The Ames Stereo Pipeline:

NASA's Open Source Automated Stereogrammetry Software

A part of the NASA NeoGeography Toolkit

Version 1.0.4

Michael J. Broxton
Ross A. Beyer
Zachary Moratto
Mike Lundy
Kyle Husmann

Intelligent Robotics Group
NASA Ames Research Center

May 25, 2011

Credits

This open source version of 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.

Principal Investigator, NASA Ames Planetary Mapping and Modeling Team

- Michael J. Broxton (NASA/Carnegie Mellon University)
  `michael.broxton@nasa.gov`

Lead Developer & Stereo Pipeline Project Lead

- Zachary Moratto (NASA/Stinger-Ghaffarian Technologies)
  `z.m.moratto@nasa.gov`

Development Team

- Dr. Ross Beyer (NASA/SETI Institute)
- Matthew Hancher (NASA)
- Kyle Husmann (Educational Associates Program/California Polytechnic State University)
- Mike Lundy (NASA/Stinger-Ghaffarian Technologies)
- Dr. Ara Nefian (NASA/Carnegie Mellon University)
- Alex Segal (Educational Associates Program/Stanford University)
- 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: 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 Dr. Laurence Edwards, 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 Principle 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 this version of the 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.

Finally, we thank Melissa Bunte, Dr. Ara Nefian, and Dr. Laurence Edwards for their contributions to this documentation, 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 grant #07-LASER07-0148, and the ESMD Lunar Mapping and Modeling Program (LMMP).

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

1 Introduction **1**

1.1 Background . . . 1

1.2 Human vs. Computer: When to Choose Automation . . . 2

1.3 Software Foundations . . . 3

1.3.1 NASA Vision Workbench . . . 3

1.3.2 The USGS Integrated Software for Imagers and Spectrometers . . . 3

1.4 Getting Help . . . 4

1.5 Typographical Conventions . . . 4

1.6 Referencing the Ames Stereo Pipeline in your own work . . . 5

1.7 Warnings to users of the Ames Stereo Pipeline . . . 5

I Getting Started **7**

2 Installation **9**

2.1 Binary Installation . . . 9

2.1.1 Quick Start . . . 9

2.1.2 Common Traps . . . 10

2.2 Source Installation . . . 11

2.2.1 Dependency List . . . 11

2.2.2 Build System . . . 12

2.3 Settings Optimization . . . 14

3 Tutorial: Processing Mars Orbiter Camera Imagery **17**

3.1 Quick Start . . . 17

3.2 Preparing the Data . . . 17

3.2.1 Loading and Calibrating Images using ISIS . . . 17

3.2.2 Aligning Images . . . 18

3.3 Running the Stereo Pipeline . . . 20

3.3.1 Setting Options in the `stereo.default` File . . . 20
3.3.2 Performing Stereo Correlation . . . 21
3.3.3 Diagnosing Problems . . . 21
3.4 Visualizing the Results . . . 24
3.4.1 Building a 3D Model . . . 24
3.4.2 Building a Digital Elevation Model . . . 24
3.4.3 Generating Color Hillshade Maps . . . 26
3.4.4 Building Overlays for Moon and Mars mode in Google Earth . . . 27

II The Stereo Pipeline in Depth **29**

4 Correlation **31**
4.1 Pre-processing . . . 31
4.2 Disparity Map Initialization . . . 33
4.2.1 Debugging Disparity Map Initialization . . . 33
4.3 Sub-pixel Refinement . . . 34
4.4 Triangulation . . . 36

5 Bundle Adjustment **39**
5.0.1 A deeper understanding . . . 40
5.1 Performing bundle adjustment with isis_adjust . . . 41
5.1.1 Options . . . 42
5.2 Visualizing bundle adjustment with bundlevis . . . 44
5.2.1 Options . . . 45
5.2.2 Controls . . . 46
5.3 Examples of Use . . . 47
5.3.1 Processing Mars Orbital Camera . . . 47
5.3.2 Processing with Ground Control Points . . . 49
5.3.3 Sharing Data with ISIS 3's qnet program . . . 50

6 Data Processing Examples **51**
6.1 Guidelines for Selecting Stereo Pairs . . . 51
6.1.1 Combatting long run times . . . 51
6.1.2 Comparing Examples to your System . . . 52
6.2 Mars Reconnaissance Orbiter HiRISE . . . 53
6.2.1 Columbia Hills . . . 54

6.2.2 East Mareotis Tholus . . . 57
6.2.3 North Terra Meridiani Crop . . . 59
6.3 Mars Reconnaissance Orbiter CTX . . . 60
6.3.1 North Terra Meridiani . . . 61
6.4 Mars Global Surveyor MOC-NA . . . 63
6.4.1 Ceraunius Tholus . . . 63
6.4.2 North Tharsis . . . 65
6.5 Lunar Reconaissance Orbiter LROC NAC . . . 67
6.5.1 Lee-Lincoln Scarp . . . 67
6.6 Apollo 15 Metric Camera Images . . . 68
6.6.1 Ansgarius C . . . 69
6.7 MESSENGER MDIS . . . 71
6.7.1 Wide Angle on flyby 2 . . . 71
6.8 Cassini ISS NAC . . . 73
6.8.1 Rhea . . . 73

III Appendices **77**

A Tools **79**
A.1 stereo . . . 79
A.1.1 Entry Points . . . 80
A.1.2 Decomposition of Stereo . . . 80
A.2 disparitydebug . . . 80
A.3 point2dem . . . 81
A.4 point2mesh . . . 82
A.5 orbitviz . . . 84
A.6 isis_adjust . . . 85
A.7 bundlevis . . . 86
A.8 cam2map4stereo.py . . . 87

B The `stereo.default` File **89**
B.1 Preprocessing . . . 89
B.2 Correlation . . . 91
B.3 Filtering . . . 92
B.4 Post-Processing . . . 93

C Guide to Output Files **95**

D Modifying SURF to output VW match files **97**

D.1 How to apply and compile 97

D.2 Example of using SURF 97

Bibliography **99**

Chapter 1

Introduction

The NASA Ames Stereo Pipeline (ASP) is a suite of automated geodesy and stereogrammetry tools designed for processing planetary imagery captured from orbiting and landed robotic explorers on other planets. It was designed to process stereo imagery captured by NASA spacecraft and produce cartographic products including digital elevation models (DEMs), ortho-projected imagery, and 3D models. These 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.

Although initially developed for ground control and scientific visualization applications, the Stereo Pipeline has evolved in recent years 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 imagery 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) Web site lists 1525 stereo pairs at the time of this writing [20]. Of these, only a few tens of stereo pairs have been processed to date; mostly on human-operated, high-end photogrammetric workstations. It is clear that much more value could be extracted from this valuable raw data if a more streamlined, efficient process could be developed.

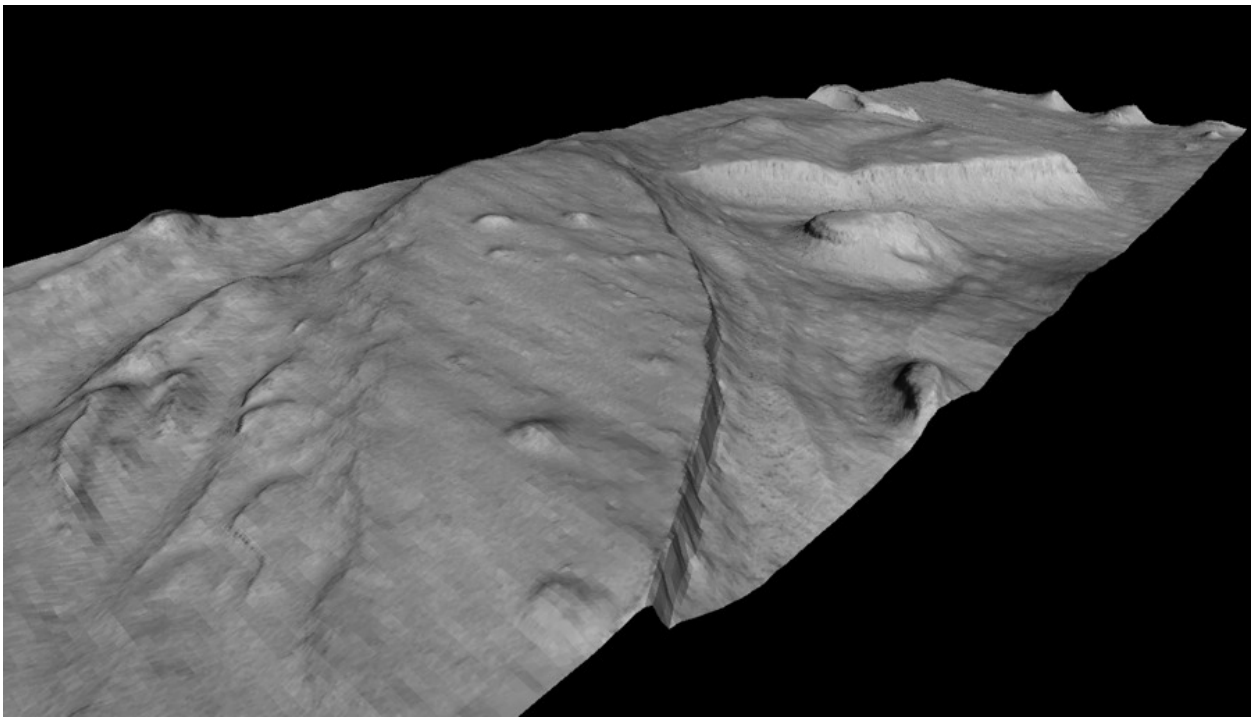


Figure 1.1: This 3D model was generated from a Mars Orbiter Camera (MOC) image pair M01/00115 and E02/01461 (34.66N, 141.29E). The complete stereo reconstruction process takes approximately thirty minutes on a 3.0 GHz workstation for input images of this size (1024 × 8064 pixels). This model, shown here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

The Stereo Pipeline was designed to address this very need. By applying recent advances in robotics and computer vision, we have created an *automated* process that is capable of generating high quality 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 imagery, 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 DEM; 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. Please don't hesitate to contact us if you have any questions!

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, but any distribution of the Stereo Pipeline requires the Vision Workbench. Unless you're compiling the Vision Workbench and Stereo Pipeline from source, the distinctions probably won't matter to you.

1.3.2 The USGS Integrated Software for Imagers and Spectrometers

This version of the Stereo Pipeline must be installed alongside a compatible version of the United States Geological Survey (USGS) Integrated Software for Imagers and Spectrometers (ISIS). ISIS is widely used in the planetary science community for processing raw spacecraft imagery into high level data products of scientific interest such as map projected and mosaicked imagery [1, 8, 21]. 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 work flow. 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.

As an additional note, the Stereo Pipeline can also process arbitrary, non-ISIS images with accompanying camera information, but doing so requires a significant amount of extra work and setup. This advanced use of the software is not covered in this user's manual, however feel free to contact us if you are interested in learning more about adapting the pipeline to other stereo data sets.

1.4 Getting Help

All bugs, feature requests, and general discussion should be sent to the Ames Stereo Pipeline user mailing list:

stereo-pipeline@lists.nasa.gov

To subscribe to this list, send an empty email message with the subject 'subscribe' (without the quotes) to:

stereo-pipeline-request@lists.nasa.gov

To contact the lead developers and project manager directly, send mail to:

stereo-pipeline-owner@lists.nasa.gov

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.

Both `--help` and `-h` are really the same thing (both will get you help). 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 (or appropriate line continuation character) and a return to continue typing on a second line in your terminal.

1.6 Referencing the Ames Stereo Pipeline in your own work

Although no peer-reviewed paper or report yet exists which details the Ames Stereo Pipeline (see the warning below about this being **RESEARCH** software), if you do use this software in your work, we'd appreciate it if you referenced one or more of these abstracts:

Moratto, Z. M., M. J. Broxton, R. A. Beyer, M. Lundy, and K. Husmann. 2010. Ames Stereo Pipeline, NASA's Open Source Automated Stereogrammetry Software. *Lunar and Planetary Science Conference* **41**, abstract #2364. [ADS Abstract].

Broxton, M. J. and L. J. Edwards. 2008. The Ames Stereo Pipeline: Automated 3D Surface Reconstruction from Orbital Imagery. *Lunar and Planetary Science Conference* **39**, abstract #2419. [ADS Abstract].

1.7 Warnings to users of the Ames Stereo Pipeline

Ames Stereo Pipeline is a **RESEARCH** product. There are known bugs and incomplete features. We reserve the ability to change the API and command line options of the tools we provide. Some of the documentation is incomplete and some of it may be out of date or incorrect. 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, they have not been systematically tested or rigorously compared to other methods in the peer-reviewed literature. We have a number of efforts underway to carefully compare Stereo Pipeline-generated data products to those produced using established processes, and we will publish those results as they become available. In the meantime, **we strongly recommend that you consult us first before publishing any results based on the cartographic products produced by this software**. You have been warned!

Part I

Getting Started

Chapter 2

Installation

2.1 Binary Installation

This is the recommended method. Only two things are required:

Stereo Pipeline Tarball. The main Stereo Pipeline page is http://ti.arc.nasa.gov/tech/asr/intelligent-robotics/ngt/stereo/. Download the *Binary* option that matches the platform you wish to use. The required ISIS version is listed next to the name; choose the newest version you have available.

USGS ISIS Binary Distribution. The Stereo Pipeline depends on ISIS version 3 from the USGS. Their installation guide is at http://isis.astrogeology.usgs.gov/documents/InstallGuide. You must use their binaries as-is; if you need to recompile, you must 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. In the meantime, you should be able to sync the previous version of ISIS which should work with Stereo Pipeline. To do so, view the listing of modules that is provided via the '`rsync isisdist.wr.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.

If you have a need to keep current with ISIS, but don't want to loose the ability to use Stereo Pipeline while we update our binaries to the new current version of ISIS, you may wish to retain the version of ISIS that matches your version of Stereo Pipeline.

2.1.1 Quick Start

Fetch Stereo Pipeline

Download the Stereo Pipeline from http://ti.arc.nasa.gov/stereopipeline.

Fetch ISIS Binaries

```
rsync -azv --delete isisdist.wr.usgs.gov::isis3_ARCH_OS_VERSION/isis .
```

Fetch ISIS Data

```
rsync -azv -delete isisdist.wr.usgs.gov::isis3data/data/base data/
rsync -azv -delete isisdist.wr.usgs.gov::isis3data/data/MISSION data/
```

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 the next chapter (Chapter 3) for an example!

2.1.2 Common Traps

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.

```
stereo: error while loading shared libraries: libisis3.so:
    cannot open shared object file: No such file or directory
```

You just need to set up your ISIS environment.

```
dyld: Library not loaded: $ISISROOT/lib/libisis3.dylib
Referenced from: /some/path/goes/here/bin/program
Reason: image not found Trace/BPT trap
```

You just need to set up your ISIS environment.

```
point2mesh E0201461-M0100115-PC.tif E0201461-M0100115-L.tif
[...]
99%  Vertices:   [************************************************************] Complete!
       > size: 82212 vertices
Drawing Triangle Strips
Attaching Texture Data
zsh: bus error  point2mesh E0201461-M0100115-PC.tif E0201461-M0100115-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.

2.2 Source Installation

This method is for advanced users with moderate build system experience. Some dependencies such as ISIS and its dependencies *(like SuperLU, Qwt, CSpice)* use custom build systems. Because of this and time, we won't help with questions on how to build dependencies.

2.2.1 Dependency List

This is a list of the direct dependencies of Stereo Pipeline. Some libraries (like ISIS) have dependencies of their own which are not covered here.

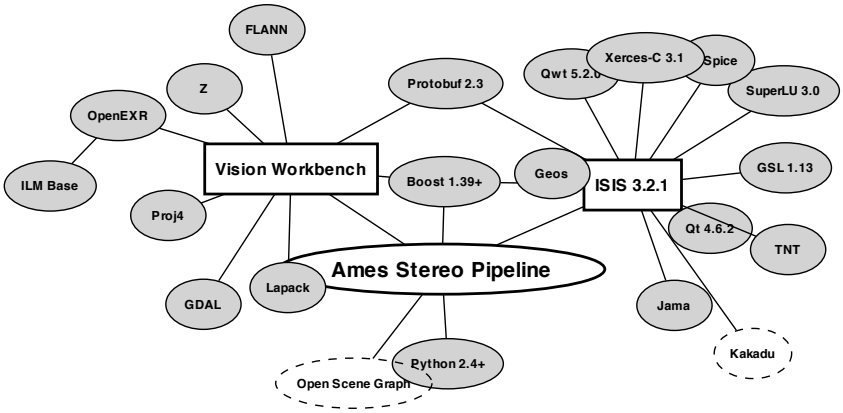


Figure 2.1: Graph outlining some dependencies. Not all of ISIS's are shown.

Boost (Required) http://www.boost.org/

Version 1.35 or greater is required. Along with the base library set, the Stereo Pipeline specifically requires: Program Options, Filesystem, Thread, and Graph.

LAPACK (Required)

There are many sources for LAPACK. For OSX, you can use the vecLib framework. For Linux, you can use the netlib LAPACK/CLAPACK distributions, or Intel's MKL, or any of a number of others. The math is unfortunately not a hotspot in the code, though, so using a faster LAPACK implementation will not change much. Therefore, you should probably just use the LAPACK your package manager (RPM for Red Hat Linux, Yast for SuSE, etc.) has available.

ISIS (Recommended) http://isis.astrogeology.usgs.gov/documents/InstallGuide

The USGS Integrated Software for Imagers and Spectrometers (ISIS) library. This library handles the camera models and image formats used for instruments. ISIS is usually downloaded and used as a binary distribution. Compilation of ISIS from source can be challenging, and their support forums may provide assistance: https://isis.astrogeology.usgs.gov/IsisSupport/. Cleaning and modification of their source code maybe required if you would like to use a newer version of ISIS's dependencies that may be available already by your system.

Because of the extreme difficulty of building ISIS (since most problems will needed be sorted out by the user themselves), it is not recommended that users try to build ISIS themselves. This leaves users the only option of using ISIS and ASP binaries that are available online.

OpenSceneGraph (Optional) http://www.openscenegraph.org/

OpenSceneGraph is required to run the `point2mesh` tool (See Section A.4). This library provides a convenient way of building OpenGL graphics through the method of scene graphs. It also provides a file format and utilities for display these scene graphs. The output file of `point2mesh` is an OpenSceneGraph binary scene graph format.

Vision Workbench (Required) http://ti.arc.nasa.gov/visionworkbench/

Vision Workbench forms much of the core processing code of the Stereo Pipeline. Vision Workbench contains almost all of the image processing algorithms, such as image filters, image arithmetic, stereo correlation, and triangulation. This means that Stereo Pipeline is just a collection of applications that implement Vision Workbench in the context of ISIS.

Python 2.4+ (Required) http://www.python.org

Some applications of Stereo Pipeline are actually python scripts.

2.2.2 Build System

The build system is built on GNU autotools. In-depth information on autotools is available from http://sources.redhat.com/autobook/. The basics, however, are simple. To compile the source code, first run `./configure` from the top-level directory. This will search for the dependencies and enable the modules you requested. There are a number of options that can be passed to `configure`; many of these options can also be placed into a `config.options` file (in the form of `VARIABLE="VALUE"`) in the same directory as `configure`. Table 2.2 lists the supported options.

Variable Name	Configure option	Default	Function
`PREFIX`	`--prefix`	/usr/local	Set the install prefix (ex: binaries will go in $PREFIX/bin)
`HAVE_PKG_XXX`	`--with-xxx`	auto	Set to "no" to disable package XXX, or a path to only search that path
`PKG_PATHS`	`--with-pkg-paths`	many	Prepend to default list of search paths
`ENABLE_PKG_PATHS_DEFAULT`	`--enable-pkg-paths-default`	yes	Append built-in list of search paths
`ENABLE_OPTIMIZE`	`--enable-optimize`	3	Level of compiler optimization?
`ENABLE_DEBUG`	`--enable-debug`	no	How much debug information?
`ENABLE_CCACHE`	`--enable-ccache`	no	Use `ccache` if available
`ENABLE_RPATH`	`--enable-rpath`	no	Set RPATH on built binaries and libraries
`ENABLE_ARCH_LIBS`	`--enable-arch-libs`	no	Pass in 64 or 32 to look for libraries by default in `lib64` or `lib32`
`ENABLE_PROFILE`	`--enable-profile`	no	Use function profiling?
`PKG_XXX_CPPFLAGS`			Append value to CPPFLAGS for package XXX
`PKG_XXX_LDFLAGS`			Prepend value to LDFLAGS for package XXX
`PKG_XXX_LIBS`			Override the required libraries for package XXX
`PKG_XXX_MORE_LIBS`			Append to required libraries for package XXX
`ENABLE_EXCEPTIONS`	`--enable-exceptions`	yes	Use C++ exceptions? Disable at own risk.
`ENABLE_MULTI_ARCH`	`--enable-multi-arch`	no	OSX Only: Build *Fat* binary with space-separated list of arches
`ENABLE_AS_NEEDED`	`--enable-as-needed`	no	Pass –as-needed to GNU linker. Use at your own risk.

Table 2.2: Supported configure options

2.3 Settings Optimization

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

Vision Workbench is a multithreaded image processing library used by Stero Pipeline. The settings by which Vision Workbench processes is configurable by having a `/.vwrc` file hidden in your home directory. Below is an example of one used on a Mac Pro workstation.

