a limited sense performs a correlation forward and backwards to double check its result. This is one of the first filtering steps to insure that we have indeed converged to a global minimum for an individual pixel. The `xcorr-threshold` parameter defines an agreement threshold in pixels between the forward and backward result.

Optionally, this parameter can be set to a negative number. This will signal the correlator to only use the forward correlation result. This will drastically improve speed at the cost of additional noise.

min-xcorr-level (*integer*) **(default = 0)**
When using the cross-correlation check controlled by xcorr-threshold, this parameter sets the minimum pyramid resolution level that the check will be performed at. By default the check will be performed at every resolution level but you may wish to increase this value to save time by not doubling up on processing the largest levels.

Currently this feature is not enabled when using the default block-matching correlation method. In that case cross correlation is only ever performed on the last resolution level.

remove-outliers-by-disparity-params (*double double*) **(default = 100 3)**
Outlier removal based on the disparity of interest points (difference between right and left pixel), when estimating the disparity search range. For example, the 10% and 90% percentiles of disparity are computed, and this interval is made three times bigger. Interest points whose disparity fall outside the expanded interval are removed as outliers. Instead of the default 100 and 3 one can specify pct and factor, without quotes.

rm-quantile-percentile (*double*) **(default = 0.85)**
See rm-quantile-multiple for details.

rm-quantile-multiple (*double*) **(default = -1)**
Used for filtering disparity values in D_sub. Disparities greater than MULTIPLE*PERCENTILE (of the histogram) will be discarded. If this value is set greater than zero, this filtering method will be used instead of the method using the values RM_MIN_MATCHES and RM_THRESHOLD. This method will help filter out clusters of pixels which are too large to be filtered out by the neighborhood method but that have disparities significantly greater than the rest of the image.

use-local-homography (**default = false**)
This flag, if provided, enables using local homography during correlation, as described in Section 7.2.3.

corr-timeout (*integer*) **(default = 1800)**
Correlation timeout for an image tile, in seconds. A non-positive value will result in no timeout enforcement. A value of 600 seconds should be sufficient in most cases.

stereo-algorithm (**default = 0**)
Use this setting to switch between the different integer correlation options supported by ASP.

- **0 - Local Search Window** - The default option searches for the best match for a local window around each window using the selected cost mode. This is the fastest algorithm and works well for similar images with good texture coverage.
- **1 - Semi-Global Matching** - Use the popular SGM algorithm [58]. This algorithm is slow and has high memory requirements but it performs better in images with less texture. See section 7.2.4 for important details on using this algorithm.
- **2 - Smooth Semi-Global Matching** - Uses the MGM variant of the SGM algorithm [33] to reduce high frequency artifacts in the output image at the cost of increased run time. See section 7.2.4 for important details on using this algorithm.
- **3 - MGM Final** - Use MGM on the final resolution level and SGM on preceding resolution levels. This produces a result somewhere in between the pure SGM and MGM options.

corr-blob-filter (*integer*) **(default = 0)**
Set to apply a blob filter in each level of pyramidal integer correlation. When the correlator fails it often leaves "islands" of erroneous disparity results. Using this blob filter to remove them cleans up the final stereo output and can even reduce processing times by preventing the correlator from searching at large, incorrect disparity amounts. The value provided is the size of blobs in pixels that will be removed at the full image resolution.

corr-tile-size (*integer*) **(default = 1024)**
Manually specifies the size of image tiles used by the correlator for multi-threaded processing. Typically there is no need to adjust this value but it is very important when using semi-global matching. See section 7.2.4 for details. This value must be a multiple of 16.

sgm-collar-size (*integer*) **(default = 512)**
Specify the size of a region of additional processing around each correlation tile when using SGM or MGM processing. This helps reduce seam artifacts at tile borders when processing an image that needs to be broken up into tiles at the cost of additional processing time. This has no effect if the entire image can fit in one tile.

sgm-search-buffer (*integer integer*) **(default = 4 4)**
This option determines the size (in pixels) searches around the expected disparity location in successive levels of the correlation pyramid. A smaller value will decrease run time and memory usage but will increase the chance of blunders. It is not recommended to reduce either value below 2.

corr-memory-limit-mb (*integer*) **(default = 6144)**

Restrict the amount of memory used by the correlation step to be slightly above this value. This only really affects SGM/MGM which use a pair of large memory buffer in their computation. The total memory usage of these buffers is compared to this limit, and if it is greater then smaller search ranges will be used for uncertain pixels in order to reduce memory usage. If the required memory is still over this limit then the program will error out. The unit is in megabytes.

B.3 Subpixel Refinement

subpixel-mode (*integer*) **(default = 1)**

This parameter selects the subpixel correlation method. Parabola subpixel is very fast but will produce results that are only slightly more accurate than those produced by the initialization step. Bayes EM (mode 2) is very slow but offers the best quality. When tuning `stereo.default` parameters, it is expedient to start out using parabola subpixel as a "draft mode." When the results are looking good with parabola subpixel, then they will look even better with subpixel mode 2. For inputs with little noise, the affine method (subpixel mode 3) may produce results equivalent to Bayes EM in a shorter time. Phase correlation (subpixel mode 4) is uses a frequency domain technique. It is slow and is best may not produce better results than mode 2 but it may work well in some situations with flat terrain.

Subpixel modes 5 and 6 are experimental. Modes 7-12 are only used as part of SGM/MGM correlation. These are much faster than subpixel modes 2-4 and if selected (with SGM/MGM) will be the only subpixel mode performed. They interpolate between the SGM/MGM integer results and should produce reasonable values. The default blend method for SGM/MGM is a custom algorithm that should work well but the you may find that one of the other options is better for your data.

Subpixel modes 1-4 can be used in conjunction with SGM/MGM. In this case subpixel mode 12 will be used first, followed by the selected subpixel mode. Depending on your data this may produce better results than using just the SGM/MGM only methods. You may get bad artifacts combining mode 1 with SGM/MGM.

0 - no subpixel refinement

1 - parabola fitting

2 - affine adaptive window, Bayes EM weighting

3 - affine window

4 - phase correlation

5 - Lucas-Kanade method (experimental)

6 - affine adaptive window, Bayes EM with Gamma Noise Distribution (experimental)

7 - SGM None

8 - SGM linear

9 - SGM Poly4

10 - SGM Cosine

11 - SGM Parabola

12 - SGM Blend

For a visual comparison of the quality of these subpixel modes, refer back to Chapter:7.

subpixel-kernel (*integer integer*) **(default = 35 35)** Specify the size of the horizontal and vertical size (in pixels) of the subpixel correlation kernel. It is advantageous to keep this small for parabola fitting in order to resolve finer details. However for the Bayes EM methods, keep the kernel slightly larger. Those methods weight the kernel with a Gaussian distribution, thus the effective area is small than the kernel size defined here.

B.4 Filtering

filter-mode (*integer*) **(default = 1)**
This parameter sets the filter mode. Three modes are supported as described below. Here, by neighboring pixels for a current pixel we mean those pixels within the window of half-size of `rm-half-kernel` centered at the current pixel.

0 - No filtering.

1 - Filter by discarding pixels at which disparity differs from mean disparity of neighbors by more than `max-mean-diff`.

2 - Filter by discarding pixels at which percentage of neighboring disparities that are within `rm-threshold` of current disparity is less than `rm-min-matches`.

rm-half-kernel (*integer integer*) **(default = 5 5)**
This setting adjusts the behavior of an outlier rejection scheme that "erodes" isolated regions of pixels in the disparity map that are in disagreement with their neighbors.

The two parameters determine the size of the half kernel that is used to perform the automatic removal of low confidence pixels. A 5×5 half kernel would result in an 11×11 kernel with 121 pixels in it.

max-mean-diff (*integer*) **(default = 3)**
This parameter sets the *maximum difference* between the current pixel disparity and the mean of disparities of neighbors in order for a given disparity value to be retained (for `filter-mode` 1).

rm-min-matches (*integer*) **(default = 60)**
This parameter sets the *percentage* of neighboring disparity values that must fall within the inlier threshold in order for a given disparity value to be retained (for `filter-mode` 2).

rm-threshold (*double*) **(default = 3)**
This parameter sets the inlier threshold for the outlier rejection scheme. This option works in conjunction with RM_MIN_MATCHES above. A disparity value is rejected if it differs by more than RM_THRESHOLD disparity values from RM_MIN_MATCHES percent of pixels in the region being considered (for `filter-mode` 2).

rm-cleanup-passes (*integer*) **(default = 1)**
Select the number of outlier removal passes that are carried out. Each pass will erode pixels that do not match their neighbors. One pass is usually sufficient.

median-filter-size (*integer*) **(default = 0)**
Apply a median filter of the selected kernel size to the subpixel disparity results. This option can only be used if `rm-cleanup-passes` is set to zero.

texture-smooth-size (*integer*) **(default = 0)**
Apply an adaptive filter to smooth the disparity results inversely proportional to the amount of texture present in the input image. This value sets the maximum size of the smoothing kernel used (in pixels). This option can only be used if `rm-cleanup-passes` is set to zero.

texture-smooth-scale (*float*) **(default = 0.15)**
Used in conjunction with `texture-smooth-size`, this value helps control the regions of the image that will be smoothed. A larger value will result in more smoothing being applied to more of the image. A smaller value will leave high-texture regions of the image unsmoothed.

enable-fill-holes (default = false)

Enable filling of holes in disparity using an inpainting method. Obsolete. It is suggested to use instead point2dem's analogous functionality.

fill-holes-max-size (*integer*) **(default = 100,000)**
Holes with no more pixels than this number should be filled in.

edge-buffer-size (*integer*) **(default = -1)**
Crop to be applied around image borders during filtering. If not set, default to subpixel kernel size.

erode-max-size (*integer*) **(default = 0)**
Isolated blobs with no more pixels than this number should be removed.

B.5 Post-Processing (Triangulation)

near-universe-radius (*float*) **(default = 0.0)**

far-universe-radius (*float*) **(default = 0.0)**

These parameters can be used to remove outliers from the 3D triangulated point cloud. The points that will be kept are those whose distance from the universe center (see below) is between `near-universe-radius` and `far-universe-radius`, in meters.

universe-center (**default = none**)
Defines the reference location to use when filtering the output point cloud using the above near and far radius options. The available options are:

None - Disable filtering.

Camera - Use the left camera's center as the universe center.

Zero - Use the center of the planet as the universe center.

bundle-adjust-prefix (*string*)
Use the camera adjustments obtained by previously running bundle_adjust with this output prefix.

min-triangulation-angle (*double*)
The minimum angle, in degrees, at which rays must meet at a triangulated point to accept this point as valid. The internal default is somewhat less than 1 degree.

point-cloud-rounding-error (*double*)

How much to round the output point cloud values, in meters (more rounding means less precision but potentially smaller size on disk). The inverse of a power of 2 is suggested. Default: $1/2^{10}$ meters (about 1mm) for Earth and proportionally less for smaller bodies.

save-double-precision-point-cloud (**default = false**)

Save the final point cloud in double precision rather than bringing the points closer to origin and saving as float (marginally more precision at twice the storage).

compute-error-vector (**default = false**)

When writing the output point cloud, save the 3D triangulation error vector (the vector between the closest points on the rays emanating from the two cameras), rather than just its length. In this case, the point cloud will have 6 bands (storing the triangulation point and triangulation error vector) rather than the usual 4. When invoking `point2dem` on this 6-band point cloud and specifying the `--errorimage` option, the error image will contain the three components of the triangulation error vector in the North-East-Down coordinate system.

