Multi-View Stereo Documentation

Release 1

Jean-Paul Chièze

August 23, 2013

CONTENTS

CHAPTER

ONE

USER GUIDE

1.1 Overview

Our goal is : given a set of photographs of a scene, taken at different viewpoints, make a 3D representation of the scene. The method is based on PMVS2 (Patch_Based Multi_View Stereopsis, http://www.di.ens.fr/pmvs/) by *Furukawa & Ponce*.

PMVS2 itself needs some pre-processing, so software includes the source of PMVS2 and of a certain number of other external programs.

The main steps to achieve 3D recontruction are:

- compute features in the images (for example Sift keys). Several programs proposed :
  - vlfeat-sift (VisionLab Features Library, http://www.vlfeat.org).
  - siftfeat a program by Rob Hess, based on opencv (http://blogs.oregonstate.edu/hess/code/sift/).
  - a personnal program using feature extractors of OpenCV library version 2.4.
- find matching points between images : program KeyMatchFull from bundler-v0.4 (http://phototour.cs.washington.edu/bundler/), with modifications to add parallelism.
- use stereo information to compute camera 3D parameters: program bundler from bundler-v0.4 by Noah Snavely (http://phototour.cs.washington.edu/bundler/) with, in some circumstances, parallel extension (http://grail.cs.washington.edu/projects/mcba/)
- divide the image set in clusters with CMVS from Yasutaka Furukawa, to speed up next step (http://www.di.ens.fr/cmvs/).
- **produce a representation of the scene as a dense set of surface patches** with PMVS2 from Yasutaka Furukawa and Jean Ponce (http://www.di.ens.fr/pmvs/)

Note: source code for CMVS and PMVS2 is a version modified by Pierre Moulon, mainly for portability reasons (TheFrenchLeaf-CMVS-PMVS-b4185d1, https://github.com/TheFrenchLeaf/CMVS-PMVS).

We also propose a program, based on module Delaunay-3d of CGAL-4.1 (Computational Geometry Algorithms Library, http://www.cgal.org), to make a facet representation of the surface.

The python script **mvs** will control the whole process from input images to dense points.

The script **add** allows one to add images, but only at the very first steps of the process.

The surface construction involves two separate programs (see below).

1.2 Licenses:

All the components of this software that we developped are under the GPL licence This software also includes third parties programs. They are all under GPL or at least open source. Refer to the license information for each of them.

The GPL (v3+, http://www.gnu.org/copyleft/gpl.html) is an Open Source license that, gives you the right to use, copy and modify the code freely. If you distribute your software based on our software, you are obliged to distribute the modifications you made, and you are furthermore obliged to distribute the source code of your own software under the GPL.

As a modified version of *do_intersect* from `include/CGAL/Triangle_3_Segment_3_do_intersect.h` is included in the triangulation part, the CGAL licenses are included in the distribution (see `CGAL/LICENSES/`).

1.3 Installation

On a linux system :

- extract the source from the tar file:

  tar zxf inria-mvs-v1.3-svnjj-mm-yyyy.src.tgz

- enter inria-mvs
- if BLAS, LAPACK or OpenCV are not in standard locations, or if you have installed CGAL (4.1), create a file *local-config.cmake* (use *local-config.cmake.example* as example)
- create a build directory, for example Linux
- enter this directory and run:

```
cmake ..
make
make install
```

- everything should be installed in `./Linux/install{bin,lib,doc,tools}`.

1.4 Quick start

You must create a working directory and provide a set of overlapping jpeg images. You can give the directory containing the images or a file containing the paths to images.

Important note: the jpeg files must contain EXIF info, specially with fields 'focalLength', 'Make' and 'Model'.

If $bin is the directory corresponding to Linux/install/bin, just run:

```
$bin/mvs -i image-dir -w work-dir
```

You will get .ply files in work-dir/pmvs/models. They can be viewed, for example, with **meshlab**.

If you are satisfied of the result, be happy, otherwise read more of this documentation and experiment with the software to increase your expertise level...

If you get a message `WARNING! Cameras missing in ccdwidth database .....` you need to add the sensor width (in mm) of your camera to the database by running the command given in the message, for example:

```
$bin/ccdwidths -c "Canon/Canon EOS 50D" -w 22.71
```

A good place to find the sensor width is http://www.dpreview.com.

You can download `examples.tgz` for a first try of the software.

1.5 Hints about images:

The result is very dependant on how the photographs are taken. Remember that the program has to find corresponding points between images, that means detect "remarkable" points in each image and find enough similar ones between images.

- Use the same camera with the same focal for all photographs of the scene.
- Move the camera around the scene with rotation or translation steps

ensuring that matched points be visible in at leat 3 images. - Parts of uniform color or texture will not give enough good points. - Moving elements may mess up the reconstruction. - Avoid heterogeneous viewpoints : several pictures should be taken at a similar distance. - Variable lighting conditions could make the matching harder. - If you have hight resolution pictures (> 2Mpixels), you should do points detection on a reduced size (see option **-maxsize** below).

1.6 Script mvs

This script does all the job in 8 steps that can be executed separately :

1. extract image features. Various algorithms can be used for feature extraction, such as Sift, SURF, Harris, but Sift seems to give the best results.
2. feature matching (program `p-KeyMatchFull`) : find feature points common to each pair of images

1. make option file for bundler
2. run bundler : compute cameras position and orientation.
3. prepare data for CMVS
4. exec CMVS
5. generate options for PMVS2
6. run PMVS2 program.

1.6.1 Arguments:

mvs [**-v**] [**-n**] [**-cpus** *nb_cpus*] [**-i** *input*] [**-w** *workdir*] [**-s** *steps-range*] [**-pre-process** *samples-list*] [**-maxsize** *n*] [**-shrink** *n*] [**-ccdwidth** *def_ccdwidth*] [**-cv** *siftfeat-options*] [**-vl** *vlsift-options*] [**-ocv** *opencv-keytype*] [**-match** *nbim*] [**-ff**] [**-pba**] [**-bdlpair** *init_bundler_pair*] [**-nocmvs**] [**-maxnbim** *nbim*] [**-pmvs** *pmvs-num*] [**-ssh** *hosts-file*] [**-csize** *n*] [**-minvis** *nb_visible_imgs*]

3D scene reconstruction from a set of images. Input images are processed to produce a 3D representation of the scene as a cloud of points in PLY format.

- **-v**: verbose mode
- **-n**: fake execution
- **-cpus** nb_cpus: Nb of CPUs to use (default = all)

- **-i** input: image dir or file of image paths (only used by step 1).
- **-w** workdir: working dir
- **-s** steps-range: steps : n or n1-n2, where step number range from 1 to 8: 1: extractFeatures, 2: matchKeys, 3: makeOptions, 4: execBundler, 5: bundle2PMVS, 6: execCmvs, 7: execGenOptions, 8: execPmvs.