```
# This is an example VW log configuration file. Save
# this file to ~/.vwrc to adjust the VW log
# settings, even if the program is already running.
#
# 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>
#
# Example: For the console log, turn on InfoMessage
# logging for the thread sub-system and log every
# message from the cache sub-system.

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

[logfile console]
20 = thread
* = cache
# Below turns off all progress bars to the console.
0 = *.progress
```

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.

default_num_threads (default=2)

This sets the maximium number of threads that can being 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 be 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 systems. 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. Remember that 32 bit systems can only allocate 4 GB of memory per process. Despite Stereo Pipeline being a multithreaded application, it is still a single process.

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. For 32 bit systems again, they can run out of memory if this value is too high for the same reason as described for `default_num_threads`.

system_cache_size (default=805306368)

Accessing a file from 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. For 32 bit systems, this means don't set this value any greater than 1 GB. The units of this property is in bytes.

0 = *.progress

This line is not assigning a value to progress, it is however setting the logging level of progress bars. In the above example, this statement is made under the `[logfile console]` state. This means that only progress bars of type `ErrorMessage` will ever be printed to the console. If you wanted progress bars up to type `InfoMessage`, then the line in log file should be changed to:

```
[logfile console]
20 = *.progress
```

Chapter 3

Tutorial: Processing Mars Orbiter Camera Imagery

3.1 Quick Start

The Stereo Pipeline package contains command-line programs that convert a stereo pair in ISIS *cube* format into a 3D "point cloud" image: *stereo-output*-PC.tif. 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 or a gridded digital elevation model for GIS purposes.

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 image_file1 image_file2 stereo-output
```

You can then make a 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 stereo-output-L.tif
```

3.2 Preparing the Data

The data set that is used in the tutorial and examples below is a pair of MOC [11, 10] 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 data/MOC/ or the examples/MOC/ directory of your Stereo Pipeline distribution.

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, like so:

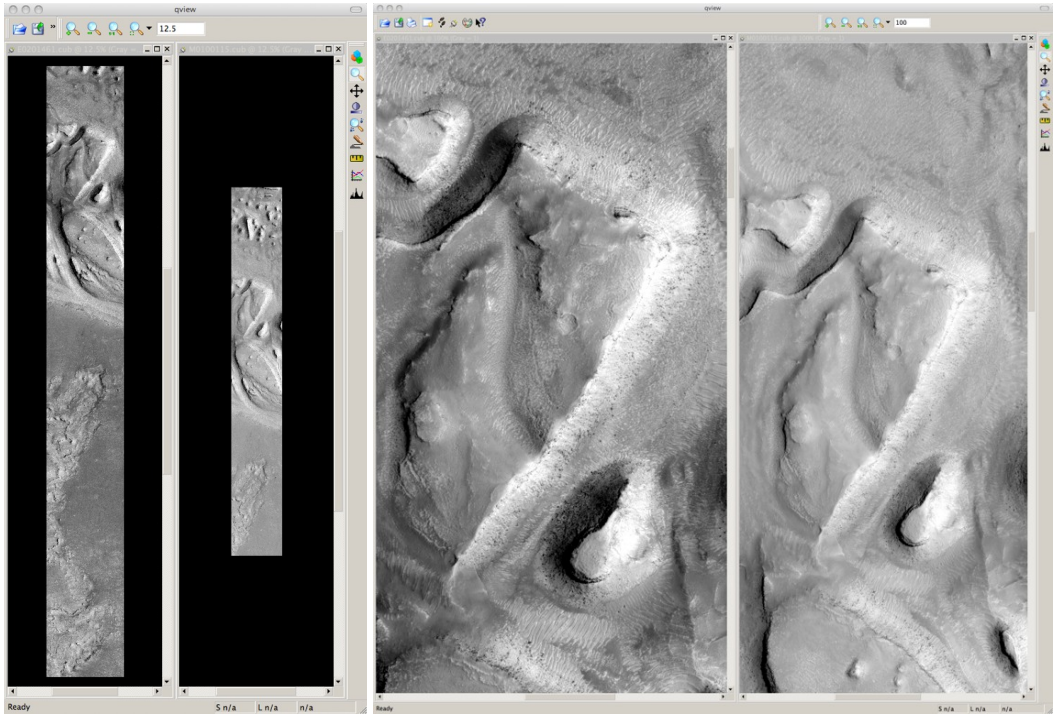


Figure 3.1: This figure shows `E0201461.cub` and `M0100115.cub` open in ISIS's qview program. The view on the left shows their full extents at the same zoom level, showing how they have different ground scales. The view on the right shows both images zoomed in on the same feature.

```
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.

3.2.2 Aligning Images

The images also need to be rectified (or aligned). There are many ways to do this (including using `DO_INTERESTPOINT_ALIGNMENT` in `stereo`'s `stereo.default` file). The most straightforward process is to align the images by map projecting them in ISIS. This example continues with the files from above, `E0201461.cub` and `M010015.cub`.

This section describes the theory behind doing each of these steps, but we also provide the `cam2map4stereo.py` program (page 87) which performs these steps automatically for you.

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 5, page 39) 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 fr= 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 fr= 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 aren't radically different in areal coverage, so this isn't really necessary for these images, its just an example.

Again, unless you are doing something complicated, using the `cam2map4stereo.py` program (page 87) will take care of all these steps for you.

3.3 Running the Stereo Pipeline

Once the data has been prepared for processing, we invoke the the `stereo` program (page 79). The `stereo` program can generate a number of output files, and you may find it helpful to create a directory to store the results of stereo processing, as illustrated below.

```
ISIS 3> ls
E0201461.cub   E0201461.map.cub   M0100115.cub   M0100115.map.cub
ISIS 3> mkdir results
```

3.3.1 Setting Options in the `stereo.default` File

The `stereo` program requires a `stereo.default` file that contains settings that effect the stereo reconstruction process. Its contents can be altered for your needs; details are found in appendix B on page 89. 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 a configuration 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.

There is a `stereo.default` file included with the example data set that is different from the example `stereo.default.example` file distributed with the Stereo Pipeline. The `stereo.default` included with the example data has a smaller correlation window (smaller values for the `H_CORR_*` and `V_CORR_*` variables) that is more suited to the MOC data.

Alternatively, it is possible to not have to define the `H_CORR_*` and `V_CORR_*` in `stereo.default`. If no such options are specified, `stereo` will attempt to guess the correct search range. The guess is printed along with the rest of the program output. If this technique does not produce satisfactory results, then it can at least be used as a starting point for picking a better search range by hand.

For this example use the `stereo.default` that is included with the example data set. It has these key properties:

```
DO_INTERESTPOINT_ALIGNMENT 0
H_CORR_MIN -35
H_CORR_MAX -15
V_CORR_MIN -280
V_CORR_MAX -265
```

The first says, *'Don't do interest point alignment!'* since we have map-projected the images. The other four lines define the range that should be searched by scanning a template from the left image over the

right image. The values above are tuned to the range of offsets that are found in this particular set of map projected images.

Given that we map projected the images using the same settings, you may be wondering why there would still be an offset. The reason is twofold: (1) the camera position may be slightly off, resulting in slight mis-alignment between stereo images; or (2) ISIS doesn't have a perfect surface to project onto during map projection, so small terrain features still produce changes in perspective. (In fact, these are precisely the features we are hoping to detect!)

> Given the uncertainties due to (1) and (2) above, it can be tricky to select a good search range for the `stereo.default` file. One way is to let `stereo` perform one round of auto search range search. Look at the results using the `disparitydebug` program. The output images will clearly show good data or bad data depending on whether the search range is correct. If the edges of these images look degraded, then the search range may need to be expanded by hand.
>
> The worst case scenario is to determine search range by hand by opening both images in qview and comparing the coordinates of points that you can match visually. Subtract locations from the first image from the second image, will yield offsets that must be in the search range. Build a bounding box around several of these sampled offsets and then expand the box by 50%. This will produce good results in most images. Future versions of the software will try to better assist the user in automatically determining the correlation window.

3.3.2 Performing Stereo Correlation

Here is how the `stereo` program is invoked:

```
ISIS 3> stereo E0201461.map.cub M0100115.map.cub results/E0201461-M0100115
```

That last option (`results/E0201461-M0100115`) is a prefix that is used when generating names for `stereo` output files. In this case the first part is `results/`, which causes the program to generate results in that directory with filename that start with `E0201461-M0100115`. If instead that last text was `E0201461-M0100115` it would have created a collection of files that start with `E0201461-M0100115` in the *same* directory as the input files.

3.3.3 Diagnosing Problems

Once invoked, `stereo` proceeds through several stages that are detailed on page 80. Intermediate and final output files are generated as it goes. See Appendix C, page 95 for a comprehensive listing. Many of these files are useful for diagnosing and debugging problems. For example, as Figure 3.2 shows, a quick look at some of the TIFF files in the `results/` directory provides some insight into the process.

Perhaps the most important file for assessing the quality of your results is the good pixel image, (`E0201461-M0100115-GoodPixelMap.tif`). If this file shows mostly good, gray pixels in the overlap area (the area that is white in both the `E0201461-M0100115-lMask.tif` and `E0201461-M0100115-rMask.tif` files), then your results are just fine. If the good pixel image shows missing 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.

Figure 3.2: These are the four viewable `.tif` files created by the `stereo` program. On the left are the two aligned, pre-processed images: (`E0201461-M0100115-L.tif` and `E0201461-M0100115-R.tif`). The next two are mask images (`E0201461-M0100115-lMask.tif` and `E0201461-M0100115-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", (`E0201461-M0100115-GoodPixelMap.tif`), which indicates (in gray) which were successfully matched with the correlator, and (in red) those that were not matched.

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 four flavors of disparity map: the `-D.tif`, the `-RD.tif`, the `-F-corrected.tif`, and `-F.tif`. You can run `disparitydebug` on any of them. Each shows the disparity map at the different stages of processing.

```
ISIS 3> cd results
ISIS 3> disparitydebug E0201461-M0100115-F.tif
```

If the output H and V files from `disparitydebug` look okay, then the point cloud image are most likely ready for post-processing. You can proceed to make a mesh or a DEM by processing `E0201461-M0100115-PC.tif` using the `point2mesh` or `point2dem` tools, respectively.

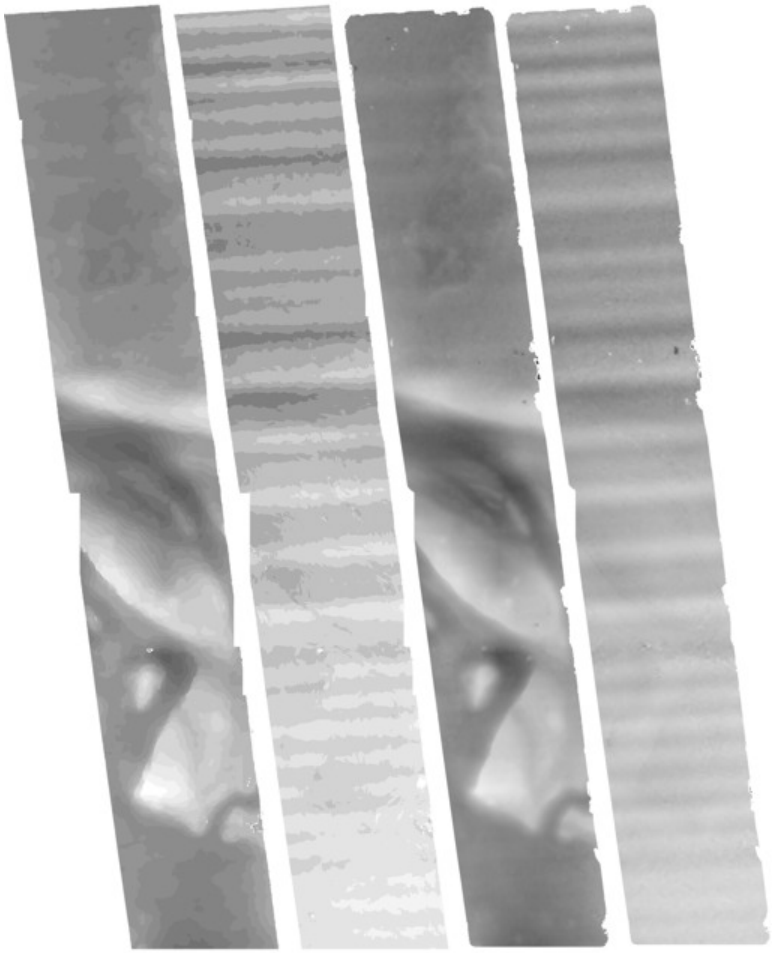


Figure 3.3: Disparity images produced using the `disparitydebug` tool. The two images on the left are the `E0201461-M0100115-D-H.tif` and `E0201461-M0100115-D-V.tif` files, which are the raw horizontal and vertical disparity components produced by the disparity map initialization phase. The two images on the right are `E0201461-M0100115-F-H.tif` and `E0201461-M0100115-F-V.tif`, which are the final filtered, sub-pixel-refined disparity maps that a 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 "wash-boarding," 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.

3.4 Visualizing 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 `E0201461-M0100115-PC.tif` point cloud file.

3.4.1 Building a 3D 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 82). The resulting file is stored in Open Scene Graph binary format [6]. 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:

```
ISIS 3> point2mesh E0201461-M0100115-PC.tif E0201461-M0100115-L.tif -l
```

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.

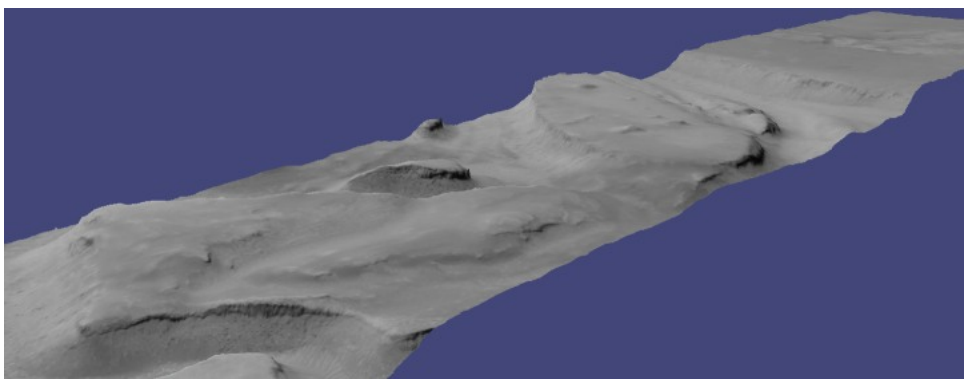


Figure 3.4: The `E0201461-M0100115.ive` file displayed in the OSG Viewer.

3.4.2 Building a Digital Elevation Model

The `point2dem` program (page 81) creates a DEM from the point cloud file.

```
ISIS 3> point2dem E0201461-M0100115-PC.tif
```

The resulting TIFF file is map projected and will contain georeferencing information stored as GeoTIFF tags. You can specify a coordinate system (e.g., mercator, sinusoidal) and a reference spheroid (i.e., calculated for the Moon or Mars).

```
ISIS 3> point2dem -r mars E0201461-M0100115-PC.tif
```

This product is suitable for scientific use, and can be imported into a variety of GIS platforms. However, the resulting file, `E0201461-M0100115-DEM.tif`, will have 32-bit floating point pixels, and will not render well in typical image viewers.

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:

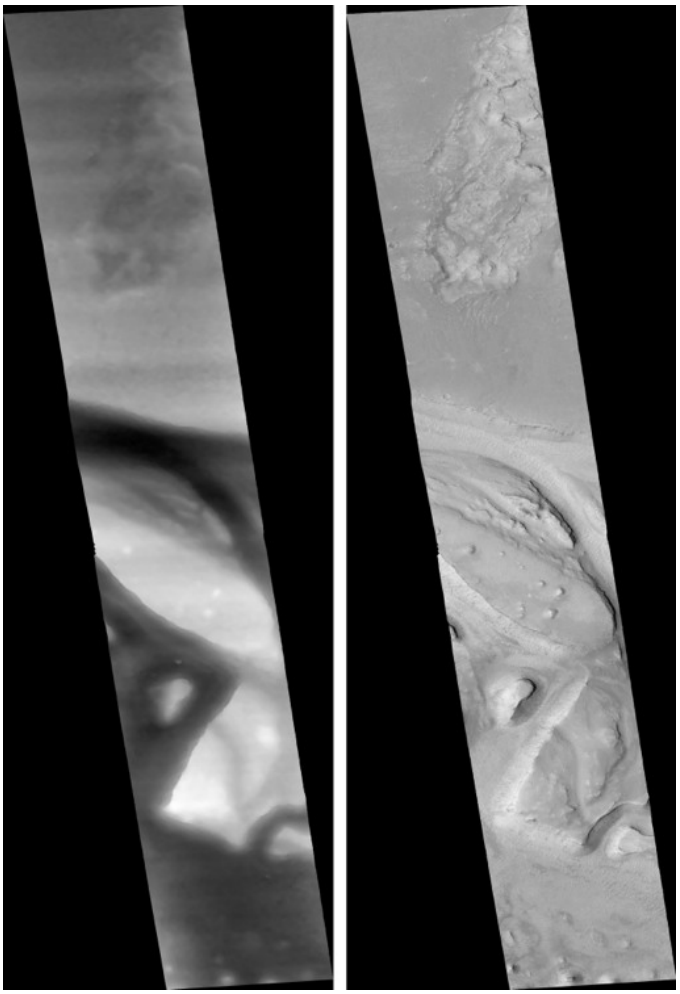


Figure 3.5: The image on the left is a normalized DEM (generated using the `-n` option), which shows low terrain values as black and high terrain values as white. The image on the right is the left input image projected onto the DEM (created using the `--orthoimage` option to `point2dem`).

```
ISIS 3> point2dem -r mars --orthoimage E0201461-M0100115-L.tif \
    E0201461-M0100115-PC.tif
```

The `point2dem` program can be used in many different ways. Be sure to explore all of the options.

3.4.3 Generating Color Hillshade Maps

Once you have generated a DEM file, you can use the Vision Workbench's `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 specific your own min and max range for the color map.

```
ISIS 3> colormap E0201461-M0100115-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.