The next several parameters are used for jitter correction for Digital Globe imagery. A usage tutorial is given in section 4.4.

image-lines-per-piecewise-adjustment (*integer*) **(default = 0)** A positive value, e.g., 1000, will turn on using piecewise camera adjustments to help reduce jitter effects. Use one adjustment per this many image lines.

piecewise-adjustment-percentiles (*float float*) **(default = 5 95)** A narrower range will place the piecewise adjustments for jitter correction closer together and further from the first and last lines in the image.

piecewise-adjustment-interp-type (*integer*) **(default = 1)** How to interpolate between adjustments. [1 Linear, 2 Using Gaussian weights]

piecewise-adjustment-camera-weight (*float*) **(default = 1.0)** The weight to use for the sum of squares of adjustments component of the cost function. Increasing this value will constrain the adjustments to be smaller.

num-matches-for-piecewise-adjustment (*integer*) **(default = 90000)** How many matches among images to create based on the disparity for the purpose of solving for jitter using piecewise adjustment.

These last two options are used internally.

compute-piecewise-adjustments-only (**default = false**)
Compute the piecewise adjustments as part of jitter correction, and then stop.

skip-computing-piecewise-adjustments (**default = false**)
Skip computing the piecewise adjustments for jitter, they should have been done by now.

Appendix C

Guide to Output Files

The `stereo` tool generates a variety of intermediate files that are useful for debugging. These are listed below, along with brief descriptions about the contents of each file. Note that the prefix of the filename for all of these files is dictated by the final command line argument to `stereo`. Run `stereo --help` for details.

***.vwip** - image feature files

If `alignment-method` is not `none`, the Stereo Pipeline will automatically search for image features to use for tie-points. Raw image features are stored in `*.vwip` files; one per input image. For example, if your images are `left.cub` and `right.cub` you'll get `left.vwip` and `right.vwip`. Note: these files can also be generated by hand (and with finer grained control over detection algorithm options) using the `ipfind` utility.

***.match** - image to image tie-points

The match file lists a select group of unique points out of the previous `.vwip` files that have been identified and matched in a pair of images. For example, if your images are `left.cub` and `right.cub` you'll get a `left__right.match` file.

The `.vwip` and `.match` files are meant to serve as cached tie-point information, and they help speed up the pre-processing phase of the Stereo Pipeline: if these files exist then the `stereo` program will skip over the interest point alignment stage and instead use the cached tie-points contained in the `*.match` files. In the rare case that one of these files did get corrupted or your input images have changed, you may want to delete these files and allow `stereo` to regenerate them automatically. This is also recommended if you have upgraded the Stereo Pipeline software.

Both `.vwip` and `.match` files can be visualized in `stereo_gui`.

***-L.tif -** rectified left input image

The left input image of the stereo pair, saved after the pre-processing step. This image may be normalized, but should otherwise be identical to the original left input image.

***-R.tif -** rectified right input image

Right input image of the stereo pair, after the pre-processing step. This image may be normalized and possibly translated, scaled, and/or rotated to roughly align it with the left image, but should otherwise be identical to the original right input image.

***-lMask.tif** - mask for left rectified image

***-rMask.tif** - mask for right rectified image

These files contain binary masks for the input images. These are used throughout the stereo process to mask out pixels where there is no input data.

***-align-L.exr** - left pre-alignment matrix

***-align-R.exr** - right pre-alignment matrix

The 3×3 affine transformation matrices that are used to warp the left and right images to roughly align them. These files are only generated if `alignment-method` is not `none` in the `stereo.default` file. Normally, a single transform is enough to warp one image to another (for example, the right image to the left). The reason we use two transforms is the following: after the right image is warped to the left, we would like to additionally transform both images so that the origin (0, 0) in the left image would correspond to the same location in the right image. This will somewhat improve the efficiency of subsequent processing.

***-D.tif** - disparity map after the disparity map initialization phase

This is the disparity map generated by the correlation algorithm in the initialization phase. It contains integer values of disparity that are used to seed the subsequent sub-pixel correlation phase. It is largely unfiltered, and may contain some bad matches.

Disparity map files are stored in OpenEXR format as 3-channel, 32-bit floating point images. (Channel 0 = horizontal disparity, Channel 1 = vertical disparity, and Channel 2 = good pixel mask)

***-RD.tif -** disparity map after sub-pixel correlation

This file contains the disparity map after sub-pixel refinement. Pixel values now have sub-pixel precision, and some outliers have been rejected by the sub-pixel matching process.

***-F-corrected.tif** - intermediate data product

Only created when `alignment-method` is not `none`. This is `*-F.tif` with effects of interest point alignment removed.

***-F.tif** - filtered disparity map

The filtered, sub-pixel disparity map with outliers removed (and holes filled with the inpainting algorithm if `FILL_HOLES` is on). This is the final version of the disparity map.

***-GoodPixelMap.tif** - map of good pixels

An image showing which pixels were matched by the stereo correlator (gray pixels), and which were filled in by the hole filling algorithm (red pixels).

***-PC.tif** - point cloud image

The point cloud image is generated by the triangulation phase of Stereo Pipeline. Each pixel in the point cloud image corresponds to a pixel in the left input image (*-L.tif). The point cloud has four channels, the first three are the Cartesian coordinates of each point, and the last one has the intersection error of the two rays which created that point (the intersection error is the closest distance between rays). By default, the origin of the Cartesian coordinate system being used is a point in the neighborhood of the point cloud. This makes the values of the points in the cloud relatively small, and we save them in single precision (32 bits). This origin is saved in the point cloud as well using the tag POINT_OFFSET in the GeoTiff header. To output point clouds using double precision with the origin at the planet center, call `stereo_tri` with the option `--save-double-precision-point-cloud`. This can effectively double the size of the point cloud.

All these images that are single-band can be visualized in `stereo_gui` (section A.2). The disparities can be first split into the individual horizontal and vertical disparity files using `disparitydebug`, then they can be seen in this viewer as well.

If the input images are map-projected (georeferenced) and the alignment method is `none`, all the output images listed above, will also be georeferenced, and hence can be overlayed in `stereo_gui` on top of the input imagery (the outputs of `disparitydebug` will then be georeferenced as well).

The point cloud file saves the datum (and projection if available) inferred from the input images, regardless of whether these images are map-projected or not.

The `point2mesh` and `point2dem` programs can be used to convert the point cloud to formats that are easier to visualize.

***-stereo.default** - backup of the Stereo Pipeline settings file

This is a copy of the `stereo.default` file used by `stereo`. It is stored alongside the output products as a record of the settings that were used for this particular stereo processing task.

Appendix D

Frame Camera Models

Ames Stereo Pipeline supports a generic Pinhole camera model with several lens distortion models which cover common calibration methods, and also the somewhat more complicated panoramic (*optical bar*) camera model.

D.1 Pinhole Models

D.2 Overview

The generic Pinhole model uses the following parameters:

- *fu* = The focal length in horizontal pixel units.
- *fv* = The focal length in vertical pixel units.
- *cu* = The horizontal offset of the principal point of the camera in the image plane in pixel units, from 0,0.
- *cv* = The vertical offset of the principal point of the camera in the image plane in pixel units, from 0,0.
- *pitch* = The size of each pixel in the units used to specify the four parameters listed above. This will usually either be 1.0 if they are specified in pixel units or alternately the size of a pixel in millimeters.

The focal length is sometimes known as the *principal distance*. The value *cu* is usually approximately half the image width in pixels times the pitch, while *cv* is often the image height in pixels times the pitch, though there are situations when these can be quite different.

A few sample Pinhole models are shown later in the text. The underlying mathematical model is described in section D.2.2.

Along with the basic Pinhole camera parameters, a lens distortion model can be added. Note that the units used in the distortion model must match the units used for the parameters listed above. For example, if the camera calibration was performed using units of millimeters the focal lengths etc. must be given in units of millimeters and the pitch must be equal to the size of each pixel in millimeters. The following lens distortion models are currently supported:

- **Null** = A placeholder model that applies no distortion.
- **Tsai** = A common distortion model similar to the one used by OpenCV and THEIA. This model uses the following parameters:

  K1, K2 = Radial distortion parameters.
  P1, P2 = Tangential distortion parameters.

  The following equations describe the distortion, starting with the undistorted pixel (Px, Py):

  $$(x, y) = \left(\frac{Px - cu}{fu}, \frac{Py - cv}{fv}\right)$$
  $$r^2 = x^2 + y^2$$
  $$x(distorted) = x\left(K_1 r^2 + K_2 r^4 + 2P_1 y + P_2\left(\frac{r^2}{x} + 2x\right)\right)$$
  $$y(distorted) = y\left(K_1 r^2 + K_2 r^4 + 2P_2 x + P_1\left(\frac{r^2}{y} + 2y\right)\right)$$

  References:

  Roger Tsai, A Versatile Camera Calibration Technique for a High-Accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses

  Note that this model uses normalized pixel units.

- **Adjustable Tsai** = A variant of the Tsai model where any number of K terms and a skew term (alpha) can be used. Can apply the AgiSoft Lens calibration parameters.
- **Brown-Conrady** = An older model based on a centering angle.

  This model uses the following parameters:

  K1, K2, K3 = Radial distortion parameters.
  P1, P2 = Tangential distortion parameters.
  xp, yp = Principal point offset.
  phi = Tangential distortion angle in radians.

  The following equations describe the distortion:

  $$x = x(distorted) - xp$$
  $$y = y(distorted) - yp$$
  $$r^2 = x^2 + y^2$$
  $$dr = K_1 r^3 + K_2 r^5 + K_3 r^7$$
  $$x(undistorted) = x + x\frac{dr}{r} - (P_1 r^2 + P_2 r^4)\sin(phi)$$

$$y(undistorted) = y + y\frac{dr}{r} + (P_1 r^2 + P_2 r^4)\cos(phi)$$

Note that this model uses non-normalized pixel units, so they are in mm.

References:

Decentering Distortion of Lenses - D.C. Brown, Photometric Engineering, pages 444-462, Vol. 32, No. 3, 1966

Close-Range Camera Calibration - D.C. Brown, Photogrammetric Engineering, pages 855-866, Vol. 37, No. 8, 1971

- **Photometrix** = A model matching the conventions used by the Australis software from Photometrix.

  K1, K2, K3 = Radial distortion parameters.
  P1, P2 = Tangential distortion parameters.
  xp, yp = Principal point offset.
  B1, B2 = Unused parameters.

  The following equations describe the distortion:

  $$x = x(distorted) - xp$$
  $$y = y(distorted) - yp$$
  $$r^2 = x^2 + y^2$$
  $$dr = K_1 r^3 + K_2 r^5 + K_3 r^7$$
  $$x(undistorted) = x + x\frac{dr}{r} + P_1(r^2 + 2x^2) + 2P_2 xy$$
  $$y(undistorted) = y + y\frac{dr}{r} + P_2(r^2 + 2y^2) + 2P_1 xy$$

  Note that this model uses non-normalized pixel units, so they are in mm.

- **RPC** = A rational polynomial coefficient model.
  In this model, one goes from distorted coordinates (x, y) to undistorted coordinates via the formula

  $$x(undistorted) = \frac{P_1(x, y)}{Q_1(x, y)}$$
  $$y(undistorted) = \frac{P_2(x, y)}{Q_2(x, y)}$$

  The functions in the numerator and denominator are polynomials in x and y with certain coefficients. The degree of polynomials can be any positive integer.