  Option -s 0 with -pre-process means 'do only pre-processing'.
- **-pre-process** samples-list: use a few images to estimate distorsion, and apply correction to images
- **-maxsize** n: max image size for feature extraction (use a shrinked copy if necessary)
- **-shrink** n: max input image size : do all processing on resized images.
- **-ccdwidth** def_ccdwidth: ccdwidth for images without camera model info
- **-cv** siftfeat-options: Options for siftfeat as "std" of "def" or string of options (std = ""; def = "-d -s 1.6 -c 0 -r 3").
- **-vl** vlsift-options: Options for vlSift as "std" or "def" (default) or string of options (std = ""; def = "–first-octave 0 –peak-thresh 0 –edge-thresh 3.5")
- **-ocv** opencv-keytype: Use points detector/extractor from opencv (['surf', 'sift', 'hsurf', 'ssift', 'osift', 'ssurf', 'hsift', 'orb', 'ofreak', 'osurf'])
- **-match** nbim: max preceding images to check against current in KeyMatchFull (all images by default).
- **-ff**: fixed focal length and no distorsion estimation
- **-pba**: Use parallel version of bundler (warning : you should first apply a distorsion correction to images, with **-pre-process**, for example)
- **-bdlpair** init_bundler_pair: Force bundler tu use this image pair for initialization.
- **-nocmvs**: Don't clusterize images with cmvs
- **-maxnbim** nbim: max nb of images per cluster (for cmvs)
- **-pmvs** pmvs-num: option file number for pmvs (0 for option-0000)
- **-ssh** hosts-file: hostnames for execution of PMVS on cmvs clusters
- **-csize** n: csize option for pmvs2 (2 or 1)
- **-minvis** nb_visible_imgs: (pmvs2) each 3D point must be visible in at least nb_visible_imgs images to be reconstructed; default 3

Notes:

1. If run without option -s, the program will skip steps already done by a previous add command
2. Addition of images made by a previous command add will become definitive (except if you force start from step 1)
3. image list is saved in <workdir>/orig-list.txt.
4. Default behaviour is equivalent to -cv std -s 1-8 -csize 2 -minvis 3

Input data and working directory can be specified in different ways:

- if argument of **-i** is a file, it must contain paths to the images and option **-w** is mandatory to specify the working directory.
- if argument of **-i** is a directory with a subdirectory named `photos`, option **-w** if not allowed; images must be in `photos` and a working directory named `result` will be created in the input directory.

- if argument of **-i** is a directory without `photos`, it must contain the images and option **-w** is mandatory.

A file named `history` will be created in the working directory to record the different steps of the program. It is overwritten at execution of step~1. It contains arguments, start and end times of each run, and elapsed time of each step.

1.6.2 Detailed description:

The different steps of the process can be executed separately with option **-s**. The argument can be a step number (1 .. 8) or a range (n1-n2). Of course a step cannot be executed before the previous one.

Pre-process: This special step is used to speed up bundler, but it should only be applied to images taken with the same camera and the same focal. It is invoked by the **-pre-process** option. The argument is a file containing paths to a few images. The program will estimate distortion coefficients and apply the same distortion correction to all images, so that the parallel version of bundler can be used on the result. To check the validity of this method, you can first run steps 1-4 on the sample list and use **check-bundler** (see doc) to verify that estimated coefficients of cameras are close enough:

```
$y/mvs -i sample-list -w $workdir -s 1-4
$y/tool check-bundle -coefs $workdir/bundle/bundle.out
```

Step 1: Do feature extraction on images. Options **-i** is mandatory. If **-pre-process** is given, the images will first be undistorted. A file `orig-list.txt` containing the absolute images paths is created in the working directory, so that option **-i** will be ignored in other steps.

Images are copied, eventually modified (options **-pre-process** or **-shrink**), to the working directory with names of the form num.jpg, where num is a number written on 8 digits startint at zero.

Features are written to files with same basename and extension `.key.gz` .

- **-maxsize** *s* : feature extraction will be done on reduced images (max dimension shrinked to *s*). 1000 to 1500 should be a good value for *s*. On large images, feature extraction will produce too many points and take a too long time.
- **-shrink** *s* [shrink images before starting. All the process will] be applied to these modified images. Will probably not give results as good as **-maxsize**, but may be useful for quick tests.
- **-ccdwidth** *w* : supply a default ccdwidth for images without camera info.
- **-cv** "*options*" : use program `cv-sift/siftfeat` for feature extraction.

  options is either "std" or "def" or a space separated list of `siftfeat` arguments (see *$bin/cv-sift/siftfeat -h*).

  "def" = "-d -s 1.6 -c 0 -r 3"

  "std" = no options (use built-in values)
- **-vl** "*options*" : use program `vlfeat/sift`. *options* is either "std" or "def" or a space separated list of arguments (see *$bin/vlfeat/sift -h*).

  "def" = "–first-octave 0 –peak-thresh 0 –edge-thresh 3.5"

  "std" = no options (use built-in values)
- **-ocv** *keytype* : use program `cvkeys` which provides various combination of feature detectors/extractors from the OpenCV lib, version 2.4. The recommended value for *keytype* is *sift*

If no feature extractor is specified, **-cv** *std* is used.

Note: feature extraction is done in parallel and may use a lot of memory. You may specify a number of cpu's lower than the maximum (option **-cpus**).

Step 2: For each pair of images, find matching keypoints. The program is parallelized (with OPENMP), but the execution time is in N*N.

- **-match** *nbim* : only try to match each image with the *nbim* previous ones. This will considerably speed-up the program, but it supposes that images are almost correctly ordered.

The matching information is written to `matches.init.txt`.

Step 3: Just build `options.txt`, the options file for **bundler**

- **-ff** : set option preventing bundler from calculating distortion and estimating real focal length.
- **-bdlpair** : force bundler tu use the given pair as initial one.

Step 4: Run **bundler**. Uses `matches.init.txt` and keypoints files to produce `bundle/bundle.out`

- **-pba** : use parallel version of bundler. The distortion coefficient are only partially estimated. Should be used with undistorted images (see abaove **-pre-process**), or the final result of PMVS2 will be bad.

Step 5: Prepare data for CMVS and PMVS2 (program **Bundle2PMVS** from bundler package). Images without keypoint match will be ignored.

- make a copy without ignored images of `bundle/bundle.init.out` to `pmvs/bundle.rd.out`
- write projection matrices of valid cameras to `.txt` files in `pmvs/txt`
- write the list of valid images to `pmvs/list.rd.txt`
- use coefficients in `bundle/bundle.out` to undistort valid images and copy them to `pmvs/visualize`.
- write image neighborhood information to `pmvs/vis.dat` for use by CMVS.

Step 6: Run **cmvs** to find cluster of images. Each cluster will be processed independantly by PMVS2

- **-nocmvs**: don't run cmvs!
- **-maxnbim** *n* : define maximum number of images allowed in a cluster (default 100)

Step 7: Create option files for PMVS2. Files are named *option-nnnn*, where *nnnn* is the CMVS cluster number on 4 digits, starting at 0. If **-nocmvs** is specified, *option-nocmvs* will be created instead.

- **-nocmvs** : The option file will be `option-nocmvs`.
- **-csize** *n* : set option *csize* of **pmvs2** (default 2)
- **-minvis *n*** [set option *minImageNum* of **pmvs2** (default 3);] It is the minimum number of images a patch must be visible in.

Step 8: Run **pmvs2**. From option files and data in `pmvs/txt` and `pmvs/visualize`, build dense 3D patches and store the result in directory `pmvs/models` (files *option-nnnn* with extensions *.ply*, *.patch*, *.pset*). Program pmvs2 uses pthreads library. Option files (`option-nnnn`) are processed sequentially.