```
ISIS 3> hillshade E0201461-M0100115-DEM.tif -o hrad-shaded.tif -e 25
```

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

```
ISIS 3> colormap E0201461-M0100115-DEM.tif -s hrad-shaded.tif -o hrad-color-shaded.tif
```

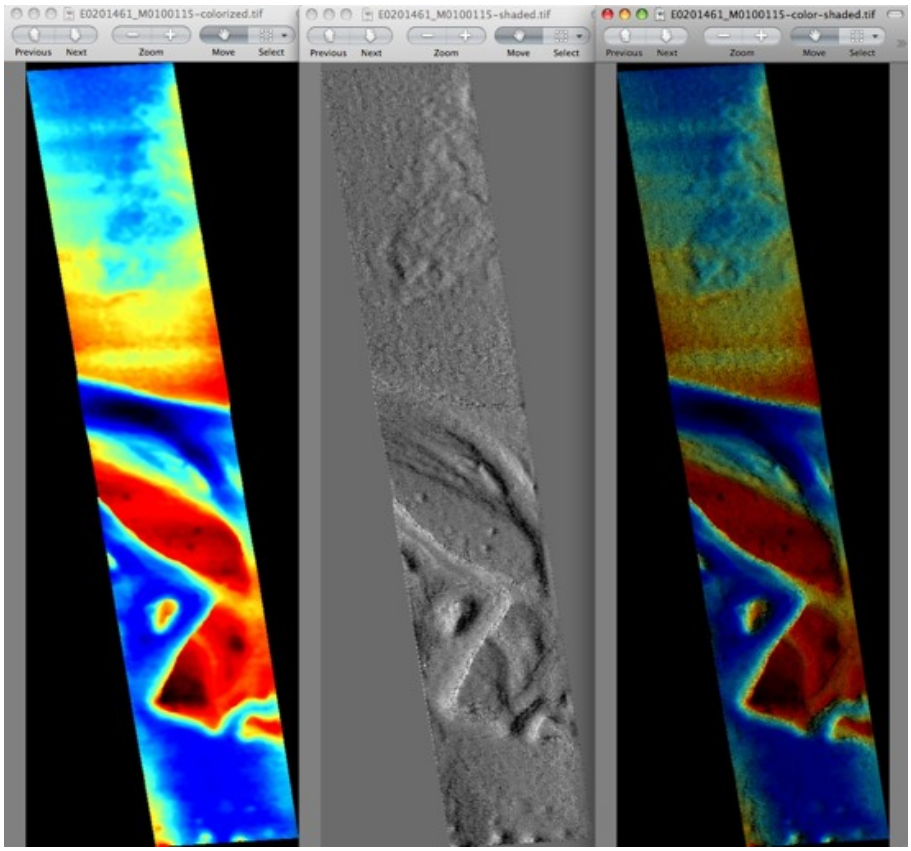


Figure 3.6: The colorized DEM, the shaded relief image, and the colorized hillshade.

3.4.4 Building Overlays for Moon and Mars mode in Google Earth

The final program in the Stereo Pipeline package that this tutorial will address is `image2qtree`. This tool was designed to create tiled, multi-resolution overlays for Google Earth. In addition to generating image tiles, it produces a metadata tree in KML format that can be loaded 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 `E0201461-M0100115-DEM-normalized.tif`, `E0201461-M0100115-DRG.tif` `hrad-shaded.tif`, `hrad-colorized.tif`, and `hrad-shaded-colorized.tif`