  RPC distortion models can be generated as approximations to other pre-existing models with the tool `convert_pinhole_model` (section A.40).

  This tool also creates RPC to speed up the reverse operation, of going from undistorted to distorted pixels, and those polynomial coefficients are also saved as part of the model.

D.2.1 File Formats

ASP Pinhole model files are written in an easy to work with plain text format using the extension `.tsai`. A sample file is shown below.

```
VERSION_4
PINHOLE
fu = 28.429
fv = 28.429
cu = 17.9712
cv = 11.9808
u_direction = 1  0  0
v_direction = 0  1  0
w_direction = 0  0  1
C = 266.943 -105.583 -2.14189
R = 0.0825447 0.996303 -0.0238243 -0.996008 0.0832884 0.0321213 0.0339869 0.0210777 0.9992
pitch = 0.0064
Photometrix
xp = 0.004
yp = -0.191
k1 = 1.31024e-04
k2 = -2.05354e-07
k3 = -5.28558e-011
p1 = 7.2359e-006
p2 = 2.2656e-006
b1 = 0.0
b2 = 0.0
```

The first half of the file is the same for all Pinhole models:

- `VERSION_X` = A header line used to track the format of the file.
- `PINHOLE` = The type of camera model, so that other types can be stored with the .tsai extension.
- `fu, fv, cu, cv` = The first four intrinsic parameters described in the previous section.
- `u, v, and w_direction` = These lines allow an additional permutation of the axes of the camera coordinates. By default, the positive column direction aligns with x, the positive row direction aligns with y, and downward into the image aligns with z.
- `C` = The location of the camera center, usually in the geocentric coordinate system (GCC/ECEF).
- `R` = The rotation matrix describing the camera's absolute pose in the coordinate system.
- `pitch` = The pitch intrinsic parameter described in the previous section.

The second half of the file describes the lens distortion model being used. The name of the distortion model appears first, followed by a list of the parameters for that model. The number of parameters may be different for each distortion type. Samples of each format are shown below:

- **Null**

```
NULL
```

- **Tsai**

```
TSAI
k1 = 1.31024e-04
k2 = -2.05354e-07
p1 = 0.5
p2 = 0.4
```

- **Adjustable Tsai**

```
AdjustableTSAI
Radial Coeff: Vector3(1.31024e-04, 1.31024e-07, 1.31024e-08)
Tangential Coeff: Vector2(-2.05354e-07, 1.05354e-07)
Alpha: 0.4
```

- **Brown-Conrady**

```
BrownConrady
xp = 0.5
yp = 0.4
k1 = 1.31024e-04
k2 = -2.05354e-07
k3 = 1.31024e-08
p1 = 0.5
p2 = 0.4
phi = 0.001
```

- **Photometrix**

```
Photometrix
xp = 0.004
yp = -0.191
k1 = 1.31024e-04
k2 = -2.05354e-07
k3 = -5.28558e-011
p1 = 7.2359e-006
p2 = 2.2656e-006
b1 = 0.0
b2 = 0.0
```

- **RPC**

```
RPC
rpc_degree = 1
image_size = 5760 3840
distortion_num_x   = 0 1 0
distortion_den_x   = 1 0 0
distortion_num_y   = 0 0 1
distortion_den_y   = 1 0 0
```

```
undistortion_num_x = 0 1 0
undistortion_den_x = 1 0 0
undistortion_num_y = 0 0 1
undistortion_den_y = 1 0 0
```

This sample RPC lens distortion model represents the case of no distortion, when the degree of the polynomials is 1, and both the distortion and undistortion formula leave the pixels unchanged, that is, the distortion transform is

$$(x, y) \to (x, y) = \left(\frac{0 + 1 \cdot x + 0 \cdot y}{1 + 0 \cdot x + 0 \cdot y}, \frac{0 + 0 \cdot x + 1 \cdot y)}{1 + 0 \cdot x + 0 \cdot y} \right).$$

In general, if the degree of the polynomials is n, there are $2(n + 1)(n + 2)$ coefficients. The zero-th degree coefficients in the denominator are always set to 1.

For several years Ames Stereo Pipeline generated Pinhole files in the binary `.pinhole` format. That format is no longer supported.

Also in the past Ames Stereo Pipeline has generated a shorter version of the current file format, also with the extension `.tsai`, which only supported the TSAI lens distortion model. Existing files in that format can still be used by ASP.

Note that the `orbitviz` tool can be useful for checking the formatting of `.tsai` files you create and to estimate the position and orientation. To inspect the orientation use the optional `.dae` model file input option and observe the rotation of the 3D model.

D.2.2 How the Pinhole model is applied

As mentioned in section D.2.1, the ASP Pinhole models store the focal length as fu and fv, the optical center (cu, cv) (which is the pixel location at which the ray coming from the center of the camera is perpendicular to the image plane, in units of the pixel pitch), the vector C which is the camera center in world coordinates system, and the matrix R that is the transform from camera to world coordinates.

To go in more detail, a point Q in the camera coordinate system gets transformed to a point P in the world coordinate system via:

$$P = RQ + C$$

Hence, to go from world to camera coordinates one does:

$$Q = R^{-1}P - R^{-1}C$$

From here the pixel location is computed as:

$$\frac{1}{p}\left(fu\frac{Q_1}{Q_3} + cu, fv\frac{Q_2}{Q_3} + cv \right)$$

where p is the pixel pitch.

D.3 Panoramic Camera Model

ASP also supports a simple panoramic/optical bar camera model for use with images such as the declassified Corona KH4 and Keyhole KH9 images. It implements the model from [132] with the motion compensation from [139].

Such a model looks as follows:

```
VERSION_4
OPTICAL_BAR
image_size = 110507 7904
image_center = 55253.5 3952
pitch = 7.0e-06
f = 0.61000001430511475
scan_time = 0.5
forward_tilt = -0.261799
iC = -1047140.9611702315 5508464.4323527571 3340425.4078937685
iR = -0.96635634448923746 -0.16918164442572045 0.1937343197650008 -0.23427205529446918 0.2680408426
speed = 7700
mean_earth_radius = 6371000
mean_surface_elevation = 4000
motion_compensation_factor = 1.0
scan_dir = left
```

Here, the image size and center are given in pixels, with the width followed by the height. The pixel pitch and focal length `f` are in meters. The scan time is seconds, the forward tilt is in radians, the speed is in meters per second, and the Earth radius and mean surface elevation are in meters. The initial camera center `iC` is in meters, and the rotation matrix `iR` stores the absolute pose. `scan_dir` must be set to 'left' or 'right'. `scan_dir` and `use_motion_compensation` control how the sensor model accounts accounts for the motion of the satellite during the image scan. Without the benefit of detailed historical documents it may require experimentation to find the good initial values for these cameras. When using `bundle_adjust`, the intrinsic parameters that are solved for are `speed`, `motion_compensation_factor`, and `scan_time`.

Appendix E

Papers that used ASP

These works have made use of the Ames Stereo Pipeline to produce their results or enable their studies. If something is missing, let us know!

1. R. A. Beyer, B. Archinal, Y. Chen, K. Edmundson, D. Harbour, E. Howington-Kraus, R. Li, A. McEwen, S. Mattson, Z. Moratto, J. Oberst, M. Rosiek, F. Scholten, T. Tran, M. Robinson, and LROC Team. LROC Stereo Data–Results of Initial Analysis. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 41*, number #2678. Lunar and Planetary Institute, Houston, March 2010

2. T. R. Watters, M. S. Robinson, R. A. Beyer, J. F. Bell, M. E. Pritchard, M. E. Banks, E. P. Turtle, N. R. Williams, and LROC Team. Lunar Thrust Faults: Implications for the Thermal History of the Moon. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 41*, number #1863. Lunar and Planetary Institute, Houston, March 2010

3. C. B. Phillips, R. A. Beyer, F. Nimmo, J. H. Roberts, and G. Robuchon. Crater Relaxation and Stereo Imaging of the Icy Satellites of Jupiter and Saturn. *AGU Fall Meeting Abstracts*, (#P21B-1596), December 2010

4. N. P. Hammond, C. B. Phillips, G. Robuchon, R. A. Beyer, F. Nimmo, and J. Roberts. Crater relaxation and stereo imaging of rhea. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2633. Lunar and Planetary Institute, Houston, 2011

5. John Huffman, Andrew Forsberg, Andrew Loomis, James Head, James Dickson, and Caleb Fassett. Integrating advanced visualization technology into the planetary geoscience workflow. *Planetary and Space Science*, 59(11–12):1273 – 1279, 2011. ISSN 0032-0633. URL http://www.sciencedirect.com/science/article/pii/S0032063310002175

6. R. A. Beyer, B. Archinal, Y. Cheng, K. Edmundson, E. Howington-Kraus, R. L. Kirk, R. Li, A. S. McEwen, S. Mattson, X. Meng, Z. Moratto, J. Oberst, M. Rosiek, F. Scholten, T. Tran, O. Thomas, W. Wang, and the LROC Team. LROC DTM comparison effort. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2715. Lunar and Planetary Institute, Houston, 2011

7. Z. Moratto, A. Nefian, T. Kim, M. Broxton, R. A. Beyer, and T. Fong. Stereo reconstruction from apollo 15 and 16 metric camera. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2267. Lunar and Planetary Institute, Houston, 2011

8. A. Lefort, D. M. Burr, R. A. Beyer, and A. D. Howard. Topographic post-formation modifications of inverted fluvial features in the western Medusa Fossae formation, Mars. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2418. Lunar and Planetary Institute, Houston, 2011

9. C. B. Phillips, N. P. Hammond, F. Nimmo, G. robuchon, R. A. Beyer, and J. H. Roberts. Crater Relaxation and Stereo Imaging of Icy Satellites. *AGU Fall Meeting Abstracts*, (#P41F-07), December 2011

10. Vytas SunSpiral, D.W. Wheeler, Daniel Chavez-Clemente, and David Mittman. Development and field testing of the footfall planning system for the athlete robots. *Journal of Field Robotics*, 29(3): 483–505, 2012. ISSN 1556-4967. URL http://dx.doi.org/10.1002/rob.20410

11. Cristina Re, Gabriele Cremonese, Elisa Dall'Asta, Gianfranco Forlani, Giampiero Naletto, and Riccardo Roncella. Performance evaluation of dtm area-based matching reconstruction of moon and mars. *Proc. SPIE 8537, Image and Signal Processing for Remote Sensing XVIII*, pages 85370V–85370V–12, 2012. URL http://dx.doi.org/10.1117/12.974524

12. Teemu Öhman and David A. Kring. Photogeologic analysis of impact melt-rich lithologies in kepler crater that could be sampled by future missions. *Journal of Geophysical Research: Planets*, 117(E12): n/a–n/a, 2012. ISSN 2156-2202. URL http://dx.doi.org/10.1029/2011JE003918

13. M. Golombek, J. Grant, D. Kipp, A. Vasavada, R. Kirk, R. Fergason, P. Bellutta, F. Calef, K. Larsen, Y. Katayama, A. Huertas, R. Beyer, A. Chen, T. Parker, B. Pollard, S. Lee, Y. Sun, R. Hoover, H. Sladek, J. Grotzinger, R. Welch, E. Noe Dobrea, J. Michalski, and M. Watkins. Selection of the mars science laboratory landing site. *Space Science Reviews*, 170(1-4):641–737, 2012. ISSN 0038-6308. URL http://dx.doi.org/10.1007/s11214-012-9916-y