- **-pmvs** *n* : execute pmvs2 only for option file `option-000n`
- **-ssh** *hosts-file* : use hosts given in *hosts-file* to dispatch execution of pmvs for all availeble option files. The localhost may be specified as *localhost* or by its hostname. Only the first definition of localhost will be used.

1.7 Script add

Allows to add images. Does an incremental update on steps 1 and 2. Other steps must be re-run with all images.

1.7.1 Arguments:

add [**-v**] [**-n**] [**-cpus** *nb_cpus*] [**-i** *input*] **-w** *workdir* [**-s** *steps-range*] [**-undo**] [**-valid**] [**-maxsize** *n*] [**-shrink** *n*] [**-cv** *siftfeat-options*] [**-vl** *vlsift-options*] [**-ocv** *opencv-keytype*] [**-ccdwidth** *def_ccdwidth*] [**-match** *nbim*]

Add images to an already pocessed sequence. The programm can only execute the first 2 steps of the whole process. Steps 3 to 8 must be executed by the normal command ('mvs').

- **-v**: verbose mode
- **-n**: fake execution
- **-cpus** nb_cpus: Nb of CPUs to use (default = all)
- **-i** input: image dir or file of image paths
- **-w** workdir: working dir
- **-s** steps-range: steps : n or n1-n2, where step number range from 1 to 2: 1: extractFeatures, 2: matchKeys
- **-undo**: go back to the original state (preceding add). Needs only -w.
- **-valid**: Make the last additions definitive
- **-maxsize** n: max image size for sift
- **-shrink** n: max input image size (resize before processing)
- **-cv** siftfeat-options: Options for siftfeat as "std" of "def" (default) or string of options (std = ""; def = "-d -s 1.6 -c 0 -r 3").
- **-vl** vlsift-options: Options for vlSift as "std" or "def" (default) or string of options (std = ""; def = "–first-octave 0 –peak-thresh 0 –edge-thresh 3.5")
- **-ocv** opencv-keytype: Use points detector/extractor from opencv (['surf', 'sift', 'hsurf', 'ssift', 'osift', 'ssurf', 'hsift', 'orb', 'ofreak', 'osurf'])
- **-ccdwidth** def_ccdwidth: ccdwidth for images without camera model info
- **-match** nbim: max preceding images to check against current in KeyMatchFull (all images by default).

NB: The modifications can be removed with -undo as long as you don't execute the commande 'mvs'

The program first makes a backup copy of files `list.txt, list_features.txt, matches.init.txt`.

See description of steps 1 and 2 in documentation of **mvs**.

Images and key files are added in the working directory; new values are added to `matches.init.txt`. But the additions are temporary and overwritten by each execution of the script, until option **-valid** is given. On the contrary, option **-undo** will revert to the original state. If one of these 2 options is given, no step is executed. Example:

```
$bin/mvs -i src -w work
$bin/add -i add-dir -w work
$bin/add  -w work -valid
$bin/mvs -s 3-8``
```

1.8 Script ccdwiths

ccdwidths [**-c** *camera-name*] [**-i** *ref-image*] [**-w** *ccdwith*]

Without arguments, list contents of camera database extensions, else add the given ccdwith to the camera database extension file

- **-c** camera-name: camera identifier "<Camera make>/<Camera model>" as given by exif image metadata
- **-i** ref-image: image from which to extract the camera identifier.

Only one of -c -i is allowed

- **-w** ccdwith: camera ccd width in mm.

Example:

```
$bin/ccdwidths -c "Canon/Canon EOS 50D" -w 22.71
```

1.9 Examples

- Simple example:

  ```
  $bindir/mvs -i $images -w $workdir
  ```

Result in `$workdir/pmvs/models/option-0000.ply`

- add some new images and recompute:

  ```
  $bindir/add -i $newimages -w $workdir
  $bin/add  -valid -w $workdir
  $bindir/mvs -w $workdir -s 3-8
  ```

- pre-undistort images to use parallel version of bundler.

  $bindir/mvs -i $images -w $workdir -pre-process sample-list -pba

- on an ordered image set:

  ```
  $bindir/mvs -i $images -w $workdir -match 10
  ```

1.10 Troubleshooting

1.10.1 Execution errors:

If an error occurs during execution, try to run the underlying command by hand. Option **-v** or **-n** will show what is exactly executed. The directory corresponding to `intall/lib` should be added to **LD_LIBRARY_PATH** (to **DYLD_LIBRARY_PATH** on Mac OS).

1.10.2 Bad results:

Sometimes the resulting ply shows a poor or bad reconstruction. The most frequent symptom is that bundler (step 4) eliminates too many of the input images. This can be verified with check-bundle (see ‘ `images without matched points` ‘), or by comparing the number of “good” images in file `pmvs/list.rd.txt` (made at step 5) with the number of input images in file `orig-list.txt`.

The result of **bundler** depends on the contents of `matches.init.txt`:

- Run check-matches to analyze this file : option **-grp** will show if key matching defines a connected graph.
- Visualize the matched points between pairs of images with visu-matches (calls $bindir/cvdraw which needs OpenCV >= 2.4)

You can also vizualise keypoints with visu-sift.

To get better results:

- If `matches.init.txt` is bad, you can try to change the feature extractor program or its options or modify **-maxsize**.
- Tell bundler which initial pair of images to use (option **-bdlpair**). Use for example, “BEST pair” given by **check-matches**.

It may also happen that **pmvs2** gives bad results because of bad distorsion correction on some images. Undistorted images used by **pmvs2** are stored in `pmvs/visualize`.

In my experience, the result may be very sensitive to the feature extractor used, or even to the conditions of its compilation.

Of course the quality of images is fundamental. A good rule of thumb is that each reference point (i.e point detected by the feature extrcator) should be visible in at least 3 images.

1.11 Acknowledgments and history

This work was initiated by Jean Ponce, scientific leader of the INRIA-ENS research team WILLOW (http://www.inria.fr/equipes/willow, http://www.di.ens.fr/willow), to make use of software PMVS easier.

PMVS was developped by Jean Ponce and Yasutaka Furukawa. See http://www.di.ens.fr/pmvs/ page and following papers:

- 1. Furukawa and J. Ponce. **Accurate, Dense, and Robust Multi-View Stereopsis**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8), 2010. (http://www.di.ens.fr/willow/pdfscurrent/journee2010_sdp.pdf)
- 1. Furukawa and J. Ponce. **Accurate Camera Calibration from Multi-View Stereo and Bundle Adjustment**. *International Journal of Computer Vision*, 84(3), september 2009. (http://www.di.ens.fr/willow/pdfs/ijcv08a.pdf)
- 1. Furukawa and J. Ponce. **Accurate camera calibration from multi-view stereo and bundle adjustment**. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008. (http://www.di.ens.fr/willow/pdfs/cvpr08a.pdf)

Yves Ubelmann used this software, in particular for archeological sites. His feedback and data samples were of great help.

Recently, Jean Ponce suggested to convert the dense patch model of PMVS to a trianglularized surface with the Delaunay triangulation of CGAL. Many aspects of the programs follow his suggestions.