```
ISIS 3> image2qtree hrad-shaded-colorized.tif -m kml --draw-order 100
```

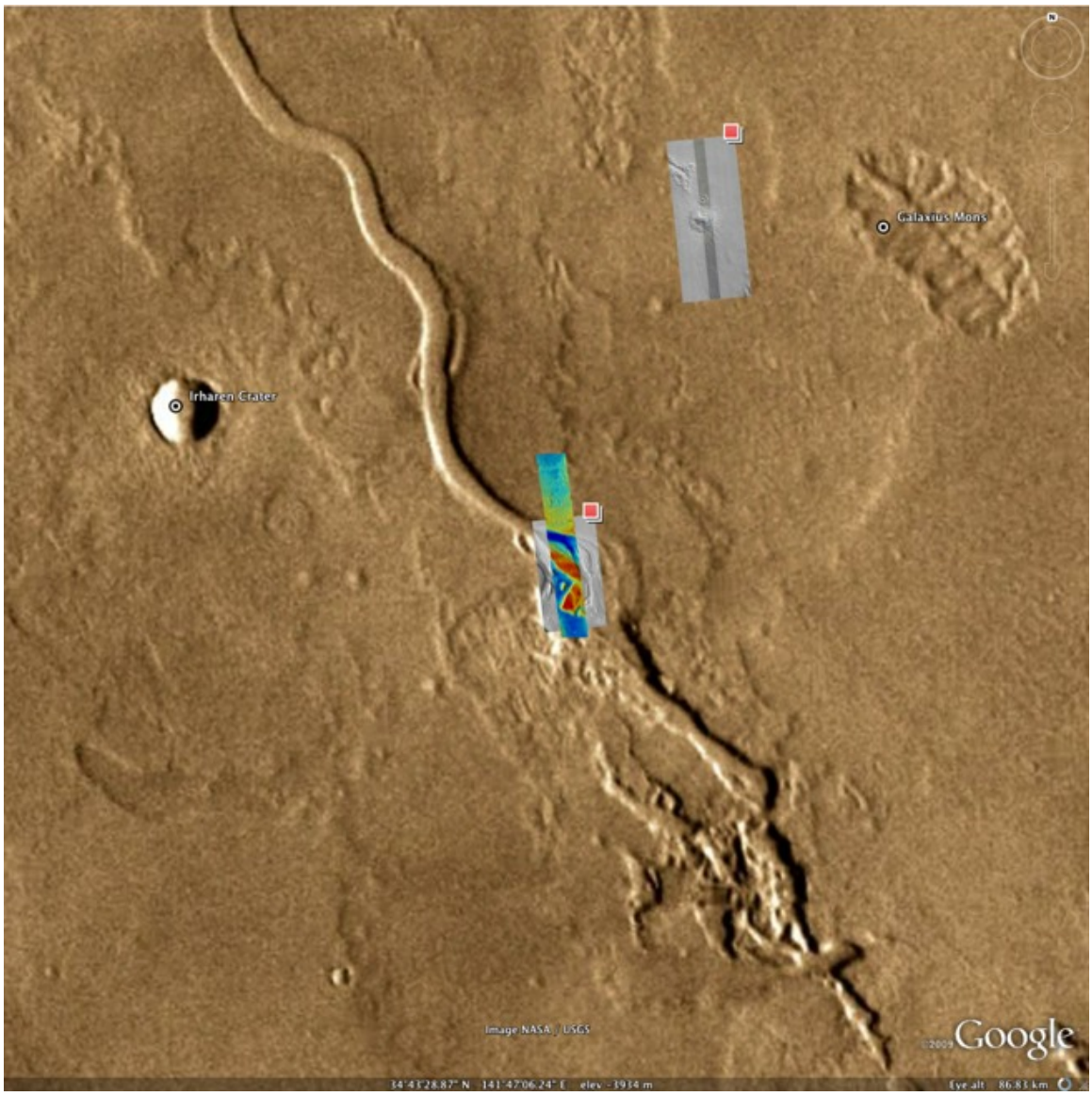


Figure 3.7: The colorized hillshade DEM as a KML overlay.

Part II

The Stereo Pipeline in Depth

Chapter 4

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. 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 4.1, and detailed in the following sections.

4.1 Pre-processing

The first optional (but recommended) step in the process is least squares Bundle Adjustment, which is described in detail in Chapter 5.

Next, the left and right images are roughly aligned using one of two methods: (1) a linear transform of the right image based on automated tie-point measurements, or (2) map projection of both the left and right images using the ISIS `cam2map` command. The former option is done automatically by the stereo pipeline when the `DO_INTERESTPOINT_ALIGNMENT` variable in the `stereo.default` file is turned on.

The latter option, running `cam2map` (or `cam2map4stereo.py`), 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 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-res terrain model into account (e.g. the Mars Orbiter Laser Altimeter, MOLA, 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.

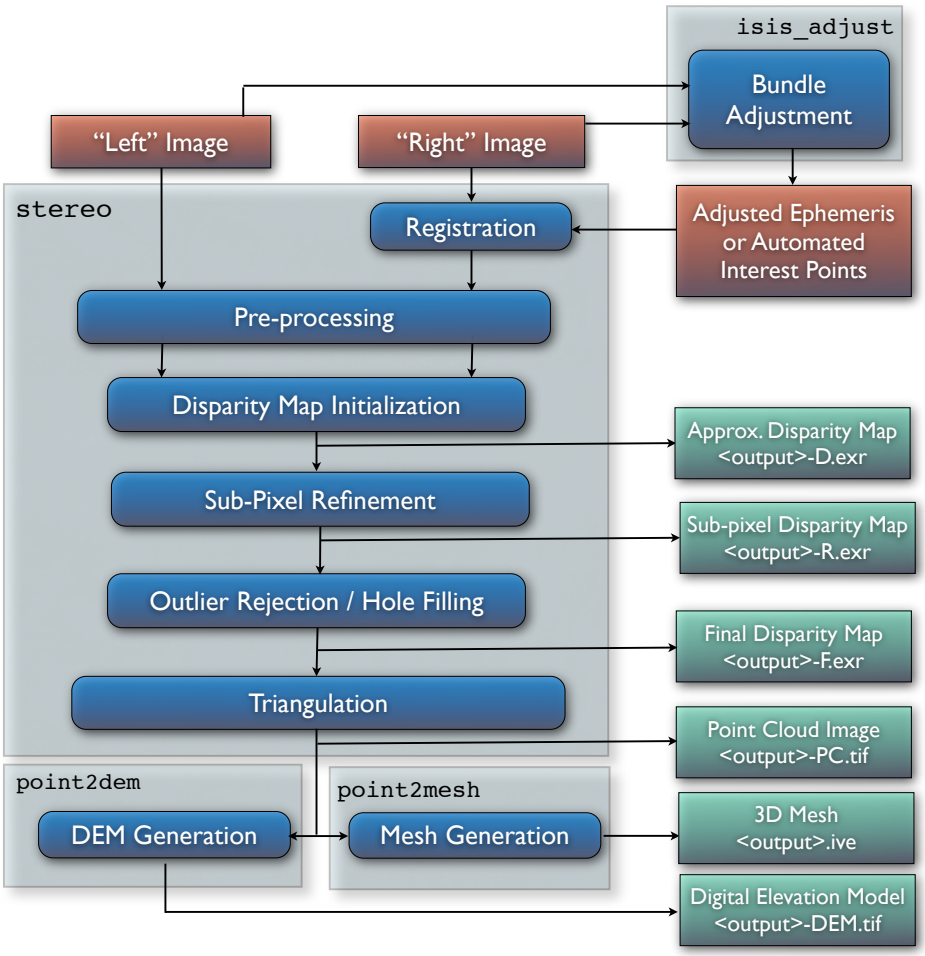


Figure 4.1: Flow of data through the Stereo Pipeline.

For this reason, we highly recommend map projection for pre-alignment of most stereo pairs. 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 effect whether or how normalization is carried out, including `DO_INDIVIDUAL_NORMALIZATION` 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 6 to see if these are necessary in your use case.

Finally, pre-processing can perform some filtering of the input images (as determined by `PREPROCESSING_FILTER_MODE`) to reduce noise and extract edges in the images. When active, these filters apply a Gaussian blur with a sigma of `SLOG_KERNEL_WIDTH` that can improve results for noisy images[1]. 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 [15]. We recommend that you use the defaults for these parameters to start with, and then experiment only if your results are suboptimal.

[1] `PREPROCESSING_FILTER_MODE` must be chosen carefully in conjunction with `COST_MODE`. (See Appendix B)

4.2 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. For large images (e.g. from HiRISE or Lunar Reconnaissance Orbiter Camera, LROC), this is very, 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 this speed for accuracy. 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 [17]; (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 [17].

Correlation itself is carried out by sliding a small, rectangular template window from the from left image over the specified search region of the right image, as in Figure 4.2. The "best" match is determined by applying a cost function that compares the two windows. The `COST_MODE` variable allows you to choose one of three cost functions, though we recommend normalized cross correlation [12], since it is most robust to slight lighting and contrast variation in between a pair of images.

4.2.1 Debugging Disparity Map Initialization

Not all pixels will be successfully matched during stereo matching. Matching may fail due to a variety of reasons:

- 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 absolute brightness.
- 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.

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

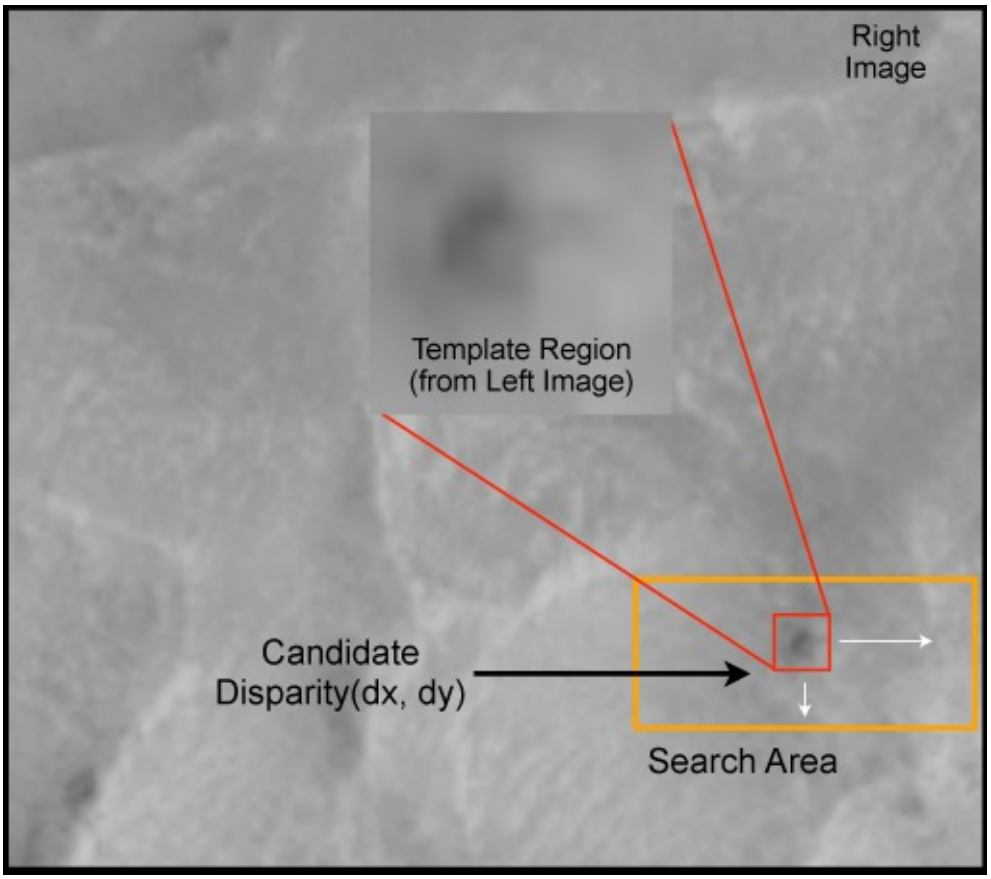


Figure 4.2: The correlation algorithm in disparity map initialization uses a sliding template window from the left image to find the best match in the right image. The size of the template window can be adjusted using the `H_KERN` and `V_KERN` parameters in the `stereo.default` file, and the search range can be adjusted using the {`H`,`V`}`_CORR_`{`MIN`/`MAX`} parameters.

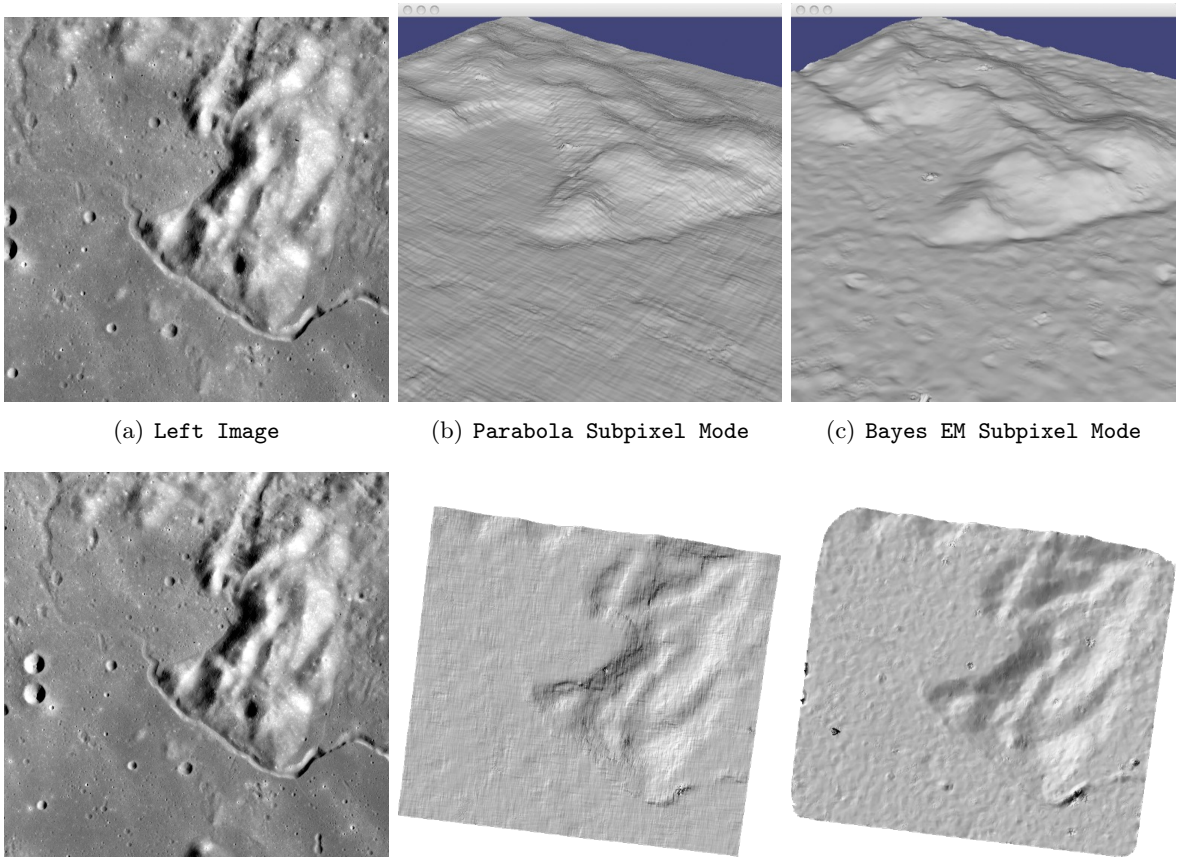
of disparity. You can open these in a standard image viewing application and see immediately which pixels were matched successfully, and which were not. Stereo matching blunders are usually also obvious when inspecting these images. With a good intuition for the effects of various `stereo.default` parameters and a good intuition for reading the output of `disparitydebug`, it is possible to quickly identify and address most problems.

4.3 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 [16, 18]. See e.g. Figure 4.3(b). Furthermore, the parabola subpixel



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

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

Figure 4.3: 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`).

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.

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.

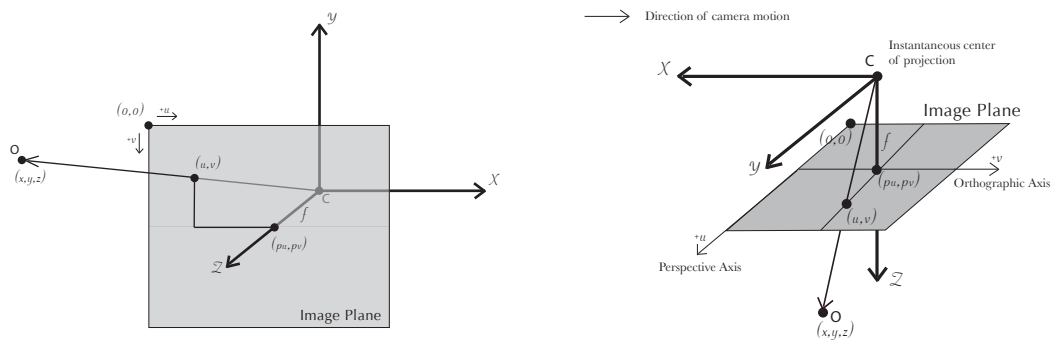
This affine-adaptive behavior is based on the Lucas-Kanade template tracking algorithm, a classic algorithm in the field of computer vision [3]. We have extended this technique; developing a Bayesian model that

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 [14, 5].

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.

4.4 Triangulation

The Stereo Pipeline uses geometric camera models available in ISIS [2]. 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.



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

Figure 4.4: 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.

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 4.4).

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

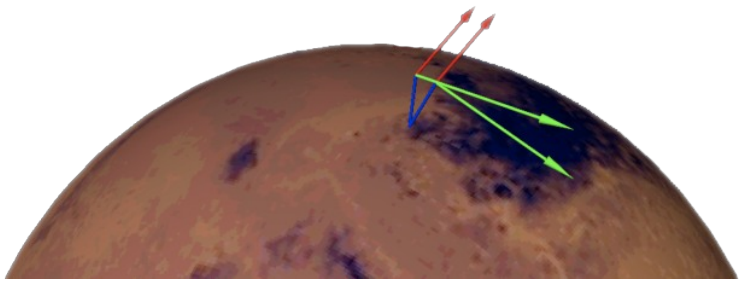


Figure 4.5: Once a disparity map has been generated and refined, it can be used in combination with the geometric camera models to compute the locations of 3D points on the surface of Mars. This figure shows the position (at the origins of the red, green, and blue vectors) and orientation of the Mars Global Surveyor at two points in time where it captured images in a stereo pair.

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 4.5). 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.

Chapter 5

Bundle Adjustment

Satellite position and orientation errors have a direct effect on the accuracy of digital elevation models produced by the Stereo Pipeline. If they're 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.

The Stereo Pipeline includes a powerful suite of tools for correcting camera position and orientation errors 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 location of the 3D objects and their actual measured location in the captured images.

That complex process can be boiled down to this simple idea: bundle adjustment ensures that observations in multiple different 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.

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,

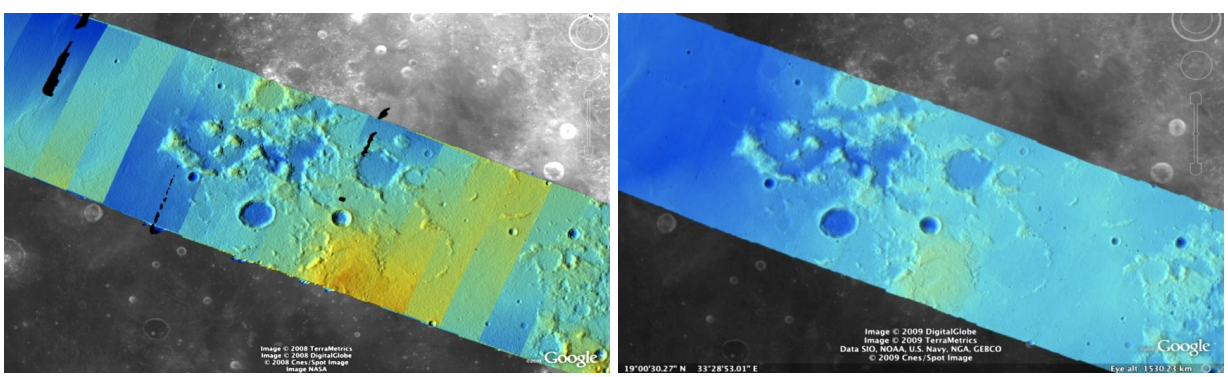


Figure 5.1: Bundle adjustment is illustrated here using a color-mapped, hill-shaded DEM mosaic from Apollo 15 Orbit 33 imagery. (a) Prior to bundle adjustment, large discontinuities can exist between overlapping DEMs made from different images. (b) After bundle adjustment, DEM alignment errors are minimized, and no longer visible.

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 ASPs to register to those types of sources.

5.0.1 A deeper understanding

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, like they sound, tie individual camera images together. Their physical manifestation would be a rock or small crater than can be observed across multiple images.

Tie-points can be automatically extracted using Vision Workbench's Interest Point module or through a number of outside methods such as the famous SURF[4]. Creating a tie point is a three step process. First, all images are processed for their natural *'interesting'* points. In most algorithms, an interest point is defined as a place where image gradients accumulate together into a peak. These interesting points are then described by the texture that surrounds them. Finally the described interesting points are matched across the multiple images. A single matched pair of interest points is now a tie point to use in bundle adjustment. In application, there is also a little filtering of the tie points with RANSAC [7].

Our bundle adjustment approach follows the method described in [19] 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. We reduce this error with the Levenberg-Marquardt algorithm (LMA). Speed is improved by using sparse methods as described in Hartley and Zisserman [9].

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. Sadly, 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 effect the cost function. This fault comes from bundle adjustment's inability to control scale and translation of the solution. It will correct geometric shape of the problem. Yet it can not guarantee that solution will have correct scale and translation.

We attempt 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. In the cost functions discussed above, errors are weighted by the inverse covariance of the measurement that gave rise to the constraint.

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.

5.1 Performing bundle adjustment with isis_adjust

First off, let it be known that USGS's ISIS has its own bundle adjustment software called `jigsaw`. It has a long history and is supported by a team of skilled developers. Despite this, Ames Stereo Pipeline provides an alternative called `isis_adjust` that has its own benefits.

Like `jigsaw`, the `isis_adjust` program is designed to perform bundle adjustment on images supported by ISIS 3 software package. The `isis_adjust` program does not discriminate based on camera type. It can perform bundle adjustment on images from line-scan imagers like MOC, Lunar Reconnaissance Orbiter Camera (LROC) NAC, HiRISE, and the Context Camera (CTX). The `isis_adjust` program can also perform bundle adjustment on traditional frame cameras (e.g. Apollo Metric Camera). Theoretically it should also work with push-frame imagers like the Thermal Emission Imaging System (THEMIS) VIS, and LROC WAC, though this is untested.

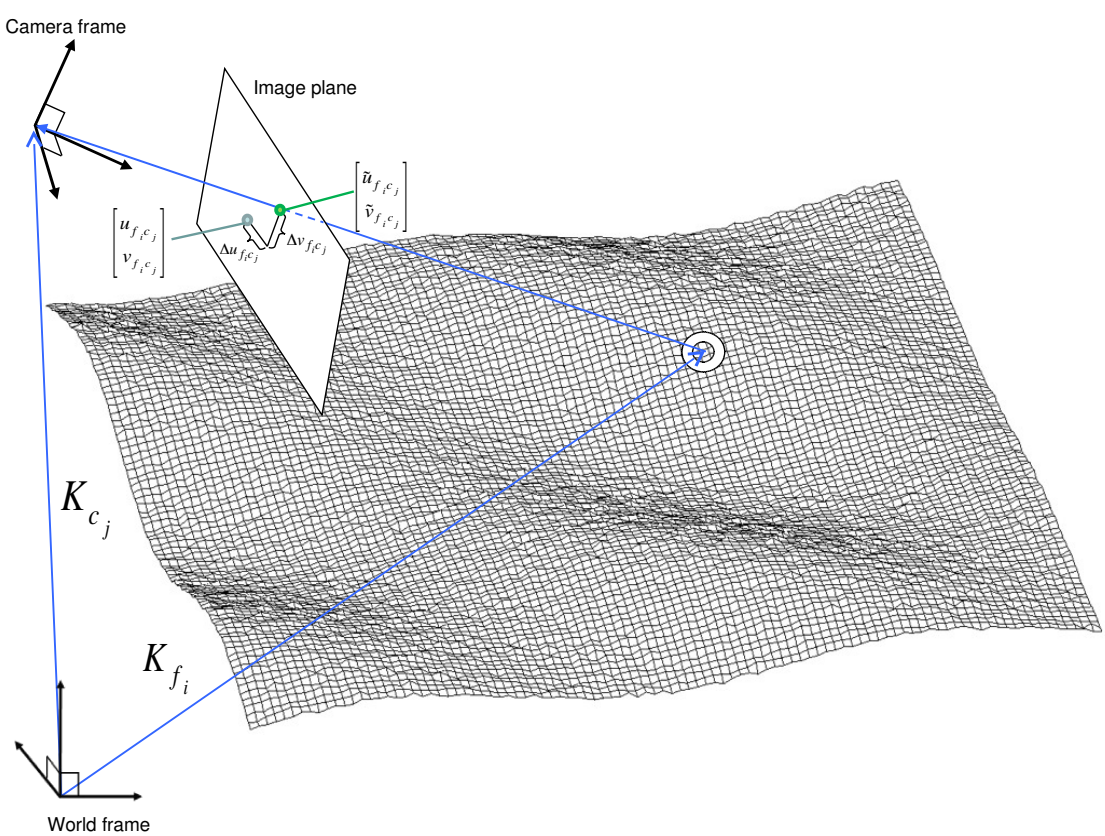


Figure 5.2: A feature observation in bundle adjustment, from Moore et al. [13]

The `isis_adjust` program works by first converting all pixel measurements in an image to measurements defined on the ideal focal plane using millimeters and the ephemeris time (ET). The ET is the absolute second at which that pixel measurement was recorded on the camera. For a frame camera, all of the pixels are captured at the same time so the ET will be identical for all measurements on the image. For pushbroom, pushframe, or other cameras which build up their 'image' over time, different parts of the image will have different ET values. For example, on a MOC image between the first and last line about 5 seconds of ET will have elapsed.

When `isis_adjust` calculates the partial derivatives of the forward projection of a point, it uses an ideal pinhole camera model. The properties of this model are defined as properties of the subject camera at the specified ET for the current measure plus the correction function, $f(t)$, that `isis_adjust` is solving for. Many forms of $f(t)$ could be used; the only limit is the number of parameters in the equations. Some initial work hints that anything greater than a second order polynomial becomes an ill-posed problem, but we hope to investigate this further in the future.

Our `isis_adjust` implements multiple bundle adjustment algorithms which can be selected from the command line. It implements the standard sparse algorithm that can be found in literature. It also implements our research work into robust cost functions. This allows the algorithm to ignore measurements that it thinks to be outliers.

Finally, `isis_adjust` stores its solution for a camera as a delta that can be added to the camera's original SPICE in a separate file. Those files have the extension `*.isis_adjust`. This is less than ideal and one feature that we value in ISIS's `jigsaw`. Their software overwrites the SPICE information in the cube file. This means that `jigsaw` results can be used in other ISIS programs like `cam2map`. This is not the case with `isis_adjust`'s results.

5.1.1 Options

The following is a listing and explanation of the options that can be given to `isis_adjust` on the command line. These options are not required.

`--cnet|-c` *`control-network-file`*

Optional. This option will force `isis_adjust` to use a pre-built built control network. This control network can either be in the USGS ISIS "cnet" format or in the binary Vision Workbench format.

If no control network is supplied using this option, `isis_adjust` will look for match files in the current working directory with base filenames that match the input images. The `isis_adjust` program will then create its own control network file and save it as `isis_adjust.cnet`.

`--cost-function L1|L2|Cauchy|Huber|PseudoHuber(=L2)`

Sets the cost function used for bundle adjustment. Default is `L2` which is the normal squared error. The full list of available options are:

L1 Proportional Error

L2 Squared Error and Default Option.

Cauchy

Huber

PseudoHuber

The options towards the end of the list are robust cost functions that deal better with non-ideal data that has outliers. These robust cost functions are performed by post-weighting the original errors yet still using equations derived for squared error.

`--bundle-adjuster Ref|Sparse|RobustRef|RobustSparse|RobustSparseKGCP(=Sparse)`

Sets the bundle adjustment code to be used. The standard to use is `Sparse`, which is a traditional squared error derived method that utilizes sparse matrices to obtain speed. Here are the complete list of options:

Ref Reference implementation that doesn't use sparse methods.

Sparse Default implementation.

RobustRef

RobustSparse

RobustSparseKGCP

The ending methods are experimental student-t derived bundle adjustment algorithms. Using the experimental robust algorithm overrides the `cost-function` option. The last two bundle adjustment algorithms are still experimental and are a work in progress.

`--disable-camera-const`

Optional. This disables the camera constraint error. Useful for debugging and just exploring what are the effects of this cost function.

`--disable-gcp-const`

Optional. This disables the GCP constraint error even if GCPs are provided. Useful for debugging and just exploring what the effects are of GCPs.

`--gcp-scalar` *`multiplier(=1)`*

Optional. Sets the multiplier that is used to adjust the sigma (or uncertainty) of the ground control points. The sigmas of ground control points are defined in the GCP data file, so this option is useful when debugging for universally scaling GCP sigmas up or down.

`--lambda|-l` *`float`*

Optional. This sets the starting value for λ: the parameter in the Levenberg Marquardt (LMA) optimization algorithm that selects between Gauss-Newton optimization and gradient descent. This parameter evolves over time on its own, but this argument can be used to override its initial value. *This is an advanced setting, not recommended for normal use.*

`--min-matches` *`integer(=5)`*

Set the minimum number of tie-points that are required between a pair of images for them to be included in the control network. This option is useful for eliminating tie-points from image pairs that have only a handful of poor or erroneous matches.

`--max-iterations` *`integer(=25)`*

Sets the maximum number of iterations for bundle adjustment. The number of required iterations will vary by problem size, so this parameter allows the user to decide how much time they're willing to dedicate to the correction of the data. We have found that 20 iterations suffices for small problems with 10 or fewer images, and tens or hundreds of iterations may be required for problems with hundreds or thousands of images.

`--poly-order` *`integer(=0)`*

Optional. Sets the order of the polynomial that is used for adjustment. Using zero means only apply offset to the camera parameters that are not time dependent. That setting is recommend for frame cameras. Linescan imagers should using either a first order polynomial or a second order. Increasing this number too high can lead to a problem that is ill-defined and would prevent the algorithm from converging on a solution. *Initial work suggest that anything beyond a 2nd order polynomial would be ill-defined. 3rd order may work with a good dose of ground control points.*

`--position-sigma` *`float(=100)`*
Sets the sigma (or uncertainty) of the spacecraft position in units of meters.

`--pose-sigma` *`float(=0.1)`*
Sets the sigma (or uncertainty) of the spacecraft pose in units of radians.

`--report-level|-r` *`integer(=10)`*
Optional. Sets the report level for the final bundle adjustment report. This report is saved as `isis_adjust.report`. Report levels available are:

0 - CommandLine Error and Final report

10 - CommandLine Iteration Error (default)

20 - Write Report file

25 - *In development*

30 - Write Stereo Triangulation Error

35 - *In development*

100 - Debug, Write Error Vectors (big human readable)

110 - Debug, Write Jacobian Matrix (massive human readable)

`--robust-threshold` *`float(=10)`*
Sets the robust threshold; an additional parameter specifically for the `PseudoHuber`, `Huber`, and `Cauchy` arguments to the `--cost-function` option.

`--save-iteration-data|-s`
Optional. Use to write `bundlevis` visualization files.

`--seed-with-previous`
Optional. Loads up the previous `isis_adjust` session's adjustment file and uses them as a starting point for this session.

`--write-isis-cnet-also`
Optional. Write an ISIS Parameter Value Language (PVL) style control network file to `isis_adjust.net`. The output file is very large compared to the binary output, `isis_adjust.cnet`, but is human readable and compatible with the ISIS 3 `qnet` tool.

`--write-kml [0|1(=0)]`
Optional. Providing this option with a zero will have the program write a Keyhole Markup Language (KML) file showing the location of all the GCPs and be colored according to their final error. Providing this option with a one will have this perform as before, but also have it write all of the 3D point estimates. This is useful for debugging and for having a quick visualization of where stress points might exist in a bundle adjustment problem when there are many cameras.

`--help|-h`
Provides a shortened list of the above.

5.2 Visualizing bundle adjustment with bundlevis

The `bundlevis` program is used to visualize the process of bundle adjustment. It will show an animated, fully interactive 3D scene containing all the 3D points and cameras across all iterations of bundle adjustment.

This tool is used to quickly determine if bundle adjustment was successful. If something does go wrong, `bundlevis` can be a powerful debugging tool for identifying the problem.

Once `bundlevis` has loaded the data from a bundle adjustment run, the user can click on and inspect the position of 3D points that are in purple and play back the iterations using the keyboard. Double clicking on a camera will cause lines to be drawn to each point viewed by the camera. Clicking on a 3D point causes lines to be drawn to all cameras that view the point.

Bundle adjustment can fail in a variety of different ways, but users should be aware of two common failure modes. The first is segmentation; where tie points will split into two or more distinct groups, producing cliffs between or clumps among points. This is usually caused by insufficient matches between a pair of images in your control network. You may need to choose some tie-points between these images by hand or add additional images that overlap with the problem area.

The second common sign of a failure is a point cloud explosion (the resulting terrain looks unrecognizably noisy). This most often results from a high number of outlying, bad tie-point measurements that the bundle adjustment algorithm can't recover from. These bad constraints can be removed by hand or mitigated using one of the robust cost modes (e.g. by using the `--cost-function` argument for `isis_adjust`).

5.2.1 Options

The following is a listing and explanation of the options that can be given to `bundlevis` from the command line.

`--camera-iteration-file|-c` *`bundlevis-camera-iteration-file`*

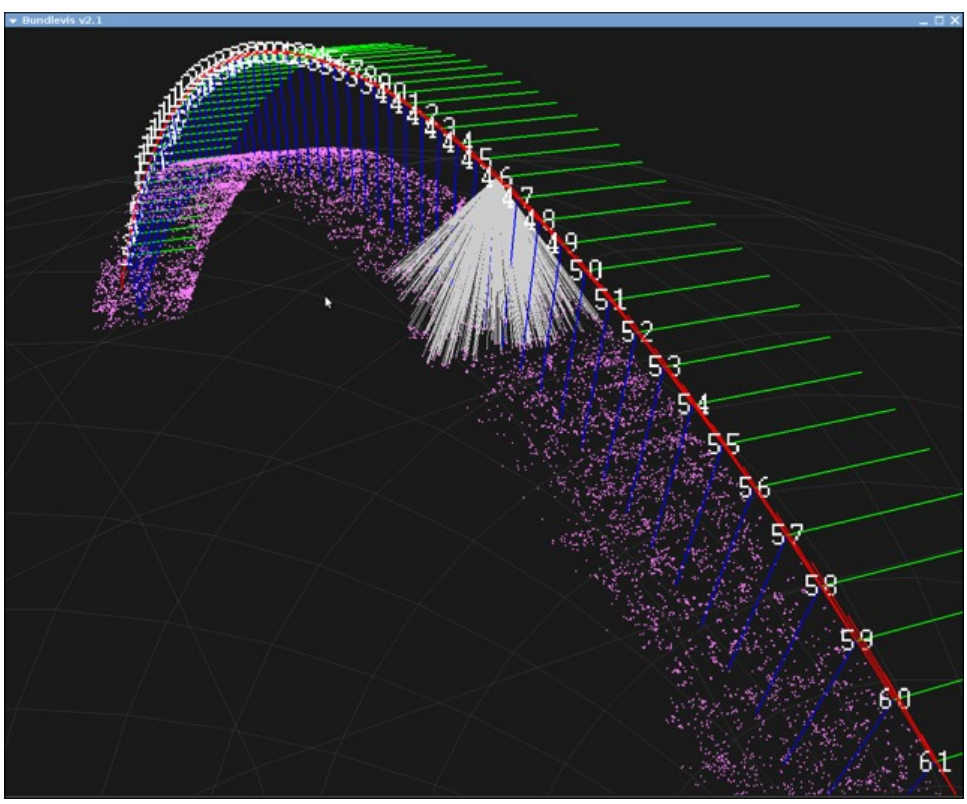


Figure 5.3: A screenshot of `bundlevis` visualizing the bundle adjustment of imagery from the Apollo 15 Metric Camera (orbit 33).

Optional. Supply a camera iteration file that was produced by `isis_adjust`. `bundlevis` will only draw cameras if you supply a camera iteration file.

`--points-iteration-file|-p` *`bundlevis-point-iteration-file`*
Optional. Supply a point iteration file that was produced by `isis_adjust`. `bundlevis` will only draw 3D points if you supply a point iteration file.

`--control-network-file|-n` *`Vision-Workbench-binary-control-network-file`*
Optional. Supply a control network file that was produced by `isis_adjust`. This allows `bundlevis` to show the relationship between points and cameras when used in conjunction with `--camera-iteration-file` and `--points-iteration-file`.

`--additional-pnt-files` *`bundlevis-point-iteration-files`*
Optional. Supply additional points to be animated alongside the camera and 3D points. The files given must be in the same format as a `bundlevis` point iteration file and have the same number of iterations.

`--fullscreen`
Optional. Displays `bundlevis` using the entire screen; otherwise the program loads in a window. *The fullscreen option does not work correctly with dual screen systems.*

`--stereo`
Optional. Render the 3D scene in red/blue anaglyph mode.

`--show-moon`
Optional. Draws a wireframe sphere with a radius of 1737.3 km that represents the Moon.

`--show-mars`
Optional. Draws a wireframe sphere with a radius of 3397 km that represents Mars.

`--show-earth`
Optional. Draws a wireframe sphere that represents the Earth.

5.2.2 Controls

Once `bundlevis` is running, there are several controls that can be used to interface with the program. There are the playback controls, jump-to-frame controls, and the mouse.

Playback Controls Playback controls are similar to those in the popular Winamp program; arranged on the keyboard like the controls on a tape deck.

Z Step back one iteration
X Play
C Pause
V Stop *(which is the same as Pause except that bundlevis goes back to iteration 0)*
B Step forward one iteration

Jump-to-Frame Controls Jump-to-Frame controls are the numbers **1-9** along the top of the keyboard. Pressing **1** will display the very first iteration. Pressing **9** will display the very last iteration. Pressing **2** through **8** will display iterations that are somewhere in between based on the value of the number. Finally, pressing **0** will cause `bundlevis` to display the points at their very last iteration with an additional tail pointing back to the starting position of the points in the first iteration.

Mouse The mouse is used to move around the model, and has the same controls found in many 3D environments (e.g. `osgviewer`). Moving the mouse with the **left mouse button** held down will cause the model to rotate. Moving with the **right mouse button** pressed will zoom, and moving with the **middle mouse button** will translate. Alternatively for systems where the mouse is button-challenged, **option** + **mouse** is translation and **command** + **mouse** is zoom.

Double clicking with the mouse on a point or camera will allow the user to query entities in the model. A double click will cause the point number and camera number to be printed to the terminal. The number identifier for a given camera or point will also appear when the viewer is zoomed in on that entity.

5.3 Examples of Use

5.3.1 Processing Mars Orbital Camera

What follows is an example of bundle adjustment using two MOC images of the south Cydonia region. We use images M10/00254 and R09/01059. 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= m1000254.imq to= m1000254.cub mapping=no
ISIS 3> mocproc from= r0901059.imq to= r0901059.cub mapping=no
```

You will 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 is perfectly parallel to the surface of its subject (and thus spherical). That makes it 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. The `isis_adjust` program does not support this.