14. A. Lefort, D. M. Burr, R. A. Beyer, and A. D. Howard. Inverted fluvial features in the Aeolis-Zephyria Plana, western Medusae Fossae Formation, Mars: Evidence for post-formation modification. *Journal of Geophysical Research (Planets)*, 117:E03007, March 2012

15. A. Lefort, D. M. Burr, R. A. Beyer, and A. D. Howard. Sinuous Ridges as Tools to Investigate Post-Flow Modification in the Aeolis-Zephyria Plana, Western Medusae Fossae Formation, Mars. In *Lunar and Planetary Science Conference 43*, number #1953, March 2012

16. J. R. Laura, D. Miller, and M. V. Paul. AMES Stereo Pipeline Derived DEM Accuracy Experiment Using LROC-NAC Stereopairs and Weighted Spatial Dependence Simulation for Lunar Site Selection. In *Lunar and Planetary Science Conference*, page 2371, March 2012

17. C. B. Phillips, N. P. Hammond, G. Robuchon, F. Nimmo, R. A. Beyer, and J. Roberts. Stereo Imaging, Crater Relaxation, and Thermal Histories of Rhea and Dione. In *Lunar and Planetary Science Conference 43*, number #2571, March 2012

18. A. M. Morgan, R. A. Beyer, A. D. Howard, and J. M. Moore. The Alluvial Fans of Saheki Crater. In *Lunar and Planetary Science Conference 43*, number #2815, March 2012

19. P. Allemand, A. Deschamps, M. Lesaout, C. Delacourt, C. Quantin, and H. Clenet. Magma rheology from 3D geometry of martian lava flows. In *EGU General Assembly Conference Abstracts*, volume 14, page 8723, April 2012

20. N.P. Hammond, C.B. Phillips, F. Nimmo, and S.A. Kattenhorn. Flexure on dione: Investigating subsurface structure and thermal history. *Icarus*, 223(1):418 – 422, 2013. ISSN 0019-1035. URL http://www.sciencedirect.com/science/article/pii/S0019103513000043

21. Samantha E. Peel and Caleb I. Fassett. Valleys in pit craters on mars: Characteristics, distribution, and formation mechanisms. *Icarus*, (0):–, 2013. ISSN 0019-1035. URL http://www.sciencedirect.com/science/article/pii/S0019103513001474

22. C. B. Phillips, N. P. Hammond, J. H. Roberts, F. Nimmo, R. A. Beyer, and S. Kattenhorn. Stereo Topography and Subsurface Thermal Profiles on Icy Satellites of Saturn. In *Lunar and Planetary Science Conference 44*, number #2766, March 2013

23. T. Öhman and P. J. McGovern. Strain Calculations for Circumferential Graben on Alba Mons, Mars. *LPI Contributions*, 1719:2966, March 2013

24. W. A. Watters, L. Geiger, and M. Fendrock. Shape Distribution of Fresh Martian Impact Craters from High-Resolution DEMs. In *Lunar and Planetary Science Conference 44*, volume 44 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2013

25. C. B. Phillips, E. El Henson, and F. Nimmo. Stereo Topography of Surface Features on Europa and Comparisons with Formation Models. In *AGU Fall Meeting Abstracts*, volume 2013, pages P53A–1846, December 2013

26. W. A. Watters and A. C. Radford. 3-D Morphometry of Martian Secondary Impact Craters from Zunil and Gratteri. In *Lunar and Planetary Science 45*, volume 45 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2014

27. A. Lucchetti, R. Thomas, G. Cremonese, M. Massironi, D. A. Rothery, S. J. Conway, and M. Anand. Analysis and Numerical Modeling of a Pit Crater on Mercury. In *Lunar and Planetary Science Conference 45*, volume 45 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2014

28. A. C. G. Hughes, E. Hauber, and A. P. Rossi. Geomorphology of Glacial and Periglacial Landforms Within a Small Crater in Terra Cimmeria, Mars: Stratigraphy and Inferred Chronology of Processes. In *Lunar and Planetary Science Conference 45*, volume 45 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2014

29. Z. M. Moratto, S. T. McMichael, R. A. Beyer, O. Alexandrov, and T. Fong. Automated and Accurate: Making DTMs from LRO-NAC Using the Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 2892, March 2014

30. R. A. Beyer, O. Alexandrov, and Z. M. Moratto. Aligning Terrain Model and Laser Altimeter Point Clouds with the Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 2902, March 2014

31. A. Lucas, A. Mangeney, and J. P. Ampuero. Frictional velocity-weakening in landslides on Earth and on other planetary bodies. *Nature Communications*, 5:3417, March 2014

32. P. J. Mouginis-Mark, J. M. Boyce, and H. Garbeil. Digital Elevation Models Aid the Analysis of Double Layered Ejecta (DLE) Impact Craters on Mars. In *AGU Fall Meeting Abstracts*, volume 2014, pages P34C–05, December 2014

33. Vladimir Yershov, Anton Ivanov, Jan-Peter Muller, Yu Tao, Mr, William Pool, Jung-Rack Kim, and Panagiotis Sidiropoulos. Assessment of Digital Terrain Model algorithms for the development of a massive processing system for all high-resolution stereo images of Mars from CTX and HiRISE. In *40th COSPAR Scientific Assembly*, volume 40, pages B0.8–11–14, January 2014

34. Laura A Stevens, Mark D Behn, Jeffrey J McGuire, Sarah B Das, Ian Joughin, Thomas Herring, David E Shean, and Matt A King. Greenland supraglacial lake drainages triggered by hydrologically induced basal slip. *Nature*, 522(7554):73–76, 2015

35. Michael J Willis, Andrew K Melkonian, and Matthew E Pritchard. Outlet glacier response to the 2012 collapse of the matusevich ice shelf, severnaya zemlya, russian arctic. *Journal of Geophysical Research: Earth Surface*, 120(10):2040–2055, 2015

36. S. McMichael, Z. M. Moratto, and R. A. Beyer. LRO-NAC Mass DTM Pipeline. In *Lunar and Planetary Science Conference*, page 2491, March 2015

37. W. A. Watters, L. Geiger, M. Fendrock, R. Gibson, and A. Radford. Statistical Morphometry of Small Martian Craters: New Methods and Results. In *Issues in Crater Studies and the Dating of Planetary Surfaces*, volume 1841, page 9032, May 2015

38. V. Yershov. A system for generating multi-resolution Digital Terrain Models of Mars based on the ESA Mars Express and NASA Mars Reconnaissance Orbiter data. In *European Planetary Science Congress*, pages EPSC2015–343, October 2015

39. R. S. Bauer, M. K. Barker, E. Mazarico, and G. A. Neumann. Calibration of Mercury Laser Altimeter Data Using Digital Elevation Models Derived from Stereo Image Pairs. In *AGU Fall Meeting Abstracts*, volume 2015, pages P41C–2079, December 2015

40. David E Shean, Oleg Alexandrov, Zachary M Moratto, Benjamin E Smith, Ian R Joughin, Claire Porter, and Paul Morin. An automated, open-source pipeline for mass production of digital elevation models (dems) from very-high-resolution commercial stereo satellite imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 116:101–117, 2016

41. Pascal Lacroix. Landslides triggered by the gorkha earthquake in the langtang valley, volumes and initiation processes. *Earth, Planets and Space*, 68(1):1–10, 2016

42. Allen Pope, TA Scambos, M Moussavi, M Tedesco, M Willis, D Shean, and S Grigsby. Estimating supraglacial lake depth in west greenland using landsat 8 and comparison with other multispectral methods. *The Cryosphere*, 10:15, 2016

43. Andrew K Melkonian, Michael J Willis, Matthew E Pritchard, and Adam J Stewart. Recent changes in glacier velocities and thinning at novaya zemlya. *Remote Sensing of Environment*, 174:244–257, 2016

44. D. P. Mayer and E. S. Kite. An Integrated Workflow for Producing Digital Terrain Models of Mars from CTX and HiRISE Stereo Data Using the NASA Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 1241, March 2016

45. Anton Ivanov, Jan-Peter Muller, Yu Tao, Jung-Rack Kim, Klaus Gwinner, Stephan Van Gasselt, Jeremy Morley, Robert Houghton, Steven Bamford, Panagiotis Sidiropoulos, Lida Fanara, Marita Waenlish, Sebastian Walter, Ralf Steinkert, Bjorn Schreiner, Federico Cantini, Jessica Wardlaw, James Sprinks, Michele Giordano, and Stuart Marsh. EU-FP7-iMARS: analysis of Mars multi-resolution images using auto- coregistration, data mining and crowd source techniques. In *41st COSPAR Scientific Assembly*, volume 41, pages B0.2–22–16, July 2016

46. David Nebouy, Claire Capanna, Laurent Jorda, Robert W. Gaskell, Stubbe Faurschou Hviid, Frank Scholten, Frank Preusker, and OSIRIS Team. Co-registration and comparison of high-resolution shape models of comet 67P/C-G. In *AAS/Division for Planetary Sciences Meeting Abstracts #48*, AAS/Division for Planetary Sciences Meeting Abstracts, page 116.08, October 2016

47. Lida Fanara, Klaus Gwinner, Ernst Hauber, and Juergen Oberst. Frequency of block displacements at the north pole of Mars based on HiRISE images. In *AAS/Division for Planetary Sciences Meeting Abstracts #48*, AAS/Division for Planetary Sciences Meeting Abstracts, page 513.11, October 2016

48. P. J. Mouginis-Mark and V. L. Sharpton. Topographic Analysis of the Asymmetric Ejecta of Zunil Crater, Mars. In *AGU Fall Meeting Abstracts*, pages P11E–01, December 2016

49. Caleb I. Fassett. Ames stereo pipeline-derived digital terrain models of Mercury from MESSENGER stereo imaging. *Planetary and Space Science*, 134:19–28, December 2016

50. Paul M. Montesano, Christopher Neigh, Guoqing Sun, Laura Duncanson, Jamon Van Den Hoek, and K. Jon Ranson. The use of sun elevation angle for stereogrammetric boreal forest height in open canopies. *Remote Sensing of Environment*, 196:76 – 88, 2017. ISSN 0034-4257. URL http://www.sciencedirect.com/science/article/pii/S0034425717301827

51. Cheng Jiang, Sylvain Douté, Bin Luob, and Liangpei Zhang. Fusion of photogrammetric and photoclinometric information for high-resolution dems from mars in-orbit imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 130:Pages 418–430, 2017. URL http://www.sciencedirect.com/science/article/pii/S0924271616306554

52. J. M. C. Belart, E. Berthier, E. Magnússon, L. S. Anderson, F. Pálsson, T. Thorsteinsson, I. M. Howat, G. Aðalgeirsdóttir, T. Jóhannesson, and A. H. Jarosch. Winter mass balance of drangajökull ice cap (nw iceland) derived from satellite sub-meter stereo images. *The Cryosphere*, 11(3):1501–1517, 2017. URL https://www.the-cryosphere.net/11/1501/2017/

53. Jan-Peter Muller, Panagiotis Sidiropoulos, Yu Tao, Kiky Putri, Jacqueline Campbell, Si-Ting Xiong, Klaus Gwinner, Konrad Willner, Lida Fanara, Marita Waehlisch, Sebastian Walter, Bjoern Schreiner, Ralf Steikert, Anton Ivanov, Federico Cantini, Jessica Wardlaw, James Sprinks, Robert Houghton, and Jung-Rack Kim. EU-FP7-iMARS: analysis of Mars multi-resolution images using auto- coregistration, data mining and crowd source techniques: A Final Report on the very variable surface of Mars. In *EGU General Assembly Conference Abstracts*, volume 19, page 18917, April 2017