And the invaluable advices of Monique Teillaud (Senior researcher in team GEOMETRICA, INRIA Sophia Antipolis) about CGAL programming saved me a lot of time and effort.

CHAPTER

TWO

TOOLS

2.1 merge-ply

merge-ply [**-v**][**-n**][**-f**][**-wp**] **-d** *basename* [**-o** *global-ply*] [**-cl** | [**-s** *1st-num*] [**-nb** *nb-sets*]]
Converts points of image clusters to the coordinate system of one set.
Working directories are named 'basename-xx', where xx is the cluster number on 2 digits.
They must contain orig-list.txt (list of paths to original imges),
pmvs/bundle.rd.out and pmvs/models/option-000.ply.
Image lists must have common images.
Points of each set are saved in a separate file (n*basename*-xx.ply in current directory).
If -o is given, all files are concatenated in *global-ply*.

- **-v** : verbose
- **-n** : dry run (just print the commands)
- **-d** : basename of working directories (names $basename-nn)
- **-s** : number of 1st cluster (>=1, default 1)
- **-nb** : number of clusters (default 2)
- **-cl** : order of clusters is given in file clusters-*basename* (ignore -s and -nb)
- **-f** : force to overwrite results.
- **-wp** : make also ply files of points used to compute transformation matrices.

With option **-cl**, order of clusters is given by file `clusters-basename`. This file has 1 line of the form `n0 n1 n2 ...` per cluster. `n0` is the cluster number (starting at 1). The line with `n0` alone defines the reference cluster (= the reference coordinate system). There must be images common to cluster `n0` and `n1` and images common to `n1` and `n2`, and so on. Points of `n0` will be converted to the coord system of `n1` and then to the one of `n2`, and so on.

Without option **-cl**, the cluster order is given by the cluster numbers. So, `-s 1 -n 3` is equivalent to the cluster file:

```
1
2 1
3 2 1
```

The script first calls *calctransform_* to compute all the needed transformation matrices, and then calls *convply_* to apply them. The script skips computation of existing result files (matrices and .ply). Use **-f** to force overwriting of files.

2.2 cat-ply

cat-ply [**-patch**] **-o** *result ply-file+* Concatenate ply files and write to *result*.ply

- **-patch** : also cat .patch files to *result*.patch

2.3 merge-bundler

merge-bundler [**-v**][**-n**] **-d** *basename* [**-f**] [**-o** *result-dir*] [**-s** *1st-num*] [**-nb** *nb-sets*]
Converts bundler results of different clusters to the coordinate system
of one set, and merges all data in a result directory.
Directories must contain pmvs/newbundle.rd.out pmvs/txt/, pmvs/visualize/, pmvs/list.rd.txt.
They are named 'basename-xx', where xx is a 2 digits number that determines their order.

- **-v** : verbose.
- **-n** : dry run (just print the commands).
- **-d** : basename of working directories (names will be $basename-nn).
- **-s** : num of 1st set (>=1, default 1).
- **-nb** : number of sets (default 2).
- **-o** : merge results in result-dir.
- **-cl** : order of clusters is given in file clusters-*basename* (ignore -s and -nb)
- **-f** : force to overwrite files.

2.4 check-bundle

check-bundle [**-zmin** *file*] [**-draw**][**-m|-wm** *file*] [**-coefs**][**-pcoefs**][**-p** *cam-num dimx dimy*] *bundler-file*

- **-zmin** : writes in *file*.ply points between zmin and zmin +0.2 deltaz
- **-draw** : writes in xb00*.pgm matched points of the corresponding image
- **-p** : print 2d and 3d points for camera cam-num and image dimension dimx dimy (set the 2d origin at top left corner)
- **-pcoefs** : print cameras params (focal, k1, k2)
- **-coefs** : print average coefs of cameras (focal, k1 k2)
- **-m** : print matrix of relations between images, with nb of matched points.
- **-wm** : write adjacency matrix of the graph, with nb of matched points in each slot.

This script is supposed to help you analyze the result of bundler when results are not satisfactory. It mainly gives information on the dependencies between images. These dependencies are much more accurate than those obtained in *matches.init.txt*.

The script prints the upper triangle of the dependency matrix : line k starts with the image number and the total number of matched points, followed by matching images with the number of matches in the pair.

Option **-grp** will show the unreferenced images (first line) and the independant graphs. A good set of images should be represented by only one graph.

2.5 check-matches

check-matches [**-n** *nb-images*][**-v**][**-grp**] <matches.init.txt Display information about image matching from matches.init.txt.

- **-v** : print image pairs
- **-grp** : print list of images in matching graphs (1st line = list of isolated images)
- **-n** : check image indices against nb of images

2.6 get-clusters

get-clusters [**-o** *nb-overlap*[*-rec**]][**-minp** *min-match-nb*][**-v**][**-grp**][**-G|-Gm** *file*][**-ext** *im0 im1*] *matches.init.txt* Extract clusters from matches.init.txt

- **-v** : print image pairs
- **-Gm** : write connectivity matrix (needed by clustering).
- **-grp** : print image groups
- **-G** : write dot graph
- **-minp** : eliminate pairs with too few matches
- **-ext** : only consider images in range [im0,im1[
- **-o** : simulate partial matching with max nb of images to check (for tests)
- **-rec** : associated with -o, try to reconnect clusters (tests)

2.7 adj2metis

Usage ./adj2metis [**-wm|wn** [**-i**]] < *adj-matrix* >metis-graph-file Convert an adjacency matrix to a metis graph.

- **-wm** : weight edges with nb of matched points
- **-wn** : weight edges with degrees of their vertices
- **-i**: invert weights

2.8 visu-sift

visu-sift [**-d** *srcdir* | **-f** *jpegfile*] [**-r**] For each jpeg image in srcdir, or for jpegfile :
write a BW version of the file in ./x*filename*.pgm,
and draw sift points from source file in green or red

- **-d** : make keypoint images for all images in workdir
- **-f** : make keypoint image of the file (uses the associated .key.gz)
- **-r** : draw red points instead of green.

2.9 visu-matches

visu-matches [**-m** *match-file*] **-n** *imgnum1 imgnum2* [**-o** *prefix*] [**-v|-draw**]
Write in *prefix*><*num1*.pgm and *prefix*><*num2*.pgm matched key points
according to matchfile (default matches.init.txt)
The script must be executed in the workdir (= with match-file and images).

- **-n** : image nums of a pair
- **-o** : prefix for resulting images (def xm)
- **-v**: print key pairs on stdout
- **-draw** :
  - print matching points to match* **-points.txt**
  - put the 2 images side by side in xmatch.png and join matching points
  - do the same in ximatch.png after removing outliers

2.10 calctransform

calctransform -d *target-dir src-dir* **-l** *imglist1 imglist2* [**-nc** *max-nb*] [**-wm** *output-matrix*] [**-wp** *output-points*] [**-v**]

Uses files pmvs/bundle.rd.out to compute the matrix to convert a PMVS model in src-dir into the coordinate system of target-dir model.