At this point, we need to automatically generate tie-points between these two images. This can be done using the `ipfind` and `ipmatch` utilities. These tools do not reliably (currently) work with photometrically calibrated images, so we must first convert these images to a standard format using the ISIS program `isis2std`:

```
ISIS 3> isis2std from= m1000254.cub to= m1000254.png format= PNG
ISIS 3> isis2std from= r0901059.cub to= r0901059.png format= PNG
```

Here is how to process those newly created PNG files for tie-points using the Interest Point Module tools from Vision Workbench.

```
> ipfind  m1000254.png r0901059.png -g 1.2
> ipmatch m1000254.png r0901059.png -d -r homography -i 30
```

> Be aware that the tie-point tools available in Vision Workbench do not always produce enough matches for bundle adjustment. An alternative would be to use outside code such as SURF. We have included a patch at the end of the book that converts SURF output to a Vision Workbench style match file. In extreme cases, users can be forced to make measurements themselves. This can be performed with ISIS's `qnet` progam.
>
> For this example we found that `ipfind` and `ipmatch` work great. Expect to find approximately 100 matched points. Your results will be slightly different due to the random nature of RANSAC.

Finally it is time to start bundle adjustment. There are many options that can be used at this stage. We have chosen those required to create visualization data for `bundlevis`. We have also set the maximum iterations to 100 and chose the option to create a detailed report file of `isis_adjust`'s results.

```
ISIS 3> isis_adjust *.cub -s --max 30 -r 50
```

Before you hit enter, notice that we are not feeding the interest point match files to `isis_adjust`. Instead, we only provide the camera files. During run time, the program will see that it doesn't have any image measurements in the form of a control network. In that case it will attempt to build its own by searching the current working directory for `*.match` files.

Now you have permission to run `isis_adjust`. You'll see the command will produce considerable debugging output and will place many output files in the current working directory. If you look through the output in the terminal or alternatively in the output report file, `isis_adjust.report`, you'll see that the problem did not converge but reduced most of the error in the first 40 iterations (again, your results may vary slightly). The lack of convergence is worrisome and a few (like 3) ground control points would probably help considerably. Notice that the error improved only slightly after those first 40 iterations, but the shape of the pointcloud and the camera paths changed considerably. This untwisting of the cameras can be seen next in `bundlevis`.

Visualizing all of the data that was exported for `bundlevis` can be carried out as follows:

```
> bundlevis -p iterPointsParam.txt -c iterCameraParam.txt \
            -n isis_adjust.cnet
```

Press escape to exit out of `bundlevis` when finished. You may also want to try viewing the data with a wire-frame of Mars to give some perspective. Note, you will have to zoom in very far since the size of a MOC frame is quite small relative to the size of Mars!

```
> bundlevis -p iterPointsParam.txt -c iterCameraParam.txt \
            -n isis_adjust.cnet --show-mars
```

Producing a DEM using the newly created corrections is the same as covered in the Tutorial on page 17, with one small difference: `stereo` needs to know of the existence of the correction files, `m1000254.isis_adjust` and `r0901059.isis_adjust`.

```
ISIS 3> stereo m1000254.cub r0901059.cub m1000254.isis_adjust \
               r0901059.isis_adjust MOC_RESULTS/M1000254_R0901059
```

The two new arguments (`*.isis_adjust`) provide `stereo` with the necessary corrections. When providing outside camera models for images, they are always the 3rd and 4th argument for `stereo`. This is also how `stereo` can be made to operate on images from standard consumer cameras.

5.3.2 Processing with Ground Control Points

Ground control point files describe a single point in the world that is seen by 1 or more cameras. How they are measured in the first place is up to the user. We use a manual process of comparing each image to a respected map projected image and then recording the latitude, longitude, and altitude of the point(s). The maps to register against can be anything, but it is recommended to register against a product with a high amount of cartographic stability and accuracy. For terrestrial work, we would use a USGS product that can provide imagery that is registered to LIDAR height measurements.

Unlike match files, ground control points must specifically be given to `isis_adjust` from the command line, but in no particular order. Ground control point files are written with the extension `.gcp`. Below is an example of a ground control point file that was created to control a series of Apollo Metric Camera images from several Apollo 15 orbits.

```
-52.8452 27.2561 1735999 300 300 500
sub4-AS15-M-2086.cub     210.9   3565.0
sub4-AS15-M-2087.cub     1476.9  3579.0
sub4-AS15-M-2088.cub     2798.9  3586.8
sub4-AS15-M-2089.cub     4133.5  3588.6
sub4-AS15-M-2344.cub     906.9   3874.8
sub4-AS15-M-2345.cub     2204.2  3913.9
sub4-AS15-M-2482.cub     939.8   4348.0
sub4-AS15-M-2483.cub     2282.0  4340.7
sub4-AS15-M-2484.cub     3642.1  4330.9
```

The first line of a `.gcp` file is like a header line and is different from the remaining lines. The first line defines the world location of the ground control point, and the rest of the lines define the image locations of the ground control points. Here are what the columns mean for the first line.

Column 1: Longitude in degrees

Column 2: Latitude in degrees

Column 3: Radius in meters

Column 4: Sigma (or uncertainty) in meters for Local X axis

Column 5: Sigma (or uncertainty) in meters for Local Y axis

Column 6: Sigma (or uncertainty) in meters for Local Z axis

The other lines describe where this GCP is found in each image:

Column 1: Image name

Column 2: Sample (X) image measurement

Column 3: Line (Y) image measurement

Make a `.gcp` file for every ground control point, then be sure to feed them as an input to `isis_adjust`. Remember that you can scale the sigma of all ground control points by using the `--gcp-scalar` flag. This can save time by allowing you to make adjustments without needing to edit all of the files individually.

5.3.3 Sharing Data with ISIS 3's qnet program

ISIS contains a program called `qnet` whose purpose is to create and edit ISIS style control network files. To share a control network with `qnet`, you will need to save our control network in the ISIS format. If bundle adjustment has already been performed once and if we want to simply convert the control network for use in `qnet`, you can use this command to save an ISIS style control network:

```
ISIS 3> isis_adjust -c isis_adjust.cnet --write-isis-cnet-also *.cub
```

Otherwise if this is the first time performing bundle adjustment and a control network does not already exist, use:

```
ISIS 3> isis_adjust --write-isis-cnet-also *.cub
```

There should now be an `isis_adjust.net` file in the project's directory. It will be quite a bit larger than the other control network file since it is stored as ASCII text, but it can be read and edited with a text editor. Before starting `qnet`, there is one additional preparation that must be performed. ISIS's `qnet` requires a text file listing of all the cubes used by the control network. Here's how to create one:

```
> ls *.cub > list_of_cubes.lis
```

Now, start up `qnet` without any command line arguments. Click File→Open. It will first ask for the list of cubes. Refer it to the newly created `list_of_cubes.lis`. Next it will ask for the control network. Give it `isis_adjust.net`.

> At this time `qnet` does not work with the Apollo Metric Camera's cube files. When loading the text file listing of cubes it will issue an error about invalid serial numbers for the listed cube files.

When finished, save the new control network file. Here's how to use the new control network in `isis_adjust`:

```
ISIS 3> isis_adjust -c the_new_control_network.net *.cub
```

Take note that to distinguish ISIS style control network files from Stereo Pipeline style control network files is by the file extension. ISIS control networks have the extension of `.net` and can be read with a text editor. Stereo Pipeline's control networks have the extension `.cnet` and are binary format files.

Chapter 6

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 and `stereo.default` files used to process data. We hope that these are useful templates that will get you started in processing your own data.

6.1 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.

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.

6.1.1 Combatting long run times

The factor that predominantly determines running time in the Stereo Pipeline is the size of the search space considered by the correlation algorithm. These are set in the `stereo.default` file using the H_CORR_MIN, H_CORR_MAX, V_CORR_MIN, and V_CORR_MAX parameters. If you comment these parameters out (either by putting a '`#`' at the beginning of their line or deleting them from your `stereo.default` file), the Stereo Pipeline will try to automatically determine the search range for you, but this does not always work perfectly. A spurious bad match can lead the pipeline to select a search range that is far too large, and

performance will suffer as a result. If you know (or can estimate) the range of horizontal and vertical offsets you expect to see between the two images, then you may want to try setting the search range yourself in your stereo.default using the aforementioned parameters.

More generally, here are three strategies that tend to keep the search range small and run-times low:

1. You can crop your stereo pair (using the ISIS `crop` command) to a small region of interest within a large stereo pair. ISIS and the Stereo Pipeline will keep track of these crop parameters automatically and take them into account when applying the camera model during triangulation. You may want to work with a cropped pair when you first start working with a new data set. Run times will be much lower (minutes instead of days), and you can quickly tune things up before scaling things up.

2. The ISIS `reduce` command can be used to subsample the image pair. In this case, 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 pulling 3D terrain out of noisy, low quality images.

3. You can map project the images (using the ISIS `cam2map` command or the `cam2map4stereo.py` program provided with the Stereo Pipeline). If you project both images into the same map projection and same pixel scale, then they will be aligned modulo uncertainty in spacecraft telemetry (typically 10-100's of meters of error when the image is projected onto the ground). By default `cam2map` will also project the image onto the local elevation model (MOLA or LOLA), which removes the stereo disparity in the images that is due to coarse topography. The resulting image pair has only small position offsets and fine 3D detail left to discover, so the search range can be kept very small and run times can be improved. Again, ISIS and the Stereo Pipeline will keep track of how these map projections affect the camera model, and take them into account when building up the 3D mesh via triangulation. If you use `cam2map`, be sure that DO_INTERESTPOINT_ALIGNMENT = 0 in your `stereo.default`. Note also that the `--lat` and `--lon` arguments to `cam2map4stereo.py` can be used to crop your stereo images, and the `--resolution` argument can be used to subsample them.

If you are working with very large images, we highly recommend cropping or subsampling and working with smaller sized images while you fine-tune the parameters in the `stereo.default` file, and once you get satisfactory results to apply those parameters to the full images.

6.1.2 Comparing Examples to your System

Since our first release we reperformed some of these examples and recorded their processing time so you the user can judge how long it will take you. Our examples were processed on our server called 'Lunokhod 2'. This server is a Dell PowerEdge Rack 900 purchased in late 2009. Below are its specifications:

CPU	Dual E7420 Xeon at 2.13 GHz *(16 logical cores)*
FSB	1066 MHz
L2 Cache	8 MB
Memory	64 GB @ 667 MHz (mis-matched?)
Storage	Local RAID5
OS	Red Hat Enterprise Linux 5.5
BogoMIPS	4256
Color	Dell Graphite

The times recorded are listed in wall hours and CPU hours. Wall-hours are how long it took the job to complete from the user's perspective. CPU-hours are how much processing time it took to complete. If

the job took 30 wall-minutes on a 2 core system, it spent 30 minutes in CPU 1 and CPU 2. Thus, the total CPU-hours would be 1. This example, though correct, is not what always happens in the real world. Inefficiency with managing multiple threads or the complete lack of multithreaded code will bring wall hours up to CPU hours. Your required CPU hours will vary based on CPU architecture. Estimating your required CPU hours for your system can be done by scaling with the BogoMIPS measurements. This can be read from Linux systems with the command: `cat /proc/cpuinfo`

6.2 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. 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
```

To use our script, first go to the directory where you have downloaded the HiRISE's RED EDR `IMG` 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.

Finally we recommend map projecting the product and normalizing both images in the stereo pair using the same dynamic range. Notice that we map project the second image using the same map settings and crop of the first image. This means the images will share the same origin and the `stereo.default` search range can be centered around zero.

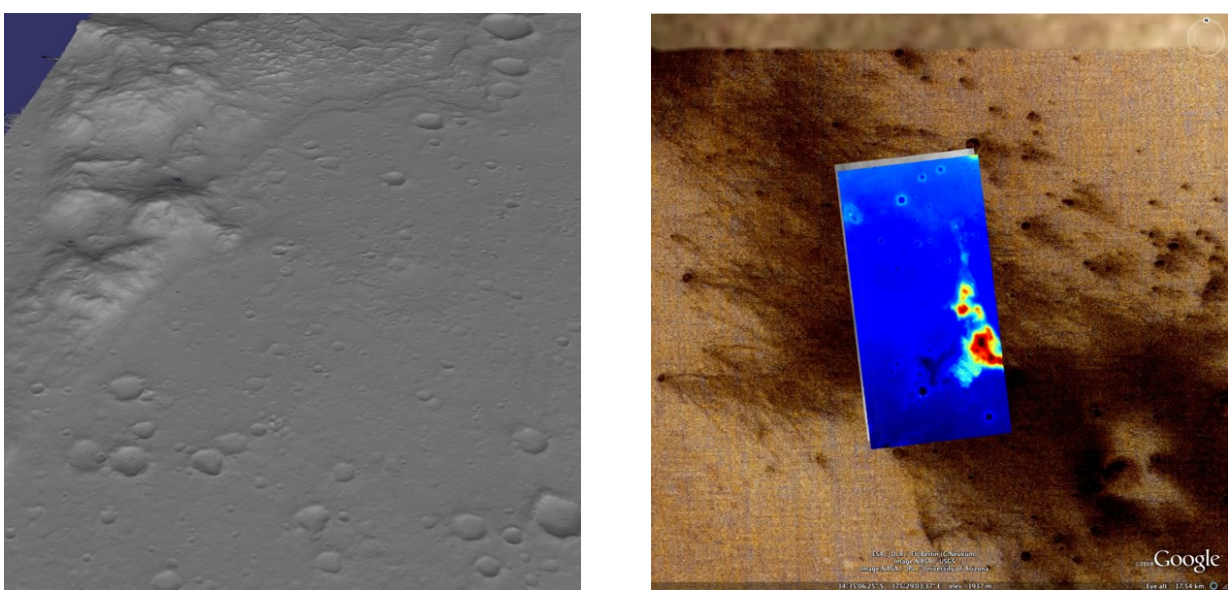
```
ISIS 3> cam2map4stereo.py first.mos_hijitreged.norm.cub second.mos_hijitreged.norm.cub
ISIS 3> bandnorm f=first.map.cub t=first.norm.cub
ISIS 3> bandnorm f=second.map.cub t=second.norm.cub
ISIS 3> ls first.norm.cub second.norm.cub > fromlist
ISIS 3> ls first.norm.cub > holdlist
ISIS 3> equalizer fromlist=fromlist holdlist=holdlist
ISIS 3> mkdir result
ISIS 3> stereo first.norm.equ.cub second.norm.equ.cub result/output
```

In the future, the HiRISE team will be producing de-jittered, noproj'ed imagery in the `extras/` directory of their PDS volume. When this happens, most of the above commands will no longer be required, as you will be able to just run `cam2map4stereo.py` on their provided imagery.

6.2.1 Columbia Hills

Prepping Files:	Wall Time	`+36:00:00.0`	CPU Time	`+36:00:00.0`
Processing in Stereo:	Wall Time	`297:28:06.0`	CPU Time	`881:39:45.54`

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 6.1: 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> bandnorm from=PSP_001513_1655_RED.map.cub \
                   to=PSP_001513_1655_RED.map.norm.cub
ISIS 3> bandnorm from=PSP_001777_1650_RED.map.cub \
                   to=PSP_001777_1650_RED.map.norm.cub
ISIS 3> rm *RED.map.cub
ISIS 3> mkdir result
ISIS 3> stereo PSP_001513_1655.map.norm.cub \
               PSP_001777_1650.map.norm.cub result/output
```

stereo.default

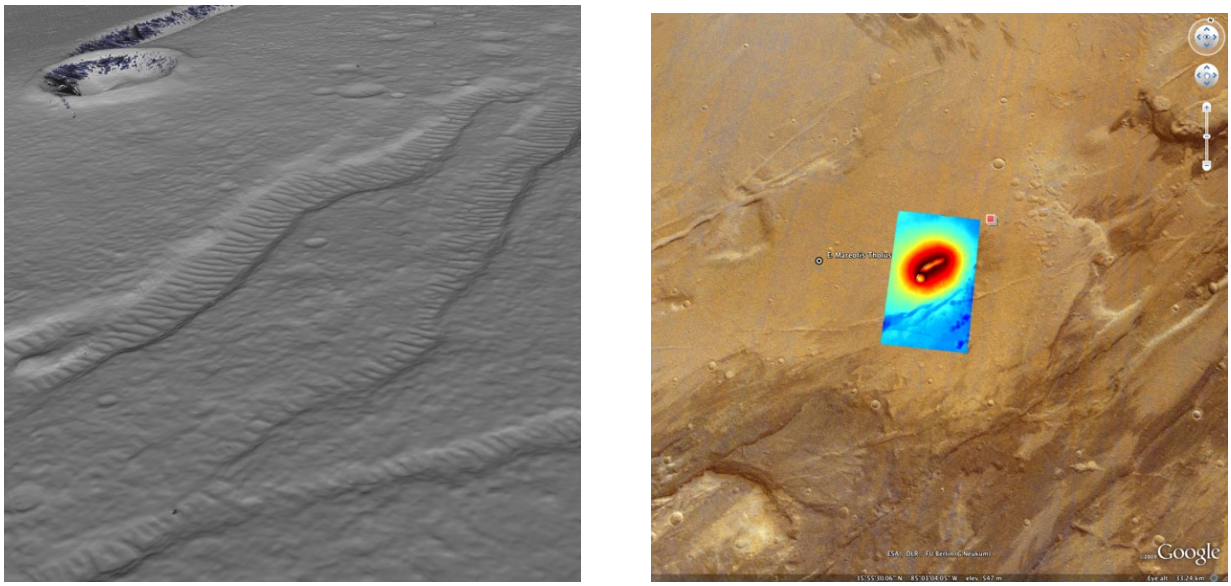
stereo.default for HiRISE Columbia Hills

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 0
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 0

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_MODE 0
COST_BLUR 0

H_KERNEL 50
V_KERNEL 50

H_CORR_MIN 210
H_CORR_MAX 450
V_CORR_MIN -320
V_CORR_MAX 320

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 25
SUBPIXEL_V_KERNEL 25

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.2.2 East Mareotis Tholus

Prepping Files:	Wall Time	04:02:11.00	CPU Time	04:05:26.81
Processing in Stereo:	Wall Time	59:51:01.00	CPU Time	164:55:02.36

HiRISE observations PSP_001760_2160 and PSP_001364_2160 cover East Mareotis Tholus, a small volcano in Tempe Terra.