54. S. McMichael, O. Alexandrov, and R. Beyer. Enhanced 3D Surface Generation in the Ames Stereo Pipeline. In *Third Planetary Data Workshop and The Planetary Geologic Mappers Annual Meeting*, volume 1986, page 7090, June 2017

55. H. M. Brown, A. A. Awumah, M. R. Henriksen, M. R. Manheim, E. Cisneros, R. V. Wagner, and M. S. Robinson. Ames Stereo Pipeline and LROC ASU Digital Terrain Model (DTM) Comparison. In *Third Planetary Data Workshop and The Planetary Geologic Mappers Annual Meeting*, volume 1986, page 7096, June 2017

56. R. L. Fergason, R. L. Kirk, G. Cushing, D. M. Galuszka, M. P. Golombek, T. M. Hare, E. Howington-Kraus, D. M. Kipp, and B. L. Redding. Analysis of Local Slopes at the InSight Landing Site on Mars. *Space Science Reviews*, 211:109–133, October 2017

57. A. M. Annex, K. W. Lewis, and C. S. Edwards. Stratigraphic Mapping of Intra-Crater Layered Deposits in Arabia Terra from High-Resolution Imaging and Stereo Topography. In *AGU Fall Meeting Abstracts*, volume 2017, pages P24C–04, December 2017

58. C. Rezza, C. B. Phillips, and M. L. Cable. 'Dem DEMs: Comparing Methods of Digital Elevation Model Creation. In *AGU Fall Meeting Abstracts*, volume 2017, pages P43D–2914, December 2017

59. S. H. Moon and H. L. Choi. Alignment and Ortho-Rectification of Lunar Surface Image Using the NASA Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 1384, March 2018

60. D. P. Mayer. An Improved Workflow for Producing Digital Terrain Models of Mars from CTX Stereo Data Using the NASA Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 1604, March 2018

61. Alfiah Rizky Diana Putri, Panagiotis Sidiropoulos, Yu Tao, and Jan-Peter Muller. Automatic Multiple-Expert Quality Assessment for Batch Processed Martian DTMs. In *EGU General Assembly Conference Abstracts*, volume 20, page 1120, April 2018

62. Jan-Peter Muller, Yu Tao, Panagiotis Sidiropoulos, Alfiah Putri, Jacqueline Campbell, and Sebastian Walter. Assessment of ≈5,000 Mars-wide CTX DTMs created using the EU-FP7 iMars CASP-GO system. In *EGU General Assembly Conference Abstracts*, volume 20, page 15971, April 2018

63. Y. Tao, J. P. Muller, P. Sidiropoulos, Si-Ting Xiong, A. R. D. Putri, S. H. G. Walter, J. Veitch-Michaelis, and V. Yershov. Massive stereo-based DTM production for Mars on cloud computers. *Planetary and Space Science*, 154:30–58, May 2018

64. Valentin Tertius Bickel, CI Honniball, SN Martinez, A Rogaski, HM Sargeant, SK Bell, EC Czaplinski, BE Farrant, EM Harrington, GD Tolometti, et al. Analysis of lunar boulder tracks: Implications for trafficability of pyroclastic deposits. *Journal of Geophysical Research: Planets*, 2019

65. S. Shahrzad, K. M. Kinch, T. A. Goudge, C. I. Fassett, D. H. Needham, C. Quantin-Nataf, and C. P. Knudsen. Crater Statistics on the Dark-Toned, Mafic Floor Unit in Jezero Crater, Mars. *Geophysical Research Letters*, 46:2408–2416, March 2019

66. P. J. Mouginis-Mark and H. Garbeil. CTX Digital Elevation Models Facilitate Geomorphic Analysis of Mars. In *Lunar and Planetary Science Conference*, page 1069, Mar 2019

67. D. P. Mayer. Filling the Gap: Building a CTX-Based Digital Terrain Model Mosaic of the South Pole of Mars. In *Lunar and Planetary Science Conference*, page 1128, Mar 2019

68. T. A. Goudge, C. I. Fassett, and G. R. Osinski. How Do Crater Lakes on Mars Develop Inlet Valleys? In *Lunar and Planetary Science Conference*, page 1223, Mar 2019

69. R. Hemmi and H. Miyamoto. HiRISE Digital Elevation Model of Phobos: Implications for Morphological Analysis of Grooves. In *Lunar and Planetary Science Conference*, page 1759, Mar 2019

70. A. M. Annex, A. H. D. Koeppel, C. Pan, C. E. Edwards, and K. W. Lewis. Scarp Associated with Martian Layered Deposits in Arabia Terra. In *Lunar and Planetary Science Conference*, page 1973, Mar 2019

71. R. Hemmi and H. Miyamoto. High-resolution Topographic Analysis of Pitted Mounds in Southern Acidalia Planitia, Mars: Updates on Morphometric Parameters of Candidate Mud Volcanoes. In *Lunar and Planetary Science Conference*, page 2479, Mar 2019

Bibliography

[1] Michael Abrams, Simon Hook, and Bhaskar Ramachandran. Aster user handbook, version 2. *Jet Propulsion Laboratory*, 4800:135, 2002.

[2] C. H. Acton. Ancillary data services of NASA's Navigation and Ancillary Information Facility. *Planetary and Space Science*, 44:65–70, January 1996.

[3] C. H. Acton. SPICE Products Available to the Planetary Science Community. In *Lunar and Planetary Science XXX*, number #1233. Lunar and Planetary Institute, Houston (CD-ROM), March 1999.

[4] C. H. Acton, N. J. Bachman, J. A. Bytof, B. V. Semenov, W. Taber, F. S. Turner, and E. D. Wright. Examining Mars with SPICE. In *Fifth International Conference on Mars*, number #6042. Lunar and Planetary Institute, Houston (CD-ROM), July 1999.

[5] P. Allemand, A. Deschamps, M. Lesaout, C. Delacourt, C. Quantin, and H. Clenet. Magma rheology from 3D geometry of martian lava flows. In *EGU General Assembly Conference Abstracts*, volume 14, page 8723, April 2012.

[6] J. A. Anderson, S. C. Sides, D. L. Soltesz, T. L. Sucharski, and K. J. Becker. Modernization of the Integrated Software for Imagers and Spectrometers. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science XXXV*, number #2039. Lunar and Planetary Institute, Houston (CD-ROM), March 2004.

[7] J.A. Anderson. ISIS Camera Model Design. In *Proc of the Lunar and Planetary Science Conference (LPSC) XXXIX*, page 2159, March 2008.

[8] A. M. Annex, K. W. Lewis, and C. S. Edwards. Stratigraphic Mapping of Intra-Crater Layered Deposits in Arabia Terra from High-Resolution Imaging and Stereo Topography. In *AGU Fall Meeting Abstracts*, volume 2017, pages P24C–04, December 2017.

[9] A. M. Annex, A. H. D. Koeppel, C. Pan, C. E. Edwards, and K. W. Lewis. Scarp Associated with Martian Layered Deposits in Arabia Terra. In *Lunar and Planetary Science Conference*, page 1973, Mar 2019.

[10] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, Feb 2004. ISSN 1573-1405. URL https://doi.org/10.1023/B:VISI.0000011205.11775.fd.

[11] R. S. Bauer, M. K. Barker, E. Mazarico, and G. A. Neumann. Calibration of Mercury Laser Altimeter Data Using Digital Elevation Models Derived from Stereo Image Pairs. In *AGU Fall Meeting Abstracts*, volume 2015, pages P41C–2079, December 2015.

[12] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *Computer Vision and Image Understanding (CVIU)*, volume 110, pages 346–359, 2008. URL http://www.vision.ee.ethz.ch/~surf/.

[13] J. M. C. Belart, E. Berthier, E. Magnússon, L. S. Anderson, F. Pálsson, T. Thorsteinsson, I. M. Howat, G. Aðalgeirsdóttir, T. Jóhannesson, and A. H. Jarosch. Winter mass balance of drangajökull ice cap (nw iceland) derived from satellite sub-meter stereo images. *The Cryosphere*, 11(3):1501–1517, 2017. URL https://www.the-cryosphere.net/11/1501/2017/.

[14] R. A. Beyer, B. Archinal, Y. Chen, K. Edmundson, D. Harbour, E. Howington-Kraus, R. Li, A. McEwen, S. Mattson, Z. Moratto, J. Oberst, M. Rosiek, F. Scholten, T. Tran, M. Robinson, and LROC Team. LROC Stereo Data–Results of Initial Analysis. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 41*, number #2678. Lunar and Planetary Institute, Houston, March 2010.

[15] R. A. Beyer, B. Archinal, Y. Cheng, K. Edmundson, E. Howington-Kraus, R. L. Kirk, R. Li, A. S. McEwen, S. Mattson, X. Meng, Z. Moratto, J. Oberst, M. Rosiek, F. Scholten, T. Tran, O. Thomas, W. Wang, and the LROC Team. LROC DTM comparison effort. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2715. Lunar and Planetary Institute, Houston, 2011.

[16] R. A. Beyer, O. Alexandrov, and Z. M. Moratto. Aligning Terrain Model and Laser Altimeter Point Clouds with the Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 2902, March 2014.

[17] Ross A. Beyer, Oleg Alexandrov, and Scott McMichael. The Ames Stereo Pipeline: NASA's open source software for deriving and processing terrain data. *Earth and Space Science*.

[18] Valentin Tertius Bickel, CI Honniball, SN Martinez, A Rogaski, HM Sargeant, SK Bell, EC Czaplinski, BE Farrant, EM Harrington, GD Tolometti, et al. Analysis of lunar boulder tracks: Implications for trafficability of pyroclastic deposits. *Journal of Geophysical Research: Planets*, 2019.

[19] Tye Brady. ALHAT Requirements. presentation at the Lunar Coordinate Systems Review Data Product Recommendation Meeting, NASA Ames Research Center, October 11.

[20] H. M. Brown, A. A. Awumah, M. R. Henriksen, M. R. Manheim, E. Cisneros, R. V. Wagner, and M. S. Robinson. Ames Stereo Pipeline and LROC ASU Digital Terrain Model (DTM) Comparison. In *Third Planetary Data Workshop and The Planetary Geologic Mappers Annual Meeting*, volume 1986, page 7096, June 2017.

[21] Michael Broxton, Ara V. Nefian, Zachary Moratto, Taemin Kim, Michael Lundy, and Aleksandr V. Segal. 3D Lunar Terrain Reconstruction from Apollo Images . In *to appear in the Proceedings of the 5th International Symposium on Visual Computing*, 2009.

[22] USGS Astrogeology Science Center. USGS ISIS Documentation. Isis 3 Application Documentation *http://isis.astrogeology.usgs.gov/Application/index.html*. URL http://isis.astrogeology.usgs.gov/Application/index.html.

[23] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, October 2008. ISSN 0098-3500. URL http://doi.acm.org/10.1145/1391989.1391995.

[24] Li Cheng and T. Caelli. Bayesian stereo matching. *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*, pages 192–192, June 2004.

[25] G. Chin, A. Bartels, S. Brylow, M. Foote, J. Garvin, J. Kaspar, J. Keller, I. Mitrofanov, K. Raney, M. Robinson, D. Smith, H. Spence, P. Spudis, S. A. Stern, and M. Zuber. Lunar Reconnaissance Orbiter Overview: The Instrument Suite and Mission. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science XXXVII*, page #1949, March 2006.