- **-d** target-dir src-dir: directories containing bundle/bundle.out for target and src models
- **-l** imglist1 imglist2: files containing original lists of images. They must have common images
- **-nc** max-nb: use max-nb common images (def all common ones)
- **-wm** output-matrix: file for saving transform matrix
- **-wp** output-points: write ply file of points used to compute transform.
- **-v**: verbose

Example:

```
$bindir/tool calctransform -d j-01 j-02 -l j-01/orig-list.txt j-02/orig-list.txt -wm M-j-0201
```

2.11 convply

convply -m *matrix1,...,matrixn* **-i** *input-ply* [**-o** *out-ply*]

Successively apply the given transform matrices to convert a 3D PMVS2 model

- **-m** matrix1,...,matrixn: transformation matrices to apply successively to move from coord system n to 1
- **-i** input-ply: ply file to convert
- **-o** out-ply: ply output file (default result.ply)

Example:

```
$bindir/tool convply -m M-j-0302,M-j-0201 -i j-03/pmvs/models/option-0000.ply -o nj-03.ply
```

2.12 convbundler

convbundler -m *matrix1,...,matrixn* **-d** *work-dir*

- **-m** matrix1,...,matrixn: transformation matrices to apply successively to move from coord system n to 1
- **-d** work-dir: directory containing files to convert (pmvs/bundle.rd.out pmvs/txt/)

2.13 drawcameras

drawcameras -o *output-ply* **-i** *bundler-file*

Make a ply file of cameras positions.

- **-o** output-ply: ply file for cameras positions
- **-i** bundler-file: bundler file (bundle/bundle.out or pmvs/bundle.rd.out)

Example:

```
$bindir/tool drawcameras -i pmvs/bundle.rd.out -o pmvs/cameras.ply
```

2.14 calcamera

calcamera [**-s** *dimx dimy*] [**-m** *projection-matrix*] [**-v**] **points-file**

Compute the projection matrix for 2d/3d point pairs given by the user.

- **-s** dimx dimy: image x y dimensions
- **-m** projection-matrix: file to store the projection matrix
- **-v**: verbose mode
- **points-file**: lines of 2d/3d corespondent point coordinates as 'x y X Y Z'

Prints K (intrinsics), R (rotation) and t (translation) for the camera
K is of the form:
xfocal skew-angle x0
0   xfocal   y0
0   0   1

2.15 projection

projection -d *srcdir* **-i** *image* [**-b** *numcam*] [**-p** *proj-matrix*] [**-pts** *points-file*] [**-col** *zoom*]

- **-d** srcdir: directory containing bundle/ and pmvs/
- **-i** image: image for projection
- **-b** numcam: use camera <numcam> from bundle.rd.out (default -b 0)

- **-p** proj-matrix: use projection matrix stored in given file
- **-pts** points-file: ply file containing 3d points
- **-col** zoom: draw squares of size 2n+1 with original color of points

2.16 metis2cluster

metis2cluster -l *img-list* **-o** *out_name* **-g** *metis-graph* **-c** *metis-nb-clusters* [**-n**] [**-ply**] [**-dot**] [**-b** *bundle-file*] [**-ax**]

Build cluster data (lists of images, clusters dependencies) for Multi-View Stereo programs.

- **-l** img-list: list of paths of original images
- **-o** out_name: basename for results (lst<name>-01,lst<name>-02 ...,clusters-<name>
- **-g** metis-graph: file basename of the metis files (i.e without the .metis extension)
- **-c** metis-nb-clusters: nb of clusters made by gpmetis (file <metis-graph>.metis.part.<nb>)
- **-n**: simulation : do not write files
- **-ply**: print cluster cam positions in .ply files
- **-dot**: print cluster data in .dot files
- **-b** bundle-file: bundle-file used for camera positions when writing dot files
- **-ax**: alternate conection algorithm : choose images directly connected to a cluster, rather than a connected subgraph

Results are clusters-<out_base>, lst<out_base>-01 (cluster 1 images list), xlst<out_base>-01 (short list for mvs pre-process), ...

Example:

If metis data is test.metis and test.metis.part.2:

```
$bindir/tool metis2cluster -l orig-list.txt -o test -g test -c 2
```

CHAPTER

THREE

DEALING WITH LARGE IMAGE SETS

Execution time can increase rapidly as the number of images grows. In particular, step 2 (feature matching), is in $O(N^2)$.

For example, on a host with 48Gb of memory and 24 cpus at 2.6Mhz, the main elapsed times for a set of 90 1200x900 images (about 2000 keypoints / image) are:

```
- step 1 (extractFeatures) took 11.719s
- step 2 (matchKeys) took 49.144s
- step 3 (makeOptions) took 0.001s
- step 4 (execBundler) took 425.344s
- step 5 (bundle2PMVS) took 2.065s
- step 6 (execCmvs) took 1.058s
- step 7 (execGenOptions) took 0.008s
- step 8 (execPmvs) took 104.782s
```

3.1 Simple optimisations:

- keep the number of key points reasonable (use option **-maxsize**). About 3000 key points seems to be enough.
- if you have OpenCV >= 2.4, try **-ocv sift**; it seems to make features extraction and key matching faster.
- try **-pre-process** + **-pba** to speedup bundler
- if images are mostly ordered, use **-match** to speedup step 2 (for example `-match 8`). Use check-matches (see doc) to check the result:

  ```
  $bindir/tool check-matches -grp $workdir/matches.init.txt
  ```

  This will display list of images in each matching graph. The first line lists images without match. Normally there should be only 2 lines : the first one empty, and the second one containing all images. But more than 2 lines with one containing almost all images could be acceptable. The quality of *matches.init.txt* has also an influence on bundler.

Another important solution is clustering input data.

3.2 Manual clustering:

This method supposes that you can easily build groups of overlapping images. Each group must have a few (about 10) images of an other one to be able to compute the transformation matrices necessary to merge the resulting sets of

3D points.

You must follow some naming conventions if you want to use the provided scripts.

Given a basename `$base`, and N images lists, named `lst-$base-01` to `lst-$base-nn`, you must create ditectories `$base-01` ... `$base-nn` and a file `clusters-$base`.

The cluster file has N lines of cluster numbers that describe the dependences between the clusters. The reference cluster (= the reference coordinate system), must be alone on a line. For example, suppose cluster 1 is the reference, the line `3 2 1` means that there are common images between clusters 3 and 2 and between 2 and 1. To convert points of cluster 3 to the reference coordinate system, we will apply the transformation from 3 to 2, and then from 2 to 1.

Once all the clusters have been processed by **mvs**, you can merge the results with **merge-ply**.

Example:

```
$bindir/mvs -i lst-test-01 -w test-01
$bindir/mvs -i lst-test-02 -w test-02
$bindir/mvs -i lst-test-03 -w test-03
cat <<EOF  >clusters-test
1
2 1
3 2 1
EOF
$bindir/tool merge-ply -d test -o all-test.ply -cl
```

This will create `./ntest-01.ply ./ntest-02.ply ./ntest-03.ply` and `all-test.ply`.

3.3 Automatic clustering:

This is an attempt to automatically clusterize a large set of images. It is very experimental and is given without any guarantee!

NB: the program **gpmetis** from ParMETIS (http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview) is required.

3.3.1 Algorithm:

1. Extract key points from all images:

   ```
   $binddir/mvs -i $datadir -w $workdir -s 1
   ```

2. Use `matches.init.txt` to build a connected graph of images relations (2 images are connected if they have matching points). This is the critical part. As we want to avoid $O(N^2)$ operations, we suppose that most of time, consecutive images have matching points:

   - First run the key matching program with a limited neighbourhood (4 to 8). We should obtain a "small" number of independant clusters.
   - Try to find a few images (tipically 4) that reconnect the clusters. We hope that the number of tests will not be too large.