(a) 3D Rendering (b) KML Screenshot

Figure 6.2: Example output using HiRISE images PSP_001364_2160 and PSP_001760_2160 of East Mareotis Tholus.

Commands

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

```
ISIS 3> hiedr2mosaic.py PSP_001364_2160_RED*.IMG
ISIS 3> hiedr2mosaic.py PSP_001760_2160_RED*.IMG
ISIS 3> cam2map4stereo.py PSP_001364_2160_RED.mos_hijitreged.norm.cub \
                          PSP_001760_2160_RED.mos_hijitreged.norm.cub
ISIS 3> bandnorm from=PSP_001364_2160_RED.map.cub \
                 to=PSP_001364_2160_RED.map.norm.cub
ISIS 3> bandnorm from=PSP_001760_2160_RED.map.cub \
                 to=PSP_001760_2160_RED.map.norm.cub
ISIS 3> ls *.map.norm.cub > fromlist
ISIS 3> ls *1760*.map.norm.cub > holdlist
ISIS 3> equalizer fromlist=fromlist holdlist=holdlist
ISIS 3> rm *RED.map.norm.cub *RED.map.cub
ISIS 3> mkdir result
ISIS 3> stereo PSP_001364_2160.map.norm.equ.cub \
               PSP_001760_2160.map.norm.equ.cub result/output
```

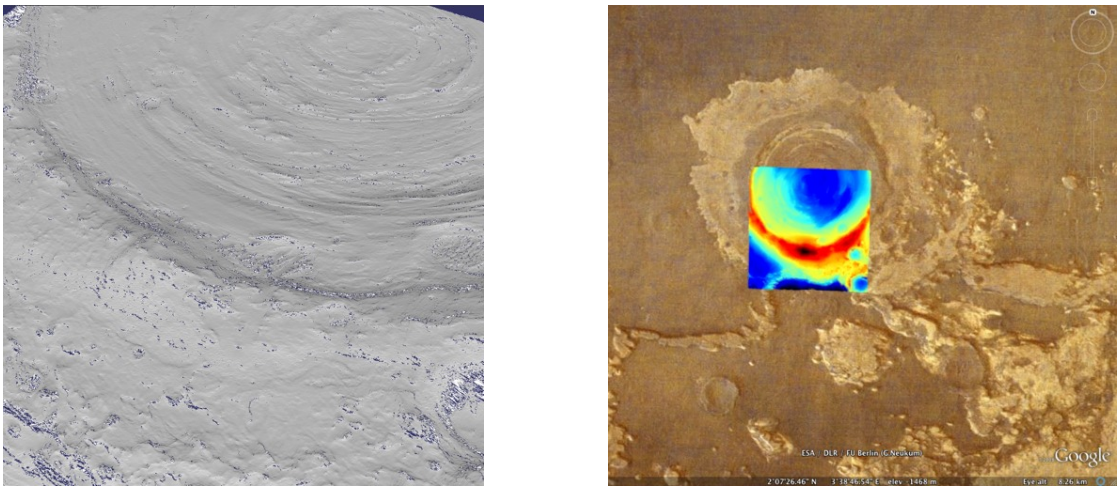
stereo.default

stereo.default for HiRISE East Mareotis Tholus

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 0
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 0

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_BLUR 0
COST_MODE 0

H_KERNEL 25
V_KERNEL 25

H_CORR_MIN -80
H_CORR_MAX 150
V_CORR_MIN -80
V_CORR_MAX 50

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 25
SUBPIXEL_V_KERNEL 25

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.2.3 North Terra Meridiani Crop

HiRISE observations PSP_001981_1825 and PSP_002258_1825 show a small crater filled by layered material.



(a) 3D Rendering (b) KML Screenshot

Figure 6.3: Example output using cropped HiRISE data of North Terra Meridiani.

Commands

Notice here that we have applied a crop to select a subset of these HiRISE images that we are interested in. Cropping is often an efficient way to go because it greatly reduces the amount of computation necessary to get results in a limited area. As always, Download all 20 of the RED EDR `.IMG` files for each observation.

```
ISIS 3> hiedr2mosaic.py PSP_001981_1825_RED*.IMG
ISIS 3> hiedr2mosaic.py PSP_002258_1825_RED*.IMG
ISIS 3> cam2map from=PSP_001981_1825_RED.mos_hijitreged.norm.cub \
                 to=PSP_001981_1825_REDmosaic.map.cub
ISIS 3> cam2map from=PSP_002258_1825_RED.mos_hijitreged.norm.cub \
                map=PSP_001981_1825_REDmosaic.map.cub \
                 to=PSP_002258_1825_REDmosaic.map.cub matchmap=true
ISIS 3> bandnorm from=PSP_001981_1825_REDmosaic.map.cub \
                  to=PSP_001981_1825_REDmosaic.map.norm.cub
ISIS 3> bandnorm from=PSP_002258_1825_REDmosaic.map.cub \
                  to=PSP_002258_1825_REDmosaic.map.norm.cub
ISIS 3> ls *.map.norm.cub > fromlist
ISIS 3> ls *1981*.map.norm.cub > holdlist
ISIS 3> equalizer fromlist=fromlist holdlist=holdlist
ISIS 3> crop from=PSP_001981_1825_REDmosaic.map.norm.equ.cub \
              to=PSP_001981_1825.crop.cub sample=7497 line=41318 nsamp=10000 nline=10000
ISIS 3> crop from=PSP_002258_1825_REDmosaic.map.norm.equ.cub \
              to=PSP_002258_1825.crop.cub sample=7982 line=41310 nsamp=10000 nline=10000
ISIS 3> rm *REDmosaic*.cub
ISIS 3> mkdir result
ISIS 3> stereo PSP_001981_1825.crop.cub PSP_002258_1825.crop.cub result/output
```

stereo.default

stereo.default for HiRISE North Terra Meridiani Crop

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 0
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 0

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_BLUR 21
COST_MODE 2

H_KERNEL 45
V_KERNEL 45

H_CORR_MIN -270
H_CORR_MAX -70
V_CORR_MIN -14
V_CORR_MAX 26

SUBPIXEL_MODE 0

SUBPIXEL_H_KERNEL 25
SUBPIXEL_V_KERNEL 25

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.3 Mars Reconnaissance Orbiter CTX

CTX is a moderate camera to work with. Processing times for CTX can be pretty long because of the file sizes involved. Luckily the disparity between images is relatively small, allowing efficient computation.

6.3.1 North Terra Meridiani

Processing in Stereo: Wall Time `13:28:04.00` CPU Time `45:54:50.10`

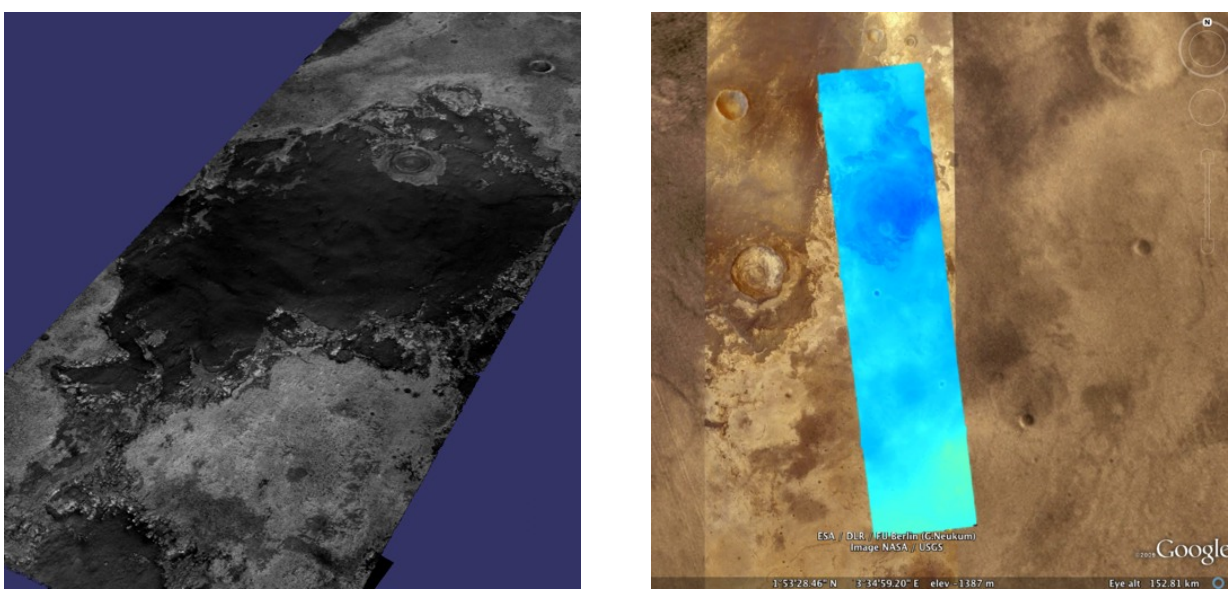
In this example, we use map projected images. Map projecting the images is the most reliable way to align the images for correlation. However when possible, use non-map-projected images with the `DO_INTERESTPOINT_ALIGNMENT` option. This greatly improves speed. For all cases using linescan cameras, triangulation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

This example is distributed in the `examples/CTX` directory.

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> mkdir result
ISIS 3> stereo P02_001981_1823.map.cub P03_002258_1817.map.cub results/out
```



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

Figure 6.4: Example output possible with the CTX imager aboard MRO.

stereo.default

stereo.default for CTX North Terra Meridiani

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 1
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 0
DO_INDIVIDUAL_NORMALIZATION 0

PREPROCESSING_FILTER_MODE 3

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_BLUR 0
COST_MODE 2

H_KERNEL 35
V_KERNEL 35

H_CORR_MIN -300
H_CORR_MAX 300
V_CORR_MIN -150
V_CORR_MAX 150

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 21
SUBPIXEL_V_KERNEL 21

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.4 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, especially Mars Global Surveyor, are particularly susceptible.

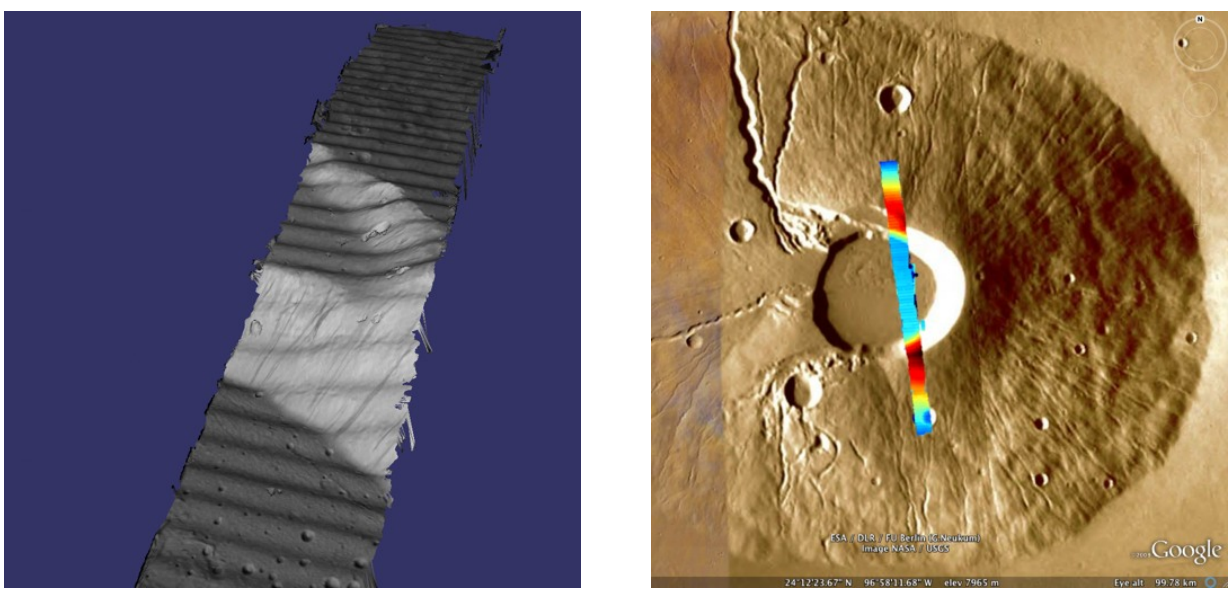
Jitter causes wave-like distortions along the track of the satellite orbit in DEMs produced from linescan camera images. This effect can be very subtle or quite pronounced, so it is important to check your data products carefully for any sign of this type of artifact. The following examples will show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

Note that the science teams of HiRISE and 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.

6.4.1 Ceraunius Tholus

Prepping Files:	Wall Time	`00:02:42.30`	CPU Time	`00:02:42.06`
Processing in Stereo:	Wall Time	`00:23:00.30`	CPU Time	`00:34:11.00`

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 6.5: 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> mkdir result
ISIS 3> stereo M0806047.map.cub R0701361.map.cub result/output
```

stereo.default

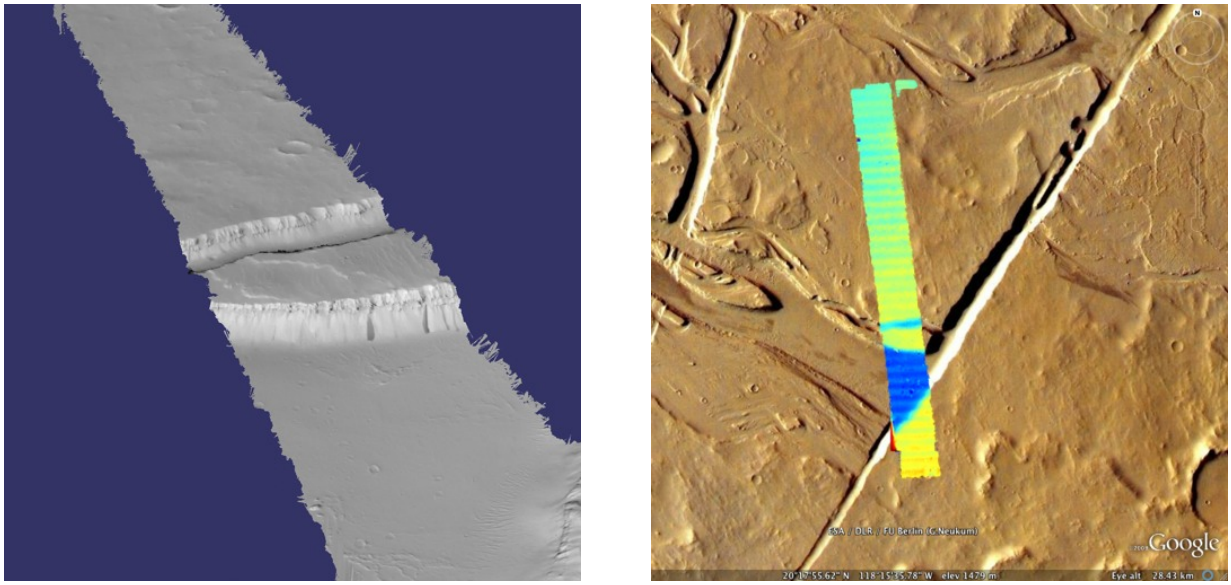
stereo.default for MOC Ceraunius Tholus

```
### PREPROCESSING

DO_INTEREST POINT_ALIGNMENT 0
INTEREST POINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 1

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_BLUR 12
COST_MODE 2

H_KERNEL 25
V_KERNEL 25

H_CORR_MIN -12
H_CORR_MAX 26
V_CORR_MIN -50
V_CORR_MAX 15

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 21
SUBPIXEL_V_KERNEL 21

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.4.2 North Tharsis

Prepping Files:	Wall Time	`00:01:57.52`	CPU Time	`00:01:56.79`
Processing in Stereo:	Wall Time	`00:24:50.70`	CPU Time	`01:34:37.40`

These images cover troughs and terraces in northern Tharsis. This DEM is located at 20.20 N and 118.18 W on Mars.



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

Figure 6.6: Example output for MOC-NA of North Tharsis.

Commands

Download the M08/03097.img and S07/01420 images from the PDS.

```
ISIS 3> moc2isis f=M0803097.img t=M0803097.cub
ISIS 3> moc2isis f=S0701420.img t=S0701420.cub
ISIS 3> cam2map4stereo.py M0803097.cub S0701420.cub
ISIS 3> mkdir result
ISIS 3> stereo M0803097.map.cub S0701420.map.cub result/output
```

stereo.default

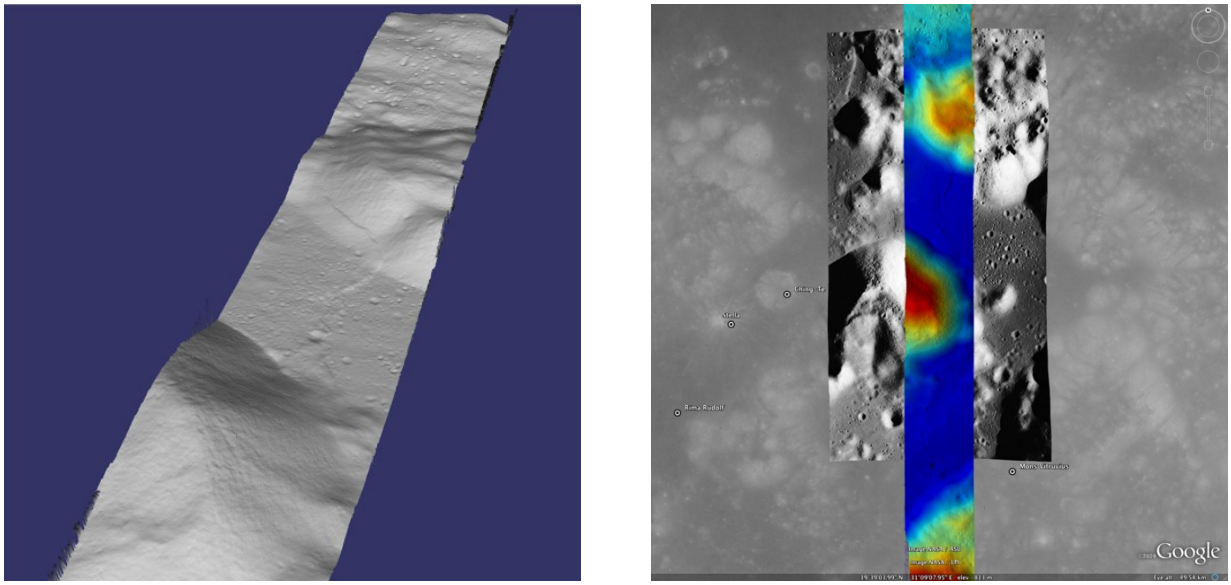
stereo.default for MOC North Tharsis

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 0
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 1

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_BLUR 12
COST_MODE 2

H_KERNEL 25
V_KERNEL 25

# Don't specify search range. Let it auto-detect.

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 21
SUBPIXEL_V_KERNEL 21

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.5 Lunar Reconaissance Orbiter LROC NAC

6.5.1 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 6.7: Example output possible with a LROC NA stereo pair, using only a single CCDs from observations.

Commands

Download the EDRs for the left CCDs for observations M104318871 and M104318871. Alternatively you can search by original IDs of 2DB8 and 4C86 in the PDS.