[26] G. Chin, S. Brylow, M. Foote, J. Garvin, J. Kasper, J. Keller, M. Litvak, I. Mitrofanov, D. Paige, K. Raney, M. Robinson, A. Sanin, D. Smith, H. Spence, P. Spudis, S. A. Stern, and M. Zuber. Lunar Reconnaissance Orbiter Overview: The Instrument Suite and Mission. *Space Science Reviews*, 129: 391–419, April 2007.

[27] The Open Scene Graph Community. The open scene graph website. 2009. URL http://www.openscenegraph.org/projects/osg.

[28] S. Debei, A. Aboudan, G. Colombatti, and M. Pertile. Lutetia surface reconstruction and uncertainty analysis. *Planetary and Space Science*, 71:64–72, October 2012.

[29] Stefano Debei, Alessio Aboudan, Giacomo Colombatti, and Marco Pertile. Lutetia surface reconstruction and uncertainty analysis. *Planetary and Space Science*, 71(1):64 – 72, 2012. ISSN 0032-0633. URL http://www.sciencedirect.com/science/article/pii/S0032063312002073.

[30] NASA ARC Intelligent Systems Division. NASA Vision Workbench. NASA Ames Research Center, Moffett Field, CA. *http://ti.arc.nasa.gov/visionworkbench/*. URL http://ti.arc.nasa.gov/visionworkbench/.

[31] L. Edwards and M. Broxton. Automated 3D Surface Reconstruction from Orbital Imagery. In *Proceedings of AIAA Space 2006*, September 2006.

[32] L. Edwards, J. Bowman, C. Kunz, D. Lees, and M. Sims. Photo-realistic Terrain Modeling and Visualization for Mars Exploration Rover Science Operations. In *Proceedings of IEEE SMC 2005*, October 2005.

[33] Gabriele Facciolo, Carlo De Franchis, and Enric Meinhardt. Mgm: A significantly more global matching for stereovision. In *Proceedings of the British Machine Vision Conference (BMVC), BMVA Press*, pages 90–1, 2015.

[34] Lida Fanara, Klaus Gwinner, Ernst Hauber, and Juergen Oberst. Frequency of block displacements at the north pole of Mars based on HiRISE images. In *AAS/Division for Planetary Sciences Meeting Abstracts #48*, AAS/Division for Planetary Sciences Meeting Abstracts, page 513.11, October 2016.

[35] Caleb I. Fassett. Ames stereo pipeline-derived digital terrain models of Mercury from MESSENGER stereo imaging. *Planetary and Space Science*, 134:19–28, December 2016.

[36] R. L. Fergason, R. L. Kirk, G. Cushing, D. M. Galuszka, M. P. Golombek, T. M. Hare, E. Howington-Kraus, D. M. Kipp, and B. L. Redding. Analysis of Local Slopes at the InSight Landing Site on Mars. *Space Science Reviews*, 211:109–133, October 2017.

[37] J Fernando, F Schmidt, X Ceamanos, P Pinet, S Douté, and Y Daydou. Surface reflectance of mars observed by crism/mro: 2. estimation of surface photometric properties in gusev crater and meridiani planum. *Journal of Geophysical Research: Planets*, 118(3):534–559, 2013.

[38] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Graphics and Image Processing*, 24(6), June 1981.

[39] The CGIAR Consortium for Spatial Information. CGIAR-CSI SRTM 90m DEM Digital Elevation Database. URL http://srtm.csi.cgiar.org.

[40] L. Gaddis, J. Anderson, K. Becker, T. Becker, D. Cook, K. Edwards, E. Eliason, T. Hare, H. Kieffer, E. M. Lee, J. Mathews, L. Soderblom, T. Sucharski, J. Torson, A. McEwen, and M. Robinson. An Overview of the Integrated Software for Imaging Spectrometers (ISIS). In *Lunar and Planetary Science Conference*, volume 28, page 387, March 1997.

[41] GeoEye. Sample Imagery Request Form. GeoEye sample imagery request form *http://geoeye.com/CorpSite/solutions/learn-more/sample-imagery.aspx*. URL http://geoeye.com/CorpSite/solutions/learn-more/sample-imagery.aspx.

[42] L Girod, C Nuth, and A Kääb. Improvement of dem generation from aster images using satellite jitter estimation and open source implementation. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(1):249, 2015.

[43] Digital Globe. Radiometric Use of WorldView 2 Imagery. Description of the WV02 camera, . URL http://www.digitalglobe.com/sites/default/files/Radiometric_Use_of_WorldView-2_Imagery%20%281%29.pdf.

[44] Digital Globe. Satellite Imagery and Geospatial Information Products. Digital Globe sample imagery *https://www.digitalglobe.com/samples*, . URL https://www.digitalglobe.com/samples.

[45] M. Golombek, J. Grant, D. Kipp, A. Vasavada, R. Kirk, R. Fergason, P. Bellutta, F. Calef, K. Larsen, Y. Katayama, A. Huertas, R. Beyer, A. Chen, T. Parker, B. Pollard, S. Lee, Y. Sun, R. Hoover, H. Sladek, J. Grotzinger, R. Welch, E. Noe Dobrea, J. Michalski, and M. Watkins. Selection of the mars science laboratory landing site. *Space Science Reviews*, 170(1-4):641–737, 2012. ISSN 0038-6308. URL http://dx.doi.org/10.1007/s11214-012-9916-y.

[46] T. A. Goudge, C. I. Fassett, and G. R. Osinski. How Do Crater Lakes on Mars Develop Inlet Valleys? In *Lunar and Planetary Science Conference*, page 1223, Mar 2019.

[47] Manuel Guizar-Sicairos, Samuel T Thurman, and James R Fienup. Efficient subpixel image registration algorithms. *Optics letters*, 33(2):156–158, 2008.

[48] Rajiv Gupta and Richard I. Hartley. Linear Pushbroom Cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9), September 1997.

[49] Ebner H., Spiegel M., Albert B., Bernd G., and Neukum G. et. al. Improving The Exterior Orientation of Mars Express Hrsc Imagery. In *XXth ISPRS Congress, Commission IV*, 2004.

[50] N. P. Hammond, C. B. Phillips, G. Robuchon, R. A. Beyer, F. Nimmo, and J. Roberts. Crater relaxation and stereo imaging of rhea. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2633. Lunar and Planetary Institute, Houston, 2011.

[51] N.P. Hammond, C.B. Phillips, F. Nimmo, and S.A. Kattenhorn. Flexure on dione: Investigating subsurface structure and thermal history. *Icarus*, 223(1):418 – 422, 2013. ISSN 0019-1035. URL http://www.sciencedirect.com/science/article/pii/S0019103513000043.

[52] Bruce Hapke. Bidirectional reflectance spectroscopy: 6. effects of porosity. *Icarus*, 195(2):918–926, 2008.

[53] Bruce W Hapke, Robert M Nelson, and William D Smythe. The opposition effect of the moon: The contribution of coherent backscatter. *Science*, 260(5107):509–511, 1993.

[54] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[55] C. Heipke and J. Oberst et. al. The HRSC DTM Test. In *Symposium of ISPRS Commission IV - Geo Spatial Databases for Sustainable Development*, 2006.

[56] R. Hemmi and H. Miyamoto. HiRISE Digital Elevation Model of Phobos: Implications for Morphological Analysis of Grooves. In *Lunar and Planetary Science Conference*, page 1759, Mar 2019.

[57] R. Hemmi and H. Miyamoto. High-resolution Topographic Analysis of Pitted Mounds in Southern Acidalia Planitia, Mars: Updates on Morphometric Parameters of Candidate Mud Volcanoes. In *Lunar and Planetary Science Conference*, page 2479, Mar 2019.

[58] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:328–341, 2008.

[59] Heiko Hirschmüller, Helmut Mayer, G Neukum, et al. Stereo processing of hrsc mars express images by semi-global matching. *Int. Arch. Photogramm. Remote Sensing Spatial Inf. Sci*, 36:305–310, 2006.

[60] Han Hua, Chongtai Chenb, Bo Wua, Xiaoxia Yangc, Qing Zhub, and Yulin Dingb. Texture-aware dense image matching using ternary census transform. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 59–66, 2016.

[61] John Huffman, Andrew Forsberg, Andrew Loomis, James Head, James Dickson, and Caleb Fassett. Integrating advanced visualization technology into the planetary geoscience workflow. *Planetary and Space Science*, 59(11–12):1273 – 1279, 2011. ISSN 0032-0633. URL http://www.sciencedirect.com/science/article/pii/S0032063310002175.

[62] A. C. G. Hughes, E. Hauber, and A. P. Rossi. Geomorphology of Glacial and Periglacial Landforms Within a Small Crater in Terra Cimmeria, Mars: Stratigraphy and Inferred Chronology of Processes. In *Lunar and Planetary Science Conference 45*, volume 45 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2014.

[63] A. B. Ivanov and J. J. Lorre. Analysis of Mars Orbiter Camera Stereo Pairs. In *Lunar and Planetary Institute Conference Abstracts*, pages 1845–+, March 2002.

[64] Anton Ivanov, Jan-Peter Muller, Yu Tao, Jung-Rack Kim, Klaus Gwinner, Stephan Van Gasselt, Jeremy Morley, Robert Houghton, Steven Bamford, Panagiotis Sidiropoulos, Lida Fanara, Marita Waenlish, Sebastian Walter, Ralf Steinkert, Bjorn Schreiner, Federico Cantini, Jessica Wardlaw, James Sprinks, Michele Giordano, and Stuart Marsh. EU-FP7-iMARS: analysis of Mars multi-resolution images using auto- coregistration, data mining and crowd source techniques. In *41st COSPAR Scientific Assembly*, volume 41, pages B0.2–22–16, July 2016.

[65] Cheng Jiang, Sylvain Douté, Bin Luob, and Liangpei Zhang. Fusion of photogrammetric and photoclinometric information for high-resolution dems from mars in-orbit imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 130:Pages 418–430, 2017. URL http://www.sciencedirect.com/science/article/pii/S0924271616306554.

[66] Jeffrey R Johnson, William M Grundy, Mark T Lemmon, James F Bell, Miles J Johnson, Robert G Deen, Raymond E Arvidson, William H Farrand, Edward A Guinness, Alexander G Hayes, et al. Spectrophotometric properties of materials observed by pancam on the mars exploration rovers: 1. spirit. *Journal of Geophysical Research: Planets*, 111(E2), 2006.

[67] M. D. Johnston, J. E. Graf, R. W. Zurek, H. J. Eisen, and B. Jai. The Mars Reconnaissance Orbiter Mission. In *2003 IEEE Aerospace Conference*, pages 447–464, 2003.

[68] J.R. Kim and J.-P. Muller. Multi-resolution topographic data extraction from martian stereo imagery. *Planetary and Space Science*, 57(14–15):2095 – 2112, 2009. ISSN 0032-0633. URL http://www.sciencedirect.com/science/article/pii/S0032063309002888.

[69] R. L. Kirk, E. Howington-Kraus, D. Galuszka, B. Redding, T. M. Hare, C. Heipke, J. Oberst, G. Neukum, and HRSC Co-Investigator Team. Mapping Mars with HRSC, ISIS, and SOCET SET. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science XXXVII*, page #2050, March 2006.

[70] R.L. Kirk, Laurence A. Soderblom, Elipitha Howington-Kraus, and Brent Archinal. USGS High-Resolution Topomapping of Mars with Mars Orbiter Camera Narrow-Angle Images. *IAPRS: GeoSpatial Theory, Processing and Applications*, 34, 2002.