   This operation is done by **n-KeyMatchFull**:

   n-KeyMatchFull *list_features.txt out-matchfile* [*min-match* [[*max-overlap*] *old-match-file*]]

   - *list_features.txt* : file created in workdir at step 1
   - *out-matchfile* : match file, similar to *matches.init.txt*

- *min-match* : minimum number of matched points between 2 images (default 16).
- *max-overlap* : number of consecutive image to check for matches (default 8)
- *old-match-file* : don't calculate initial matches, and directlly the cluster reconnection step.

```
cd $workdir
n-KeyMatchFull list_features.txt matches.init.txt
```

3. Split this graph in N subgraphs :
   - make the matching graph adjacency matrix (see get-clusters):

     ```
     $bindir/tool get-clusters matches.init.txt -Gm mybase
     ```

   - Convert adjacency matrix to metis graph (see adj2metis):

     ```
     $bindir/tool adj2metis <mybase.mat >mybase.metis
     ```

   - Split the graph into **N** parts with gpmetis (http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview). For example, with **N** = 5:

     ```
     gpmetis -ptype=kway -contig -ncuts=2 -objtype=cut mybase.metis 5
     ```

     The resulting file will be `base.metis.part.5` (line *k* give the cluster number of image *k*).

4. Use the initial graph to extend each subgraph with a few images that can interconnect it with another subgraph (see metis2cluster):

   ```
   $bindir/tool metis2cluster -l orig-list.txt -o mybase -g mybase -c 5
   ```

   We will obtain `clusters-mybase`, `lstmybase-01 xlstmybase-01` ... `lstmybase-05 xlstmybase-05`.

1. Run mvs on each cluster. For example:

   ```
   mkdir mybase-01
   $bindir/mvs -i lstmybase-01 -w mybase-01 -pre-process xlstmybase-01 -pba
   ```

2. Merge results (see *merge_ply_*):

   ```
   $bindir/tool merge-ply -d mybase -cl -o mybase-all.ply
   ```

CHAPTER

FOUR

INTERNALS

4.1 config

This module contains parameters of the application.

exception `BadParamException`
Bad parameter or file. The execution cannot continue

class `Config()`

Parameters of the application :

- progs : names of external progs called by the application
- params : configuration parameters
- bundlerOptions : default options for bundler
- execList : tasks to execute in given order

exception `ExecAbortException`
Programm terminated with error

exception `MissingInputFile`
Missing input file

`getStatus()`
Retrieve state of last add operation from file *workdir*/add-status

`initConf(`***args*`)`

`isFocalPredef()`
Get focal computed by pre-process from file *workdir*/pre-processed.txt.

Returns: None if no file or null focal in file, value of focal otherwise.

`prepSteps(`*l, firststep, log, maxstep=None*`)`
Set exec flags in steps list.

Input []

- l is a 1 or 2 elements list (1st and last step, 1 to 8)
- firststep = minimum value required for 1st step number.
- maxstep = max possible step if < 8

Returns : (stat,istart); stat = True if correct, istart = index in step table (0-7)

setStatus (*level*)
Writes the status of an add operation to file *workdir*/add-status

4.2 bundler

Execute tasks corresponding to the 4 first steps (up to bundler).

class **Bundler** (*lst, binpath=None, add_num=0*)
Process programs of the bundler package

bundle2PMVS ()

Prepare data for pmvs

- input : list.txt, bundle/bundle.out
- output : (in pmvs/) bundle.rd.out, list.rd.txt, pmvs_options.txt, vis.dat, txt/*.txt, visualize/*.jpg, prep_pmvs.sh

doBundler ()
Sequence of programs to execute from the bundler package

execBundler ()

Execute bundler program

- input : list.txt, options.txt, matches.init.txt
- output : out, bundle/*.out, bundle/*.ply, constraints.txt, pairwise_scores.txt, matches.*, nmatches.*

extractFeatures ()
Extract camera focal length from input images and calculate key-points. Start features extraction program in parallel on al cpu's.

•copy-rename images to *workdir*

•make list.txt, list_features.txt, *.key.gz (sift keys),

makeOptions ()
Builds the option file (options.txt) for bundler

matchKeys (*keylistfile*)

Call the key matching program.

- input : list_features.txt
- output : matches.init.txt

4.3 cmvs

Call cmvs

class **Cmvs** ()
Process programs of the cmvs package

doCmvs ()
Optionally run the 2 steps of CMVS : cmvs, genOptions

execCmvs ()

Execute cmvs

- input : (in pmvs/) bundle.rd.out list.rd.txt pmvs_options.txt txt/*.txt visualize/*.jpg
- output : pmvs/centers-*.ply pmvs/ske.dat pmvs/vis.dat (overwrite vis.dat from bundle2PMVS)

`execGenOptions``()`

Make option files for pmvs

- input : pmvs/ske.dat
- output : pmvs/pmvs.sh pmvs/option-*

4.4 pmvs

Call pmvs2

class `Pmvs``()`

Process programs of the pmvs2 package

`doPmvs``()`

Optionally execute pmvs2

`execPmvs``()`

Call pmvs2 program

- input : (in pmvs/) /option-*, vis.dat, visualize/*.jpg, txt/*.txt
- output : (in pmvs/) models/{*.ply, *.patch, *.pset}

4.5 preprocess

`get_distorsion``()`

Compute average focal and distorsion coefficients from a bundle file (*workdir*/bundle/bundle.out). Ignore focals that differ from 1st focal by more than 10% (outliers ?)

Returns: (focal,k1,k2) as strings

`get_focal``()`

Retrieve the focal of the first camera from *workdir*/list.txt

Returns []

- the focal value as a float

Raise an exception if not available.

`save_focal``(`*s, sk1=None, sk2=None*`)`

Store focal and distorsion coefficients in file *workdir*/pre-processed.txt

Input:

- s : the focal value as a string
- sk1, sk2 (optionals) : distorsion coefs as strings

`undistort``(`*src, sfocal, sk1, sk2*`)`

Undistort images and copy them to *workdir*.

Input:

- src : the original images (as given by option **-i**)
- sfocal, sk1, sk2 : focal and distorsion coeffs (strings)

Output: in files named by images number on 8 digits in *workdir* (00000000.jpg, ...)

4.6 myimage

class `MyImage` (*fileDef*)
Interface for image list operations

`extractFocal` ()
Open the image file and extract focal.

`extractSiftKeys` ()

Extract feature keys from the image The jpeg file is converted to a BW pnm image, eventually with a smaller size. Keys data is written to a compressed text file in directory *workdir*, with same name as image and extension *.key.gz*.

`getInfo` ()
Return parameters associated to an image as a dictionnary of *param-name*:*value* pairs

4.7 camdb

class `CcdWidths` ()

Database of some cameras ccd widths as a dictionary

- keys are of the form *Make/Model* where *Make* and *Model* are respectively the camera maker and the model as they appear in the exif header of photos
- values are the width in mm of the CCD sensor

To add a new entry you must know the ccd width in mm of the sensor. But this is rarely present in the specifications of the camera. The best place to find the width in mm is http://www.dpreview.com : select the maker in the cameras menu.