```
ISIS 3> -- isis lro tools yet to be released --
ISIS 3> cam2map from=M104318871LE.cal.cub to=M104318871LE.map.cub
ISIS 3> cam2map from=M104311715LE.cal.cub map=M104318871LE.map.cub \
                to=M104311715LE.map.cub matchmap=true
ISIS 3> mkdir result
ISIS 3> stereo M104318871LE.map.cub M104311715LE.map.cub result/output
```

stereo.default

stereo.default for LROC NAC

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 0
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 0

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_BLUR 12
COST_MODE 2

H_KERNEL 29
V_KERNEL 29

H_CORR_MIN -425
H_CORR_MAX 150
V_CORR_MIN -100
V_CORR_MAX 100

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 25
SUBPIXEL_V_KERNEL 25

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.6 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.

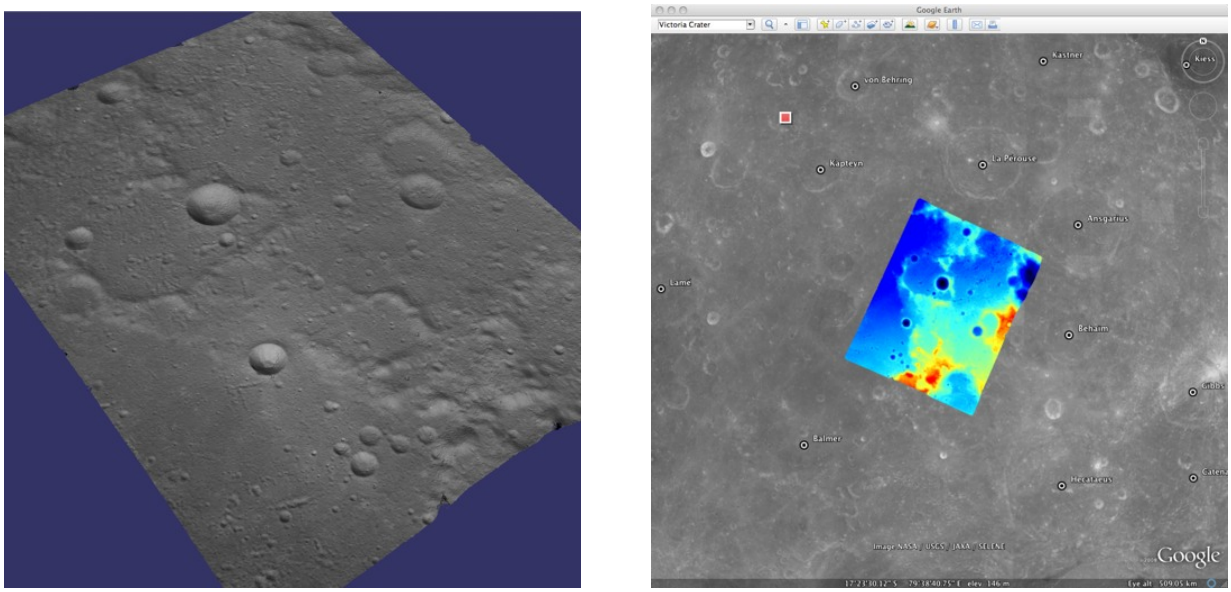
The scans performed by ASU are sufficiently detailed to exhibit film grain at the highest resolution. The amount of noise at the full resolution is not helpful for the correlator, so we recommended subsampling the images by a factor of 4.

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.

6.6.1 Ansgarius C

Prepping Files
Wall Time `00:00:02.11` CPU Time `00:00:01.29`
Processing in Stereo
Wall Time `01:52:23.00` CPU Time `21:36:07.61`

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



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

Figure 6.8: 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> mkdir result
ISIS 3> stereo sub4-AS15-M-2380.cub sub4-AS15-M-2381.cub result/output
```

stereo.default

stereo.default for Apollo 15 Metric Camera

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 1
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 0

PREPROCESSING_FILTER_MODE 3

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_MODE 2
COST_BLUR 25

H_KERNEL 35
V_KERNEL 35

H_CORR_MIN -250
H_CORR_MAX 250
V_CORR_MIN -70
V_CORR_MAX 100

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 25
SUBPIXEL_V_KERNEL 25

# Hidden advanced function
CORRSCORE_REJECTION_THRESHOLD 1.4

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.7 MESSENGER MDIS

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

For future mappers, we suggest checking out Mercury Flyby 3 data which was not available at the time of this writing. Flyby 3 and Flyby 2 seem to have covered some of the same terrain with the narrow angle camera.

6.7.1 Wide Angle on flyby 2

In most flyby imagery it is very hard to find good stereo pairs. This pair was taken from a single flyby just seconds apart. Note also that this pair is taken from different wavelengths (the letter at the end of the filename designates the current filter being used on the wide angle camera). Unfortunately there is not enough of a perspective change here to make anything other than the spherical surface, but that alone is still an interesting result nonetheless.

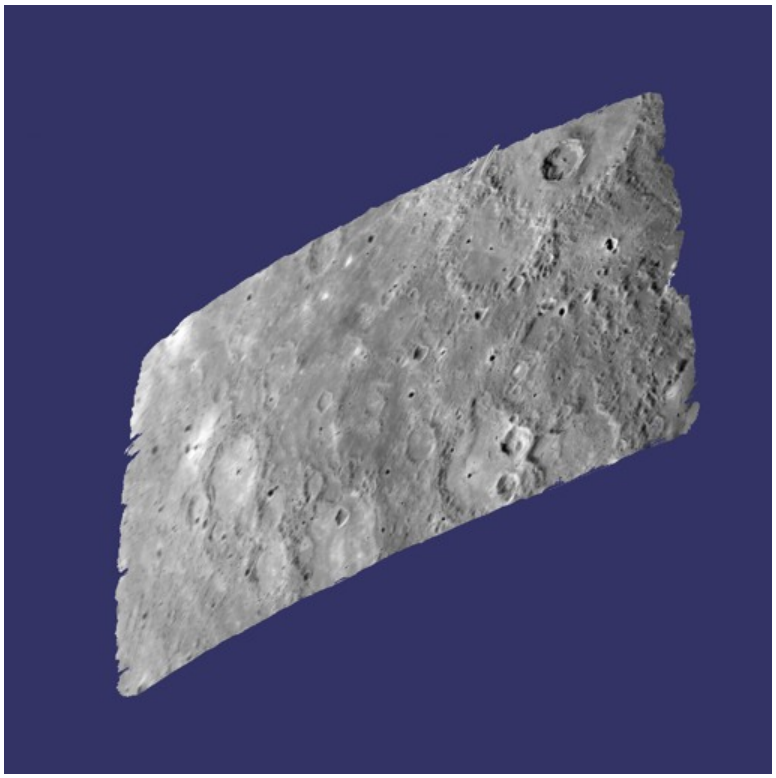


Figure 6.9: A rough attempt at stereo reconstruction from MDIS imagery.

Commands

```
ISIS 3> mdis2isis from=EW0108825359A.IMG to=EW0108825359A.cub
ISIS 3> mdis2isis from=EW0108825379C.IMG to=EW0108825379C.cub
ISIS 3> spiceinit from=EW0108825359A.cub
ISIS 3> spiceinit from=EW0108825359C.cub
ISIS 3> mkdir result
ISIS 3> stereo EW0108825359A.cub EW0108825379C.cub stereo/output
```

stereo.default

stereo.default for MDIS

```
### PREPROCESSING

DO_INTEREST POINT_ALIGNMENT 1
INTEREST POINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 0
DO_INDIVIDUAL_NORMALIZATION 1

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_BLUR 5
COST_MODE 0

H_KERNEL 25
V_KERNEL 25

H_CORR_MIN -10
H_CORR_MAX 10
V_CORR_MIN -2
V_CORR_MAX 2

SUBPIXEL_MODE 2

SUBPIXEL_H_KERNEL 19
SUBPIXEL_V_KERNEL 19

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

6.8 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.

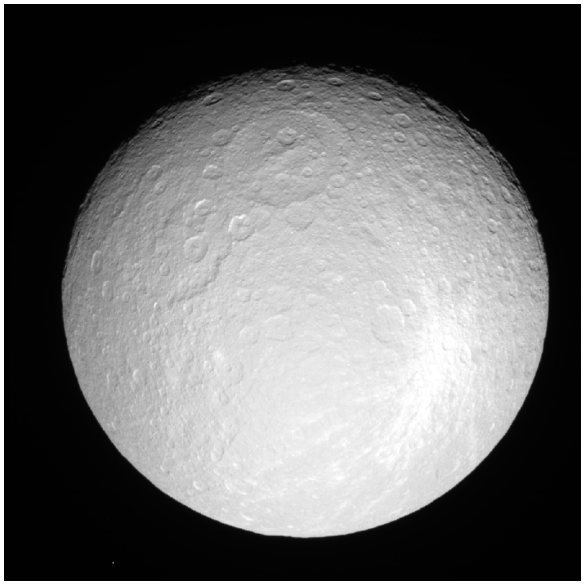
6.8.1 Rhea

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.

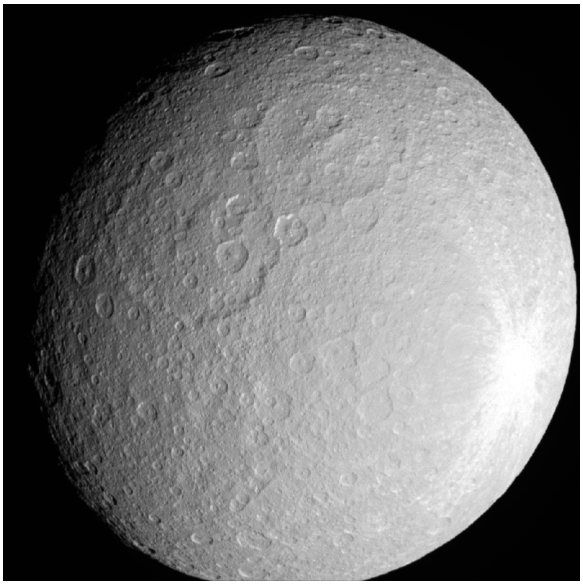
Commands

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

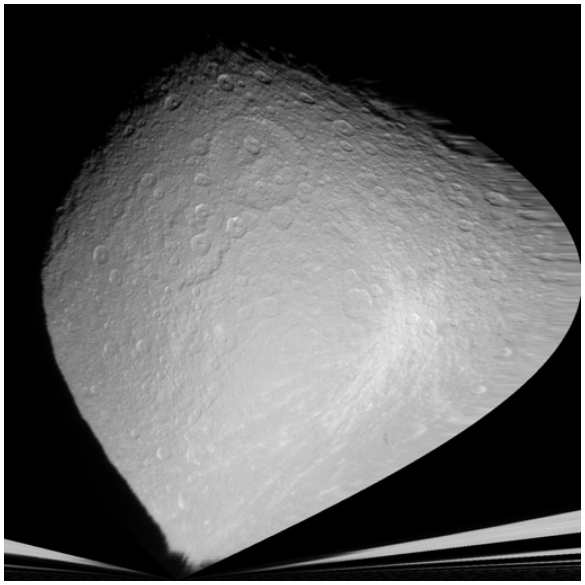
```
ISIS 3> ciss2isis f=N1511700120_1.LBL t=N1511700120_1.cub
ISIS 3> ciss2isis f=W1567133629_1.LBL t=W1567133629_1.cub
ISIS 3> cisscal from=N1511700120_1.cub to=N1511700120_1.lev1.cub
ISIS 3> cisscal from=W1567133629_1.cub to=W1567133629_1.lev1.cub
ISIS 3> fillgap from=W1567133629_1.lev1.cub to=W1567133629_1.fill.cub %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 fr= N1511700120_1.norm.cub
ISIS 3> spiceinit fr= 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> ls *.map.cub > fromlist
ISIS 3> ls N*.map.cub > holdlist
ISIS 3> equalizer fromlist=fromlist holdlist=holdlist
ISIS 3> mkdir result
ISIS 3> stereo N1511700120_1.map.equ.cub W1567133629_1.map.equ.cub result/rhea
```



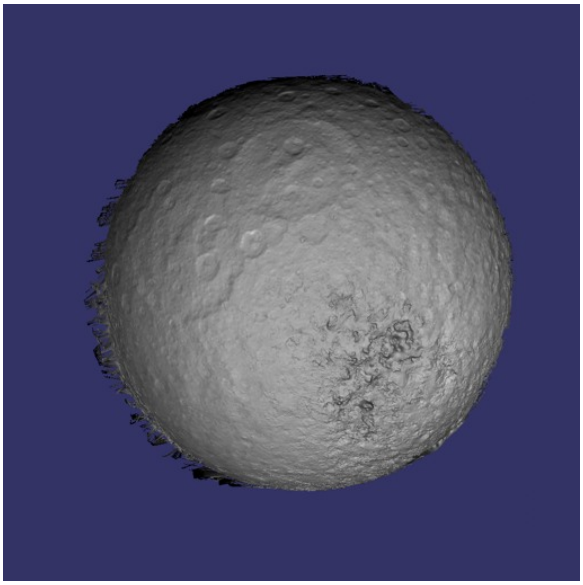
(a) `Original Left Image`



(b) `Original Right Image`



(c) `Map Projected Left`



(d) `3D Rendering`

Figure 6.10: Example output of what is possible with Cassini's ISS NAC

stereo.default

stereo.default for Cassini ISS

```
### PREPROCESSING

DO_INTERESTPOINT_ALIGNMENT 0
INTERESTPOINT_ALIGNMENT_SUBSAMPLING 0
DO_EPIPOLAR_ALIGNMENT 0

FORCE_USE_ENTIRE_RANGE 1
DO_INDIVIDUAL_NORMALIZATION 1

PREPROCESSING_FILTER_MODE 2

SLOG_KERNEL_WIDTH 1.5

### CORRELATION

COST_MODE 2
COST_BLUR 11

H_KERNEL 25
V_KERNEL 25

H_CORR_MIN -55
H_CORR_MAX -5
V_CORR_MIN -2
V_CORR_MAX 10

SUBPIXEL_MODE 3 # Experimental Subpixel Mode

SUBPIXEL_H_KERNEL 21
SUBPIXEL_V_KERNEL 21

### FILTERING

RM_H_HALF_KERN 5
RM_V_HALF_KERN 5
RM_MIN_MATCHES 60 # Units = percent
RM_THRESHOLD 3
RM_CLEANUP_PASSES 1

FILL_HOLES 1

### DOTCLOUD

NEAR_UNIVERSE_RADIUS 0.0
FAR_UNIVERSE_RADIUS 0.0
```

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 workhorse 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 3D model or DEM using the `point2mesh` or `point2dem` programs, respectively.

Usage:

`ISIS 3> stereo [options]` *`Left_input_image Right_input_image output_file_prefix`*

This release of the stereo pipeline has been specifically designed to process USGS ISIS `.cub` files. However, the 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 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/out` 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 this help information
`--threads` *`integer(=0)`*	Set the number threads to use. 0 means use default defined in the program or in the .vwrc file
`--session-type|-t pinhole|isis`	Select the stereo session type to use for processing. Usually the program can select this automatically for the file extension.
`--stereo-file|-s` *`filename(=./stereo.default)`*	Define the stereo.default file to use
`--entry-point|-e 1|2|3|4`	Pipeline entry point

`-draft-mode` *`debug-image-prefix`*	Cause the pyramid correlator to save out debug imagery named with this prefix
`--optimized-correlator`	Cause scale space search to not be performed and will hurt quality. This option is for software debugging.

More information about the stereo.default configuration file can be found in Appendix B on page 89. Similarly, `stereo` creates a lot of files, and they are all described in Appendix C on page 95.

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 handy 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 (Sub-pixel Refinement) performs sub-pixel correlation that refines the disparity map.

Stage 3 (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 4 (Triangulation) generates a 3D point cloud from the disparity map.

A.1.2 Decomposition of Stereo

Users watching their system closely will notice that the stereo executable is actually a python script that makes calls to seperate C++ executables for each entry point.

Stage 0 (Preprocessing) calls `stereo_pprc`. Multithreaded.

Stage 1 (Disparity Map Initialization) calls `stereo_corr`. Multithreaded.

Stage 2 (Sub-pixel Refinement) class `stereo_rfne`. Multithreaded.

Stage 3 (Outlier Rejection and Hole Filling) calls `stereo_fltr`. Multithreaded.

Stage 4 (Triangulation) calls `stereo_tri`. Single-Threaded.

All of the sub-programs have the same interface as `stereo`. Users processing a large number of stereo pairs on a cluster my find it advantageous to call these executables in their own manner. An example would be to run stages 0-3 in order for each stereo pair. Then run several sessions of `stereo_tri` since it is single threaded.

A.2 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.

Table A.2: Command-line options for disparitydebug

Options	Description
`--help|-h`	Display this help
`--input-file` *`filename`*	Explicitly specify the input file
`--output-prefix|-o` *`filename`*	specify the output file prefix
`--output-filetype|-t` *`type(=tif)`*	Specify the outfile type
`--float-pixels`	Save the resulting debug images as 32 bit floating point files (if supported by the selected file type)

A.3 point2dem

The `point2dem` program produces a GeoTIFF terrain model or an orthographic image from a point cloud image produced by the `stereo` command.

Example:

```
point2dem output-prefix-PC.tif -o stereo/filename -r moon \
    --default-value -10000 -n
```

This produces a digital elevation model that has been referenced to the lunar spheroid of 1737.4 km. Pixels with no data will be set to a value of -10000, and the resulting DEM will be saved in a simple cylindrical map projection. The resulting 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 an 8-bit GeoTIFF image in a simple cylindrical map projection.

Table A.3: Command-line options for point2dem

Options	Description
`--help|-h`	Display this table
`--default-value` *`float(=min-z)`*	Explicitly set the default missing pixel value. By default, the minimum z value in the model is used.
`--use-alpha`	Create images that have an alpha channel
`--dem-spacing|-s` *`float(=0)`*	Set the DEM post size (if this value is 0, the post spacing size is computed for you)

`--normalized|-n`	Also write a normalized version of the DEM (for debugging)
`--orthoimage` *texture-file*	Write an orthoimage based on the texture file given as an argument to this command line option
`--grayscale`	Use grayscale image processing for creating the orthoimage
`--offset-files`	Also write a pair of ASCII offset files (for debugging)
`--input-file` *pointcloud-file*	Explicitly specify the input file
`--texture-file` *texture-file*	Explicitly specify the texture file
`--output-prefix|-o` *output-prefix*	Specify the output prefix
`--output-filetype|-t` *type(=tif)*	Specify the output file type
`--reference-spheroid|-r moon|mars`	Set a reference surface to a hard coded value. This will override manually set datum information.
`--semi-major-axis` *float(=0)*	Set the dimensions of the datum in meters
`--semi-minor-axis` *float(=0)*	Set the dimensions of the datum in meters
`--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
`--sinusoidal`	Save using a sinusoidal projection
`--mercator`	Save using a Mercator projection
`--transverse-mercator`	Save using transverse Mercator projection
`--orthographic`	Save using an orthographic projection
`--stereographic`	Save using a stereographic 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)
`--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
`--cache-dir` *path(=/tmp)*	Sets directory to use for temporary files. Specify another directory if system is restrictive to large files in /tmp.

A.4 point2mesh

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. [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 and an optional texture file (*output-prefix*`-PC.tif` and normally *output-prefix*`-L.tif`). 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*`.ive` file that contains the 3D model in OpenSceneGraph format.

[1] The full OpenSceneGraph package is not bundled with the Stereo Pipeline, but the `osgviewer` program is. You can download and install this package separately from http://www.openscenegraph.org/.

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.

Example:

```
point2mesh -l -s 2 output-prefix-PC.tif output-prefix-L.tif
```

To view the resulting *output-prefix*`.ive` file use `osgviewer`.

Fullscreen:

```
> osgviewer output-prefix.ive
```

or Windowed:

```
> osgviewer output-prefix.ive --window 50 50 1000 1000
```

Inside `osgviewer`, the keys L, T, and W can be used to toggle on and off lighting, texture, and wireframe modes. The left, middle, and right mouse buttons control rotation, panning, and zooming of the model.

Table A.4: Command-line options for point2mesh

Options	Description
`--help|-h`	Display this help
`--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 mesher.
`--input-file` *pointcloud-file*	Explicitly specify the input file
`--texture-file` *texture-file*	Explicitly specify the texture file
`--output-prefix|-o` *output-prefix*	Specify the output prefix
`--output-filetype|-t` *type(=ive)*	Specify the output file type
`--enable-lighting|-l`	Enables shades and light on the mesh
`--center`	Center the model around the origin. Use this option if you are experiencing numerical precision issues.
`--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

A.5 orbitviz

Produces a Google Earth KML file useful for visualizing camera position. The input for this tool is one or more `*.cub` files.

Table A.5: Command-line options for orbitviz

Options	Description
`--help|-h`	Display this help
`--output|-o` *`filename(=orbit.kml)`*	Specifies the output file name
`--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.*

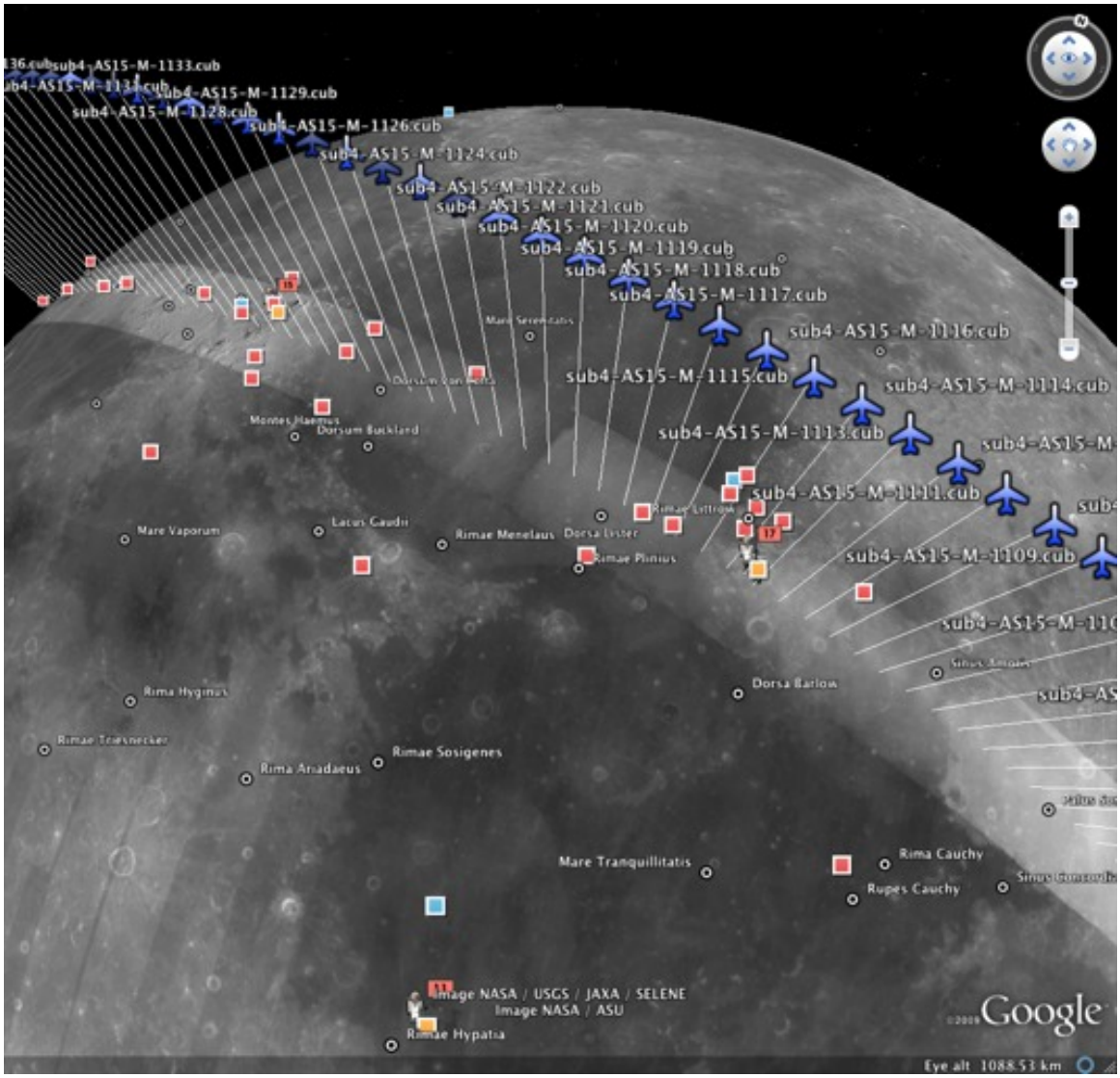


Figure A.1: Example of a KML visualization produced with `orbitviz` depicting camera locations for the Apollo 15 Metric Camera during orbit 33 of the Apollo command module.

A.6 isis_adjust

Bundle Adjustment for ISIS images. This tool supports adjustment of linescan cameras as well as simple frame cameras. For an in depth view into how to use this tool, please read Chapter 5.

Table A.6: Command-line options for isis_adjust

Options	Description
`--help|-h`	Display this help
`--cnet|-c` *`control-network-file`*	Load a control network from a file
`--cost-function` *`function(=L2)`*	Choose a robust cost function from `[L1|L2|Cauchy|Huber|PseudoHuberL1|L2|Cauchy|Huber|PseudoHuber]`
`--bundle-adjuster` *`adjuster(=Sparse)`*	Choose a bundle adjustment version from `[Ref|Sparse|RobustRef|RobustSparse|RobustSparseKGCP]`
`--disable-camera-const`	Disable the camera constraint error. This allows the cameras to move to pretty much anywhere.
`--disable-gcp-const`	Disable the GCP constraint error.
`--gcp-scalar` *`multiplier(=1)`*	Sets a scalar to multiply to the sigmas (uncertainty) defined for the gcps. GCP sigmas are defined in the .gcp files.
`--lamda|-l` *`float`*	Set the initial value of the LM paramater `g_lambda`. If not set the algorithm will find the optimum starting point.
`--min-matches` *`integer(=30)`*	Set the minimum number of matches between images that will be considered.
`--max-iterations` *`integer(=25)`*	Set the maximum number of iterations
`--poly-order` *`integer(=0)`*	Set the order of the polynomial used to adjust the camera properties. If using a frame camera, leave at 0 (meaning scalar offsets). For line scan cameras try 2.
`--position-sigma` *`float(=100)`*	Set the sigma (uncertainty) of the spacecraft position. (meters)
`--pose-sigma` *`float(=0.1)`*	Set the sigma (uncertainty) of the spacecraft pose. (radians)
`--report-level|-r` *`integer(=10)`*	Changes the detail of the Bundle Adjustment Report. Valid values are 0 to 100
`--robust-threshold` *`float(=10)`*	Set the threshold for robust cost functions.
`--save-iteration-data|-s`	Saves all camera/point/pixel information between iterations for later viewing in `bundlevis`
`--seed-with-previous`	Use previous `isis_adjust` files at starting for this run
`--write-isis-cnet-also`	Writes an ISIS style control network
`--write-kml [0|1(=0)]`	Selecting this will cause a KML file to be written with the GCPs. Set this flag to 1 and it will also write all the 3D position estimates of the points it is tracking in the KML.

A.7 bundlevis

The `bundlevis` program is a bundle adjustment result visualizer. See Chapter 5 for more information.

Table A.7: Command-line options for bundlevis

Options	Description
`--help|-h`	Display this help
`--camera-iteration-file|-c` *filename*	Load the camera parameters for each iteration from this file
`--points-iteration-file|-p` *filename*	Load the 3D points parameters for each iteration from this file
`--pixel-iteration-file|-x` *filename*	Load pixel information data. Allowing for an illustration of the pixel data over time
`--control-network-file|-n` *filename*	Load a control network for point and camera relationship status
`--additional-pnt-files` *filename(s)*	Plot additional point files simultaneously with the above data
`--fullscreen`	Render with the entire screen
`--stereo`	Render in anagylph mode
`--show-moon`	Draw a wireframe moon
`--show-mars`	Draw a wireframe mars
`--show-earth`	Draw a wireframe earth

A.8 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 disucssion on aligning images on page 18.

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.8: Command-line options for cam2map4stereo.py

Options	Description
`--help|-h`	Display this help
`--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.
`--suffix|-s`	Suffix that gets inserted in the output file names, defaults to 'map'.

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.

Below we will walk through the contents of the `stereo.default` and discuss its various parameters. If you want to start with a clean slate, you can copy the `stereo.default.example` file that is included in the top level of the Stereo Pipeline software distribution.

Note: The parameters that begin with '`DO_*`' are true/false options, when set to '1' they are 'on' or 'true,' and if set to '0' they are 'off' or 'false.' All parameters below have their default values listed after the parameter name.

B.1 Preprocessing

CACHE_DIR (default = /tmp)
Place for to store intermediate files for when Stereo Pipeline can't store entire image in memory. User should change directory if operating system has limit on filesize inside the `/tmp` directory.

DO_INTERESTPOINT_ALIGNMENT (default = 0)
When `DO_INTERESTPOINT_ALIGNMENT` is set to 1, `stereo` will attempt to pre-align the images by automatically detecting tie-points between images using a feature based image matching technique. Tiepoints are stored in a `*.match` file that is used to compute a linear affine 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.

INTERESTPOINT_ALIGNMENT_SUBSAMPLING (= 1,2,3,...,N) **(default = 1)**
This settings is only for the "keypoint" stereo session. It is not used for the "isis" stereo session.

Use this option to subsample images before interest point alignment when DO_INTERESTPOINT_ALIGNMENT is activated. This can significantly speed up the interest point alignment step at the expense of alignment accuracy. When this is set to 1, there is no subsampling, and the `stereo` program will do its best to find as many interest points within the imagery as it can. When this is set to $N > 1$, the program will subsample the images by a factor of N before detecting interest points. This parameter can be set to any positive integer.

DO_EPIPOLAR_ALIGNMENT (default = 0)
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.*

This method uses the inherent underlining geometry of the cameras to create a rectified version of the images where stereo disparity occurs only in the horizontal direction.

FORCE_USE_ENTIRE_RANGE (= 0,1) **(default = 0)**
This setting is only for ISIS stereo session.

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 in the stereo pipeline 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.

DO_INDIVIDUAL_NORMALIZATION (**default = 0**)
This setting is only for ISIS stereo session.

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.

PREPROCESSING_FILTER_MODE (= 0,1,2,3) **(default = 3)**
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 - Gaussian Blur - Pre-blur images using a Gaussian kernel. This option can improve correlation results by blurring out small-scale image noise.

2 - LoG Filter - Same as above, but take the Laplacian of the result. This provides some immunity to differences in lighting conditions between a pair of images by isolating and matching on edge features in the image.

3 - Signed LoG Filter - Same as above, but retain only the sign of the Laplacian image (+1 or -1). This option provides the best immunity to variations in lighting conditions between images.

For modes 1, 2, and 3 above, the size of the Gaussian blur is determined by the SLOG_KERNEL_WIDTH variable 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. Blurred preprocessing can sometimes produce the best results with well-calibrated images when working with the normalized cross correlation cost function.

SLOG_KERNEL_WIDTH (= *float*) **(default = 1.5)**
This defines the diameter of the Gaussian convolution kernel used for the preprocessing modes 1, 2, and 3 above. A value of 1.5 works well in a variety of applications.

B.2 Correlation

COST_MODE (= 0,1,2) (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.

0 - absolute difference

1 - squared difference

2 - normalized cross correlation

COST_BLUR (= *integer* $N >= 0$) **(default = 0)**
Reduces the number of missing pixels by blurring the fitness landscape computed by the cost function by an $N \times N$ box filter. Increases the number of stereo matches during initialization at the expense of overall accuracy. **Cost blurring must be used in conjunction with affine adaptive window subpixel modes below, which are capable of achieving highly accurate results even when seeded by slightly inaccurate matches from the initialization step.**

H_KERNEL (= *integer*) **(default = 25)**

V_KERNEL (= *integer*) **(default = 25)**
These two items determine the size (in pixels) of the correlation kernel used in the initialization step. A different size can be set in the horizontal *(H)* and vertical *(V)* directions, but square correlation kernels are almost always used in practice.

H_CORR_MIN (= *integer*)

H_CORR_MAX (= *integer*)

V_CORR_MIN (= *integer*)

V_CORR_MAX (= *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.

Note: Commenting out these settings will cause `stereo` to make an attempt to guess its search range using interest points.

SUBPIXEL_MODE (= 0,1,2,3) **(default = 2)**
This parameter selects the subpixel correlation method. These algorithms are arranged in order of decreasing speed and increasing quality. 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.

0 - no subpixel refinement

1 - parabola fitting

2 - affine adaptive window, Bayes EM weighting

3 - affine adaptive window, Bayes EM with Gamma Noise Distribution (experimental)

For a visual comparison of the quality of these subpixel modes, refer back to Chapter:4.

SUBPIXEL_H_KERNEL (= *integer*) **(default = 35)**

SUBPIXEL_V_KERNEL (= *integer*) **(default = 35)**
Specify the size of the horizontal *(H)* and vertical *(V)* size (in pixels) of the subpixel correlation kernel.

B.3 Filtering

RM_H_HALF_KERN (= *integer*) **(default = 5)**

RM_V_HALF_KERN (= *integer*) **(default = 5)**
Taken together, the RM_* settings adjust 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 RM_H_HALF_KERN and RM_V_HALF_KERN 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.

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.

RM_THRESHOLD (= *integer*) **(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.

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.

FILL_HOLES (= 0,1) **(default = 1)**
When this option is on, the holes in the disparity map that result from poor stereo matching will be filled by an inpainting algorithm. Inpainting is a convolution method that takes the values at the edges of holes and spreads those values inward. This method performs best for small holes.

Note: you can always use the good pixel mask image (`*-GoodPixelMap.TIF`) to determine which pixels represent "real" data matched by the stereo correlator, and which pixels represent interpolated data produced by inpainting.

FILL_HOLE_MAX_SIZE (= *integer*) **(default = 100,000)**
This defines the maximum size of a hole that the inpainting technique should attempt. Default is 100,000 pixels.

B.4 Post-Processing

NEAR_UNIVERSE_RADIUS (= *float*) **(default = 0.0)**

FAR_UNIVERSE_RADIUS (= *float*) **(default = 0.0)**

These parameters can be used to filter out triangulated points in the 3D point cloud by setting an near and far radius value from origin of the point cloud's coordinate system, [0,0,0]. For most ISIS cameras, the origin is the center of the body (e.g. the Moon or Mars), and is part of a body-fixed Cartesian coordinate system that rotates with the planet.

These settings are most useful for other stereo session types (e.g. pinhole), where the origin of the coordinate system is often one of the cameras in a stereo pair. In this case, these parameters can be used to reject pixels that are too close or too far from the camera system.

Setting both values zero turns off this restriction and allows the dot cloud to be as big as the data allows for.

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 `DO_INTERESTPOINT_ALIGNMENT` is enabled, 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.

***-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.exr** - pre-alignment matrix

The 3×3 affine transformation matrix that was used to warp the right image to roughly align with the left image. This file is only generated if `DO_INTERESTPOINT_ALIGNMENT` is enabled in the `stereo.default` file.

***-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 `DO_INTERESTPOINT_ALIGNMENT` is on. 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 the Stereo Pipeline. It contains 3D locations for each valid pixel; stored as a 64-bit, 3-channel TIFF, with coordinates in a body-fixed planetocentric coordinate system. Each pixel in the point cloud image corresponds to a pixel in the left input image.

Note: it is unlikely that your usual TIFF viewing programs will visualize this file properly. This file should be considered a 'data' file, not an 'image' file. Other programs in the Stereo Pipeline, such as `point2mesh` and `point2dem` will convert the contents of this file to more easily visualized formats.

***-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

Modifying SURF to output VW match files

SURF *v1.0.9* is a fast a relatively robust interest point algorithm. It is not open source, but it is freely available for academic uses at http://www.vision.ee.ethz.ch/~surf/. This software is currently only available for Windows and Linux 32 bit.

SURF creates it own results files. What is available online was probably only meant for demonstrations. What we've done is created a patch that allows the SURF match utility, `match.ln`, to create Vision Workbench match files. The patch is available as the `surf_match.patch` in the `examples/` directory of the Stereo Pipeline distribution.

D.1 How to apply and compile

First move to the directory containing your copy of the SURF v1.0.9 code. Then copy `surf_match.patch` to the active directory. At this point you are ready to start running the following commands.

```
> patch < surf_match.patch
> make match.ln
```

> *Note:*
> If you are unfortunate enough to run into an error such as *g++-4.0.2: Command not found*, don't worry. Edit `Makefile` at line 10 and 11 to refer to `g++` instead of `g++-4.0.2`.
>
> Also since you've incurred that error, you'll probably need to add an include to `<stdlib.h>` in `imload.cpp` in the same directory. This all stems from differences in using a newer version of g++.

D.2 Example of using SURF

For this example it is assumed you have a directory containing two images named `m1000254.png` and `r0901059.png` like in the example found in Section 5.3.

SURF code only works with images in the grayscale format PGM. A free Linux utility to convert the images is `mogrify`. That utility is part of the package `ImageMagick` and is likely to be available in most package managers.

Below are the commands to take an input of PNG files, process them with SURF, and then finally create a match file which can be used by `isis_adjust`.

```
> mogrify -format pgm m1000254.png r0901059.png
> surf.ln -i m1000254.pgm -o m1000254.surf
> surf.ln -i r0901059.pgm -o r0901059.surf
> match.ln -k1 m1000254.surf -k2 r0901059.surf \
         -im1 m1000254.pgm -im2 r0901059.pgm \
         -o out.pgm -m m1000254__r0901059.match
> rm m1000254.pgm r0901059.pgm *.surf
```

> It is important to note that though SURF is very good at performing matches it does not perform a step of RANSAC with its output. There may be a couple of outliers.

Bibliography

[1] 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.

[2] J.A. Anderson. ISIS Camera Model Design. In *Proc of the Lunar and Planetary Science Conference (LPSC) XXXIX*, page 2159, March 2008.

[3] Simon Baker, Ralph Gross, and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56:221–255, 2004.

[4] 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/.

[5] 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.

[6] The Open Scene Graph Community. The open scene graph website. 2009. URL http://www.openscenegraph.org/projects/osg.

[7] 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.

[8] 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.

[9] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[10] 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.

[11] 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.

[12] 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.

[13] Zach Moore, Dan Wright, Chris Lewis, and Dale Schinstock. Comparison of bundle adjustment formulations. In *ASPRS Annual Conf., Baltimore, Maryland*, 2009.

[14] 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.

[15] H.K. Nishihara. PRISM: A Practical real-time imaging stereo matcher. *Optical Engineering*, 23(5): 536–545, 1984.

[16] 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.

[17] Changming Sun. Rectangular Subregioning and 3-D Maximum-Surface Techniques for Fast Stereo Matching. *International Journal of Computer Vision*, 47(1-3), 2002.

[18] Richard Szeliski and Daniel Scharstein. Sampling the Disparity Space Image. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26:419 – 425, 2003.

[19] 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.

[20] Tuscon University of Arizona. The high resolution imaging science experiment. 2009. URL http://hirise.lpl.arizona.edu/.

[21] AZ U.S. Geological Survey, Flagstaff. Integrated software for imagers and spectrometers (ISIS). 2009. URL http://isis.astrogeology.usgs.gov/.