[71] Kurt Konolige. Sparse sparse bundle adjustment. In *British Machine Vision Conference*, Aberystwyth, Wales, 08/2010 2010.

[72] C. Kunz and H. Singh. Stereo self-calibration for seafloor mapping using auvs. In *Autonomous Underwater Vehicles (AUV), 2010 IEEE/OES*, pages 1–7, 2010.

[73] Pascal Lacroix. Landslides triggered by the gorkha earthquake in the langtang valley, volumes and initiation processes. *Earth, Planets and Space*, 68(1):1–10, 2016.

[74] J. R. Laura, D. Miller, and M. V. Paul. AMES Stereo Pipeline Derived DEM Accuracy Experiment Using LROC-NAC Stereopairs and Weighted Spatial Dependence Simulation for Lunar Site Selection. In *Lunar and Planetary Science Conference*, page 2371, March 2012.

[75] S. J. Lawrence, M. S. Robinson, M. Broxton, J. D. Stopar, W. Close, J. Grunsfeld, R. Ingram, L. Jefferson, S. Locke, R. Mitchell, T. Scarsella, M. White, M. A. Hager, T. R. Watters ad E. Bowman-Cisneros, J. Danton, and J. Garvin. The Apollo Digital Image Archive: New Research and Data Products. In *Proc of the NLSI Lunar Science Conference*, page 2066, 2008.

[76] A. Lefort, D. M. Burr, R. A. Beyer, and A. D. Howard. Topographic post-formation modifications of inverted fluvial features in the western Medusa Fossae formation, Mars. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2418. Lunar and Planetary Institute, Houston, 2011.

[77] A. Lefort, D. M. Burr, R. A. Beyer, and A. D. Howard. Inverted fluvial features in the Aeolis-Zephyria Plana, western Medusae Fossae Formation, Mars: Evidence for post-formation modification. *Journal of Geophysical Research (Planets)*, 117:E03007, March 2012.

[78] A. Lefort, D. M. Burr, R. A. Beyer, and A. D. Howard. Sinuous Ridges as Tools to Investigate Post-Flow Modification in the Aeolis-Zephyria Plana, Western Medusae Fossae Formation, Mars. In *Lunar and Planetary Science Conference 43*, number #1953, March 2012.

[79] Rongxing Li, Juwon Hwangbo, Yunhang Chen, and Kaichang Di. Rigorous photogrammetric processing of hirise stereo imagery for mars topographic mapping. *Geoscience and Remote Sensing, IEEE Transactions on*, 49(7):2558–2572, 2011. ISSN 0196-2892.

[80] Volker Lohse, Christian Heipke, and Randolph L Kirk. Derivation of planetary topography using multi-image shape-from-shading. *Planetary and space science*, 54(7):661–674, 2006.

[81] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 2004.

[82] A. Lucas, A. Mangeney, and J. P. Ampuero. Frictional velocity-weakening in landslides on Earth and on other planetary bodies. *Nature Communications*, 5:3417, March 2014.

[83] A. Lucchetti, R. Thomas, G. Cremonese, M. Massironi, D. A. Rothery, S. J. Conway, and M. Anand. Analysis and Numerical Modeling of a Pit Crater on Mercury. In *Lunar and Planetary Science Conference 45*, volume 45 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2014.

[84] Daniel Machacek. Images from the long-awaited Dawn Vesta data set. http://www.planetary.org/blogs/guest-blogs/20121129-machacek-dawn-vesta.html, 2012.

[85] M. C. Malin and K. S. Edgett. Mars Global Surveyor Mars Orbiter Camera: Interplanetary cruise through primary mission. *Journal of Geophysical Research*, 106(E10):23429–23570, October 2001.

[86] M. C. Malin, G. E. Danielson, A. P. Ingersoll, H. Masursky, J. Veverka, M. A. Ravine, and T. A. Soulanille. Mars Observer Camera. *Journal of Geophysical Research*, 97(E5):7699–7718, May 1992.

[87] D. P. Mayer. An Improved Workflow for Producing Digital Terrain Models of Mars from CTX Stereo Data Using the NASA Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 1604, March 2018.

[88] D. P. Mayer. Filling the Gap: Building a CTX-Based Digital Terrain Model Mosaic of the South Pole of Mars. In *Lunar and Planetary Science Conference*, page 1128, Mar 2019.

[89] D. P. Mayer and E. S. Kite. An Integrated Workflow for Producing Digital Terrain Models of Mars from CTX and HiRISE Stereo Data Using the NASA Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 1241, March 2016.

[90] Alfred S McEwen. Photometric functions for photoclinometry and other applications. *Icarus*, 92(2): 298–311, 1991.

[91] S. McMichael, Z. M. Moratto, and R. A. Beyer. LRO-NAC Mass DTM Pipeline. In *Lunar and Planetary Science Conference*, page 2491, March 2015.

[92] S. McMichael, O. Alexandrov, and R. Beyer. Enhanced 3D Surface Generation in the Ames Stereo Pipeline. In *Third Planetary Data Workshop and The Planetary Geologic Mappers Annual Meeting*, volume 1986, page 7090, June 2017.

[93] Andrew K Melkonian, Michael J Willis, Matthew E Pritchard, and Adam J Stewart. Recent changes in glacier velocities and thinning at novaya zemlya. *Remote Sensing of Environment*, 174:244–257, 2016.

[94] Christian Menard. *Robust Stereo and Adaptive Matching in Correlation Scale-Space*. PhD thesis, Institute of Automation, Vienna Institute of Technology (PRIP-TR-45), January 1997.

[95] Paul M. Montesano, Christopher Neigh, Guoqing Sun, Laura Duncanson, Jamon Van Den Hoek, and K. Jon Ranson. The use of sun elevation angle for stereogrammetric boreal forest height in open canopies. *Remote Sensing of Environment*, 196:76 – 88, 2017. ISSN 0034-4257. URL http://www.sciencedirect.com/science/article/pii/S0034425717301827.

[96] S. H. Moon and H. L. Choi. Alignment and Ortho-Rectification of Lunar Surface Image Using the NASA Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 1384, March 2018.

[97] Zach Moore, Dan Wright, Chris Lewis, and Dale Schinstock. Comparison of bundle adjustment formulations. In *ASPRS Annual Conf., Baltimore, Maryland*, 2009.

[98] Z. Moratto, A. Nefian, T. Kim, M. Broxton, R. A. Beyer, and T. Fong. Stereo reconstruction from apollo 15 and 16 metric camera. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 42*, number #2267. Lunar and Planetary Institute, Houston, 2011.

[99] Z. M. Moratto, S. T. McMichael, R. A. Beyer, O. Alexandrov, and T. Fong. Automated and Accurate: Making DTMs from LRO-NAC Using the Ames Stereo Pipeline. In *Lunar and Planetary Science Conference*, page 2892, March 2014.

[100] Zachary Moratto. Creating control networks and bundle adjusting with isis3. http://lunokhod.org/?p=468, 2012.

[101] Zachary Moratto. Making well registered dems with isis and ames stereo pipeline. http://lunokhod.org/?p=559, 2012.

[102] A. M. Morgan, R. A. Beyer, A. D. Howard, and J. M. Moore. The Alluvial Fans of Saheki Crater. In *Lunar and Planetary Science Conference 43*, number #2815, March 2012.

[103] P. J. Mouginis-Mark and H. Garbeil. CTX Digital Elevation Models Facilitate Geomorphic Analysis of Mars. In *Lunar and Planetary Science Conference*, page 1069, Mar 2019.

[104] P. J. Mouginis-Mark and V. L. Sharpton. Topographic Analysis of the Asymmetric Ejecta of Zunil Crater, Mars. In *AGU Fall Meeting Abstracts*, pages P11E–01, December 2016.

[105] P. J. Mouginis-Mark, J. M. Boyce, and H. Garbeil. Digital Elevation Models Aid the Analysis of Double Layered Ejecta (DLE) Impact Craters on Mars. In *AGU Fall Meeting Abstracts*, volume 2014, pages P34C–05, December 2014.

[106] Jan-Peter Muller, Panagiotis Sidiropoulos, Yu Tao, Kiky Putri, Jacqueline Campbell, Si-Ting Xiong, Klaus Gwinner, Konrad Willner, Lida Fanara, Marita Waehlisch, Sebastian Walter, Bjoern Schreiner, Ralf Steikert, Anton Ivanov, Federico Cantini, Jessica Wardlaw, James Sprinks, Robert Houghton, and Jung-Rack Kim. EU-FP7-iMARS: analysis of Mars multi-resolution images using auto- coregistration, data mining and crowd source techniques: A Final Report on the very variable surface of Mars. In *EGU General Assembly Conference Abstracts*, volume 19, page 18917, April 2017.

[107] Jan-Peter Muller, Yu Tao, Panagiotis Sidiropoulos, Alfiah Putri, Jacqueline Campbell, and Sebastian Walter. Assessment of ≈5,000 Mars-wide CTX DTMs created using the EU-FP7 iMars CASP-GO system. In *EGU General Assembly Conference Abstracts*, volume 20, page 15971, April 2018.

[108] David Nebouy, Claire Capanna, Laurent Jorda, Robert W. Gaskell, Stubbe Faurschou Hviid, Frank Scholten, Frank Preusker, and OSIRIS Team. Co-registration and comparison of high-resolution shape models of comet 67P/C-G. In *AAS/Division for Planetary Sciences Meeting Abstracts #48*, AAS/Division for Planetary Sciences Meeting Abstracts, page 116.08, October 2016.

[109] Ara V. Nefian, Kyle Husmann, Michael Broxton, Mattew D. Hancher, and Michael Lundy. A Bayesian Formulation for Subpixel Refinement in Stereo Orbital Imagery. In *to appear in the Proceedings of the 2009 IEEE International Conference on Image Processing*, 2009.

[110] Diego Nehab, Szymon Rusinkiewicz, and James Davis. Improved sub-pixel stereo correspondences through symmetric refinement. *Computer Vision, IEEE International Conference on*, 1:557–563, 2005. ISSN 1550-5499.

[111] G. Neukum and R. Jaumann. HRSC: the High Resolution Stereo Camera of Mars Express. In Andrew Wilson and Agustin Chicarro, editors, *Mars Express: the scientific payload*, number ESA SP-1240, pages 17–35. ESA Publications Division, Noordwijk, Netherlands, August 2004.

[112] G. A. Neumann, F. G. Lemoine, E. Mazarico, J. F. McGarry, D. D. Rowlands, D. E. Smith, X. Sun, M. Torrence, T. Zagwodski, R. Zellar, and M. T. Zuber. Status of Lunar Reconnaissance Orbiter Laser Ranging and Laser Altimeter Experiments. *AGU Fall Meeting Abstracts*, pages B1419+, December 2008.

[113] et al. Nguyen, L. Virtual reality interfaces for visualization and control of remote vehicles. *Autonomous Robots*, 11(1), 2001.

[114] H.K. Nishihara. PRISM: A Practical real-time imaging stereo matcher. *Optical Engineering*, 23(5): 536–545, 1984.

[115] H.K. Nishihara. Practical real-time imaging stereo matcher. *Optical Engineering*, 23(5):536–545, 1984.

[116] T. Öhman and P. J. McGovern. Strain Calculations for Circumferential Graben on Alba Mons, Mars. *LPI Contributions*, 1719:2966, March 2013.

[117] Teemu Öhman and David A. Kring. Photogeologic analysis of impact melt-rich lithologies in kepler crater that could be sampled by future missions. *Journal of Geophysical Research: Planets*, 117(E12): n/a–n/a, 2012. ISSN 2156-2202. URL http://dx.doi.org/10.1029/2011JE003918.