4.8 fileutils

`backupFilesMaybe` (*paths*)
Make a backup copy of each file of the list, if the backup does not yet exist

`execInDir` (*workdir, args, **kwargs*)

•Execute a command in directory *workdir*

•Log the command in verbose mode

•Don't execute it in simulation mode

`execMaybe` (*name, func, *args*)
Execute a step of the MVS suite if validated by the user at script start.

Input:

- name : the execution step name

- func : the boolean function to call if the step is marked executable in Config.validExec
- args : arguments for *func*

Returns: True if step is skipped, the return value of *func* otherwise.

`getFilteredFileList`(*dir, rexp*)
Retrive files of a given directory that match a regular expression.

Input:

- dir : the directory to list
- rexp : compiled regular expression

Returns: Sorted list of files. Each list element is a dictionnary {'dir' : *dir*, 'name' : *filename*}

`getImagelist`(*src, firstnum=0*)
Return the list of image files

Input:

- src : original images (directory of list of paths, as given to option **-i**)
- firstnum : number of first image to return

Returns:

images as a list of dictionnary with items

- 'dir' : directory component
- 'name' : basemname of original
- 'newname' : name (%08d) that the file will have when copied to working directory

`mkdirsMaybe`(**paths*)
Create a list of directories if they don't exist

`progPath`(*progName*)
Return the absolute path of a given program name supposed to be in the bin directory of the software

`readBackup`(*file*)
Return (as a list) the content of the backup file associated to file (= with extension *.save*).

`readFileMaybe`(*file*)
Read a file if possible.

Returns the list of lines; empty list if the file cannot be read.

`restoreBackupFiles`(*dirs*)
Silently restores all backup file in each dir of the list

`rmBackupFiles`(*dirs*)
Silently remove all backup file in each dir of the list

`rmNoerr`(*paths*)
Silently remove a list of files

`stripReadFileMaybe`(*file*)
Read a file if possible and strip lines leading and trailing spaces.

Returns the list of stripped lines; empty list if the file cannot be read.

`validAdd`()

Make addition operation definitive : remove all files created by -add (backups and status file)

`validName` (*rexp, x*)
Check a name against a regular expression

Input:

- rexp : compiled regular expression (result of re.compile)
- x : the string to check

Returns: True if x matchex rexp

4.9 myoptions

`doc_of_options` (*options*)
Build a doc string from options list (removes some rst tags)

Input []

- options = list of option definitions (see parse_args below)

Returns: tuple made of usage line and concatenation of detailed options help strings.

`parse_args` (*options, argv, usage=<function usage at 0x2a8e578>*)
Retrieve options from command line arguments (argv), according to a list of tuples (options).

Input:

- options : a list of 5 elements tuples describing each option
  - 1st element : the option string (for ex -i). If None, option is only used for making the doc string.
  - 2nd element: number of arguments of the option.
  - 3rd element : True if option is mandatory.
  - 4th element : argument name(s).
  - 5th element : description string.
- **argv** [the list of command line arguments. Arguments starting with] '-' arte considered options and checked against *options*.
- usage : the usage function to call (myoptions.usage() by default).

Returns: (list of (option-strings,arg-value), list of non options arguments).

`prepare_doc` (*options, before='n', after='', name=None*)
Returns a string mde of options documentation with optional strings prepended and appended. The resulting string is stored in a global variable for use by usage()

`sphinxdoc_of_options` (*options*)
Build a doc string in sphinx format from options list

Input []

- options = list of option definitions (see parse_args below)

Returns: tuple made of usage line and concatenation of detailed options help strings.

`usage` (*msg=None*)
Prints the optional *msg* string, followed by the *cur_doc* string, that must have been built with prepare_doc

4.10 runutils

`save_history` (*argv, start, elapsedtime, cputime, add_mode=False*)

Write information to the history file *work-dir*/history.

Input:

- start : True if the command is starting, False if at end of command
- argv : the arg list of the command, to be written if *start* is True
- elapsedtime, cputime : written if *start* is False

The file is overwritten if step 1 is executed, otherwise the data is appended to the file.

`verifSteps` (*steps, firststep, usage, maxstep=None*)

Verifies validity of the steps argument string. Check that files needed by 1st step exist in *workdir*.

Input:

- steps : the steps string (‘d’ or ‘d1-d2’)
- firststep : minimum first step allowed
- usage : usage function to call in case of error
- maxstep : maximum step allowed (None = 8)

Returns: call usage if string is incorrect, raises an exception if some required files are missing.

CHAPTER

FIVE

TRIANGULATION

5.1 Introduction

Given a dense set of 3D patches produced by PMVS2 (http://www.di.ens.fr/pmvs/), we want to build a triangle facets representation of the surface of the scene.

The programs use the Delaunay-3d module of **CGAL** library, version 4.1 : Computational Geometry Algorithms Library, http://www.cgal.org

The surface generation needs two programs :

1. **delaunay**
    - make a delaunay 3D triangulation of PMVS + camera poinrs.
    - as PMVS returns the list of points visible from each camera, we can draw a path from a camera to the points it sees, and find which tetrahedrons are crossed.
2. **triangclean**
    - use a criterion based on the number of rays crossing a tetrahedron to eliminate useless tetrahedrons.
    - find facets of remaining tetrahedrons that belong to the surface..

5.2 delaunay

This program builds a Delaunay triangulation and compute information about tetrahedrons visibility. It is based on the CGAL library (http://www.cgal.org).

We first build a delaunay triangulation on the points given by **PMVS2** plus the positions of cameras. Then we compute the number of intersections of each tetrahedron with rays between each camera and the corresponding PMVS2 visible points.

The loop on camera is parallelized with *OPENMP*.

delaunay1 is the sequential version of **delaunay**.

There is also a program **delaunayb** which gets it's data from the bundler result (2nd argument should be `pmvs/bundle.rd.out`).

5.2.1 Arguments:

delaunay *cam_points points-basename* [**-D** *debug-level*] [**-o** *outfile*] [**-a** *coef_min coef_max*]
Make a delaunay triangulation with PMVS points and camera positions,

and compute intersections of tetrahedrons with rays issued from camera points.

- *cam_points* : ply file of camera positions, with color attribute (generated by tool **drawcameras.py**).
- *points-basename* : basename for .ply and .patch files
- **-D** : debug level, level / 10 for display, level % 10 for interactivity
- **-o** : result file (default output.cgal).
- **-a** : add a point (barycenter) to "large" tretrahedrons, i.e with an edge >= coef_min*average-edge-length and <= coef_max*average-edge-length.

option **-D** is only for debug purpose and small points sets.

5.2.2 Detailed description:

- input data : input data comes from 3 files:
  - the *.ply* file of the camera positions and normals. This file is made by drawcameras.
  - the *.ply* file of PMVS2 points (with normals and colors), for example *pmvs/models/option-0000.ply*.
  - the *.patch* file made by PMVS2 and giving visible points for each camera, for example *pmvs/models/option-0000.patch*
- output data : it is a binary file defined by option **-o**, which defaults to *output.cgal*. Conventionally its extension should be *.cgal*. It will contain all the information : delaunay triangulation, PMVS2 points (with normals and colors), cameras, number of ray intersections with tetrahedrons ... This file can be quite big.
- messages : The program will print some statistics, execution times and information about the data. The number of valid cameras is less than number of cameras when some cameras have no visible points. Delaunay triangulation will remove duplicate points.