[118] Samantha E. Peel and Caleb I. Fassett. Valleys in pit craters on mars: Characteristics, distribution, and formation mechanisms. *Icarus*, (0):–, 2013. ISSN 0019-1035. URL http://www.sciencedirect.com/science/article/pii/S0019103513001474.

[119] C. B. Phillips, R. A. Beyer, F. Nimmo, J. H. Roberts, and G. Robuchon. Crater Relaxation and Stereo Imaging of the Icy Satellites of Jupiter and Saturn. *AGU Fall Meeting Abstracts*, (#P21B-1596), December 2010.

[120] C. B. Phillips, N. P. Hammond, F. Nimmo, G. robuchon, R. A. Beyer, and J. H. Roberts. Crater Relaxation and Stereo Imaging of Icy Satellites. *AGU Fall Meeting Abstracts*, (#P41F-07), December 2011.

[121] C. B. Phillips, N. P. Hammond, G. Robuchon, F. Nimmo, R. A. Beyer, and J. Roberts. Stereo Imaging, Crater Relaxation, and Thermal Histories of Rhea and Dione. In *Lunar and Planetary Science Conference 43*, number #2571, March 2012.

[122] C. B. Phillips, E. El Henson, and F. Nimmo. Stereo Topography of Surface Features on Europa and Comparisons with Formation Models. In *AGU Fall Meeting Abstracts*, volume 2013, pages P53A–1846, December 2013.

[123] C. B. Phillips, N. P. Hammond, J. H. Roberts, F. Nimmo, R. A. Beyer, and S. Kattenhorn. Stereo Topography and Subsurface Thermal Profiles on Icy Satellites of Saturn. In *Lunar and Planetary Science Conference 44*, number #2766, March 2013.

[124] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP Variants on Real-World Data Sets. *Autonomous Robots*, 34(3):133–148, February 2013.

[125] Allen Pope, TA Scambos, M Moussavi, M Tedesco, M Willis, D Shean, and S Grigsby. Estimating supraglacial lake depth in west greenland using landsat 8 and comparison with other multispectral methods. *The Cryosphere*, 10:15, 2016.

[126] Alfiah Rizky Diana Putri, Panagiotis Sidiropoulos, Yu Tao, and Jan-Peter Muller. Automatic Multiple-Expert Quality Assessment for Batch Processed Martian DTMs. In *EGU General Assembly Conference Abstracts*, volume 20, page 1120, April 2018.

[127] Cristina Re, Gabriele Cremonese, Elisa Dall'Asta, Gianfranco Forlani, Giampiero Naletto, and Riccardo Roncella. Performance evaluation of dtm area-based matching reconstruction of moon and mars. *Proc. SPIE 8537, Image and Signal Processing for Remote Sensing XVIII*, pages 85370V–85370V–12, 2012. URL http://dx.doi.org/10.1117/12.974524.

[128] C. Rezza, C. B. Phillips, and M. L. Cable. 'Dem DEMs: Comparing Methods of Digital Elevation Model Creation. In *AGU Fall Meeting Abstracts*, volume 2017, pages P43D–2914, December 2017.

[129] M. S. Robinson, E. M. Eliason, H. Hiesinger, B. L. Jolliff, A. S. McEwen, M. C. Malin, M. A. Ravine, D. Roberts, P. C. Thomas, and E. P. Turtle. LROC – Lunar Reconnaissance Orbiter Camera. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science XXXVI*, number #1576. Lunar and Planetary Institute, Houston (CD-ROM), March 2005.

[130] M.S. Robinson, E.M. Eliason, H. Hiesinger, B.L. Jolliff, A.S. McEwen, M.C. Malin, M.A. Ravine, D. Roberts, P.C. Thomas, and E.P. Turtle. LROC - Lunar Reconaissance Orbiter Camera. In *Proc of the Lunar and Planetary Science Conference (LPSC) XXXVI*, page 1576, March 2005.

[131] Mathias Rothermel, Konrad Wenzel, Dieter Fritsch, and Norbert Haala. Sure: Photogrammetric surface reconstruction from imagery. In *Proceedings LC3D Workshop, Berlin*, volume 8, 2012.

[132] Toni Schenk, Beata Csatho, and Sung Woong Shin. Rigorous panoramic camera model for disp imagery. In *Proceedings of the ISPRS Workshop: High Resolution Mapping from Space*, 2003.

[133] Jakob Schwendner and Javier Hidalgo. Terrain aided navgiation for plaentary exploration missions. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, September 2012.

[134] S. Shahrzad, K. M. Kinch, T. A. Goudge, C. I. Fassett, D. H. Needham, C. Quantin-Nataf, and C. P. Knudsen. Crater Statistics on the Dark-Toned, Mafic Floor Unit in Jezero Crater, Mars. *Geophysical Research Letters*, 46:2408–2416, March 2019.

[135] David E Shean, Oleg Alexandrov, Zachary M Moratto, Benjamin E Smith, Ian R Joughin, Claire Porter, and Paul Morin. An automated, open-source pipeline for mass production of digital elevation models (dems) from very-high-resolution commercial stereo satellite imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 116:101–117, 2016.

[136] Greg Slabaugh, Ron Schafer, and Mark Livingston. Optimal ray intersection for computing 3d points from n-view correspondences. http://www.soi.city.ac.uk/~sbbh653/publications/opray.pdf, 2001.

[137] D. E. Smith, M. T. Zuber, H. V. Frey, J. B. Garvin, J. W. Head, D. O. Muhleman, G. H. Pettengill, R. J. Phillips, S. C. Solomon, H. J. Zwally, W. B. Banerdt, T. C. Duxbury, M. P. Golombek, F. G. Lemoine, G. A. Neumann, and et al. Mars Orbiter Laser Altimeter: Experiment summary after the first year of global mapping of Mars. *Journal of Geophysical Research*, 106(E10):23689–23722, October 2001.

[138] DE Smith, MT Zuber, GA Neumann, E Mazarico, J Head, MH Torrence, et al. Results from the lunar orbiter laser altimeter (lola): global, high resolution topographic mapping of the moon. In *Lunar and Planetary Science Conference*, volume 42, page 2350, 2011.

[139] Hong-Gyoo Sohn, Gi-Hong Kim, and Jae-Hong Yom. Mathematical modelling of historical reconnaissance corona kh-4b imagery. *The Photogrammetric Record*, 19(105):51–66, 2004.

[140] Andrew Stein, Andres Huertas, and Larry Matthies. Attenuating stereo pixel-locking via affine window adaptation. In *IEEE International Conference on Robotics and Automation*, pages 914 – 921, May 2006.

[141] Laura A Stevens, Mark D Behn, Jeffrey J McGuire, Sarah B Das, Ian Joughin, Thomas Herring, David E Shean, and Matt A King. Greenland supraglacial lake drainages triggered by hydrologically induced basal slip. *Nature*, 522(7554):73–76, 2015.

[142] C. et al. Stoker. Analyzing Pathfinder data using virtual reality and superresolved imaging. *Journal of Geophysical Research*, 104(E4):8889–8906, April 1999.

[143] Changming Sun. Fast stereo matching using rectangular subregioning and 3d maximum-surface techniques. *International Journal of Computer Vision*, 47(1):99–117, Apr 2002. ISSN 1573-1405. URL https://doi.org/10.1023/A:1014585622703.

[144] Vytas SunSpiral, D.W. Wheeler, Daniel Chavez-Clemente, and David Mittman. Development and field testing of the footfall planning system for the athlete robots. *Journal of Field Robotics*, 29(3): 483–505, 2012. ISSN 1556-4967. URL http://dx.doi.org/10.1002/rob.20410.

[145] Richard Szeliski and Daniel Scharstein. Sampling the Disparity Space Image. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26:419 – 425, 2003.

[146] Y. Tao, J. P. Muller, P. Sidiropoulos, Si-Ting Xiong, A. R. D. Putri, S. H. G. Walter, J. Veitch-Michaelis, and V. Yershov. Massive stereo-based DTM production for Mars on cloud computers. *Planetary and Space Science*, 154:30–58, May 2018.

[147] Roberto Toldo, Alberto Beinat, and Fabio Crosilla. Global registration of multiple point clouds embedding the generalized procrustes analysis into an icp framework. In *Proc. 3DPVT*, pages 109–122, 2010.

[148] Bill Triggs, Philip F. Mclauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment – a modern synthesis. *Lecture Notes in Computer Science*, 1883:298+, January 2000.

[149] Tuscon University of Arizona. The high resolution imaging science experiment. 2009. URL http://hirise.lpl.arizona.edu/.

[150] AZ U.S. Geological Survey, Flagstaff. Integrated software for imagers and spectrometers (ISIS). 2009. URL http://isis.astrogeology.usgs.gov/.

[151] T. R. Watters, M. S. Robinson, R. A. Beyer, J. F. Bell, M. E. Pritchard, M. E. Banks, E. P. Turtle, N. R. Williams, and LROC Team. Lunar Thrust Faults: Implications for the Thermal History of the Moon. In S. Mackwell and E. Stansbery, editors, *Lunar and Planetary Science Conference 41*, number #1863. Lunar and Planetary Institute, Houston, March 2010.

[152] W. A. Watters and A. C. Radford. 3-D Morphometry of Martian Secondary Impact Craters from Zunil and Gratteri. In *Lunar and Planetary Science 45*, volume 45 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2014.

[153] W. A. Watters, L. Geiger, and M. Fendrock. Shape Distribution of Fresh Martian Impact Craters from High-Resolution DEMs. In *Lunar and Planetary Science Conference 44*, volume 44 of *Lunar and Planetary Institute Science Conference Abstracts*, March 2013.

[154] W. A. Watters, L. Geiger, M. Fendrock, R. Gibson, and A. Radford. Statistical Morphometry of Small Martian Craters: New Methods and Results. In *Issues in Crater Studies and the Dating of Planetary Surfaces*, volume 1841, page 9032, May 2015.

[155] Michael J Willis, Andrew K Melkonian, and Matthew E Pritchard. Outlet glacier response to the 2012 collapse of the matusevich ice shelf, severnaya zemlya, russian arctic. *Journal of Geophysical Research: Earth Surface*, 120(10):2040–2055, 2015.

[156] Jiang Xiang, Ziyun Li, David Blaauw, Hun Seok Kim, and Chaitali Chakrabarti. Low complexity optical flow using neighbor-guided semi-global matching. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 4483–4487. IEEE, 2016.

[157] V. Yershov. A system for generating multi-resolution Digital Terrain Models of Mars based on the ESA Mars Express and NASA Mars Reconnaissance Orbiter data. In *European Planetary Science Congress*, pages EPSC2015–343, October 2015.

[158] Vladimir Yershov, Anton Ivanov, Jan-Peter Muller, Yu Tao, Mr, William Pool, Jung-Rack Kim, and Panagiotis Sidiropoulos. Assessment of Digital Terrain Model algorithms for the development of a massive processing system for all high-resolution stereo images of Mars from CTX and HiRISE. In *40th COSPAR Scientific Assembly*, volume 40, pages B0.8–11–14, January 2014.

[159] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *European conference on computer vision*, pages 151–158. Springer, 1994.

[160] M. T. Zuber, D. E. Smith, S. C. Solomon, D. O. Muhleman, J. W. Head, J. B. Garvin, J. B. Abshire, and J. L. Bufton. The Mars Observer laser altimeter investigation. *Journal of Geophysical Research*, 97(E5):7781–7797, May 1992.