Points from PMVS2 are normally dense, but may be sparse in some regions. Most tetrahedrons are small, but they are larger in sparse regions. And there are generally artificial large tetrahedrons on the boundary of the scene. You may want to try to fill sparse region, but not obviously empty regions. To do so we propose to add barycenters of "medium" size tetrahedrons to the set of points. Affected tetrahedrons are those having their longest edge bounded by a minimum and maximum value.

These value are equal to the average edge length multiplied by the 2 coefficients of option **-a**. All this is done, of course, before insertion of camera points. But be careful, this operation may give bad results.

5.3 triangclean

This program will make a surface reconstruction of a scene from the result of program **delaunay** described above.

5.3.1 Arguments:

triangclean *input_cgal out_ply* [**-e** *extract-mode*] [**-i** *min_intersect* | **-mxf** *nb-intersections*] [**-iw**] [**-a** *normals-angle*] [**-ct**] [**-lg** *lg_coef*] [**-s** *surf_coef*][**-sm** *lambda nb_iter*]
Extract, from the result of program delauny, the facets of the surface of the scene,
and write the result to a ply file in binary format The program first eliminates empty tretrahedrons
and then determines facets belonging to the surface.

- *input_cgal* : data generated by program **delaunay**.
- *out_ply* : .ply output file.

- **-e** [extraction mode :]
  - 0 = use removed tretrahedrons
  - 1 = use kept tretrahedrons
  - 2 = extract all facets of removed tretrahedrons (for test purpose)
  - 3 = extract all facets of kept tetrahedrons (for test purpose)
- **-i** : remove tetrahedrons with at least *min_intersect* rays going through them.
- **-iw** : use a number of intersections larger than *min_intersect* for "large" tetrahedrons
- **-mxf** : use maxflow algorithm, with a threshold of *nb-intersections*, to remove tetras
- **-a** [remove facets that do not satisfy a constraint on normal angles.]
  - if *normals-angle* > 0, rm facets where the angle between the facet normal and a vertex normal is greater than angle
  - if *normals-angle* < 0, rm facets where the angle between the facet normal and the sum of vertex normals is greater than |angle|. Angle value from 0 to 180; default value 180 means "no angle constraint".
- **-ct** : remove tetrahedrons that have 3 facets on the surface (rough patches).
- **-sm** [try to smooth the surface. Replace each point p by *lambda* * p + (1 - *lambda*) * neighbours_mean; repeat *nb_iter* times.] if *lambda* <= 0., use default value of 0.8; if *nb_iter* < 0, do a weighted mean based on normal angles.
- **-lg** : remove tetrahedrons with too long edges (greater than *lg_coef* * mean-edge-length. Default value of 0.14 if *lg_coef* = 0.
- **-s** : remove tetrahedrons with too large facets (surface greater than *surf_coef* * mean-edge-length * mean-edge-length. Default value of 0.01 if *surf_coef* = 0.

5.3.2 Description:

The algorithm is as follows:

1. Find tetrahedrons to remove according to their number of intersecting rays (computed by **delaunay**), 2 possibilities :

- Remove tetrahedrons having a number of intersections greater or equal to the value given by option **-i** (1 by default). This number can be weighted by the tetrahedron size (option **-iw**).
- Minimize an energy function with min-cut/max-flow algorithm (option **-mxf**):

  $E(x) = \sum_{v \in \nu} \alpha_v x_v + \sum_{(u,v) \in \epsilon} \beta(u,v) \mid x_u - x_v \mid$

  ν being the set of cells we want to remove, n_v the number of rays crossing node v and n_0 the number of inersections threshold, the cost α_v is defined by $n_v - n_0$. The smoothness coefficient β represents the similarity between adjacent tetrahedrons u and v.

1. Optionally (**-ct**), add to this list tetrahedrons having 3 faces adjacent to infinite tetrahedrons (rough patches).
2. Find facets canditate for the surface. Two possibilities (option **-e**):

- (**-e** *0*, default) for each removed tetrahedron, find "mirror" facets of its facets adjacent to a non removed finite tetrahedron. This is a bit faster, because the number of removed tetrahedrons is less than the number of valid ones.
- (**-e** *1*) for each kept tetrahedron, find facets that are adjacent to an infinite or to a removed tetrahedron.

1. Ignore some of the selected facets, considered as artefacts, accordingg to following optional criterions :

- too long facet : an edge is longer than some threshold proportionnal to the average edge length (**-lg** *coef*).
- too wide facet : its surface is greater than a threshold proportional to the square of the average edge length (**-s** *coef*).
- too badly oriented facet : the angle of its delaunay normal with one of the PMVS normals at the vertices or with the average vertices normal is too great (**-a** *angle*).

Unfortunately, there is no rule to choose optimal parameters.

5.4 Utility tools

Here are some graphical tools that will bring some little help to analyze results of PMVS2 and triangulation.

Valid cameras are drawn in green, and bad ones (with no visible points) in red.

These programs will be slow on large data sets.

5.4.1 camtest

camtest file.cgal *facets.ply* [**-b** *x1 y1 x2 y2*] View the results of PMVS and triangulation in an OpenGL window .

- **-b** : Only draw rays to points in the XY box.

To see which points are visible by a camera : hit keyboard key **S**. This will toggle drawing red rays from current camera to its visible points.

For more readability, you can restrict target points to an XY box.

Use the *Select* function (*shift+LeftMouse*) to print the xy ccordinates of a point of the scene.

5.4.2 camtest0

camtest0 cameras.ply *model-basename*[*-b** x1 y1 x2 y2*]
Display PMVS points and cameras, and draw rays to visible points.
model-basename is the basename of files .ply and .patch built by PMVS2.

- **-b** : Only draw rays to points in the XY box.

5.4.3 cmpcgal

cmpcgal {points1.ply *points2.ply | data1.cgal data2.cgal}* [**-b** *x1 y1 x2 y2*
Alternatively display 2 datasets for comparison,
either facets (.ply) or delaunay (.cgal) files.

- **-b** : display points inside xy box

5.4.4 hist

hist input.cgal Repartition of number of intersections
Each line gives the nb of tetrahedrons more intersections than given after '>'.

5.5 Examples

Notations :

- *$bindir* = intallation bin directory
- *$workdir* = working directory for multi-view-stereo programs.

```
cd $workdir/pmvs
$bindir/tool drawcameras -i bundle.rd.out -o cameras.ply
```

Visualize how cameras see points:

```
$bindir/camtest0 cameras.ply models/option-000
```

Triangulation:

```
$bindir/delaunay cameras.ply models/option-000
$bindir/triangclean output.cgal scene.ply
```

Extraction variant:

```
$bindir/triangclean output.cgal scene2.ply -i 5
```

Compare results:

```
$bindir/cmpcgal scene.ply scene2.ply
```

5.6 Programming Documentation

The internal programs documentation, made with Doxygen, is available here in html and as `triangulation-refman.pdf` in pdf.

CHAPTER

SIX

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

INDEX