Surfel Tools

J.P. Mellor

15 October 1999

1 Building the System

The following packages are required to build the surfel tools

- GNU C compiler
- X11
- OpenGL or Mesa
- XForms
- Pthreads
- Dynamic link library
- libc6 or better (linux only)

The sytem is organized as a top level directory containing configuration files and some scripts and several sub directories. The source code is contained in the `./src` directory, header files in `./include`, and binaries (including object files) in `./bin`. With `./bin` there are separate directories for each architecture.

1.1 Autoconf

The entire system uses *autoconf* to simplify configuring and building the system on multiple architectures. Currently, IRIX and linux are the only tested operating systems. The top level configuration file is *configure.in*. This file describes the tests used to check the preconditions for building. It also describes files to be built and preprocessor variables to be set during the configuration. The configuration script *configure* is created by autoconf from configure.in. Configure is a simple shell script. Some of the most important command line options are

- –enable-libc5 - Not appropriately named. Should be used on all linux builds.
- –with-MESA-LIB - Specify the path to the Mesa libraries.

- –with-FORMS-LIB - Specify the path to the XForms libraries. This is necessary on all builds unless the library is installed in a standard location.
- –with-MESA-INC - Specify the path to the Mesa header files.
- –with-FORMS-INC - Specify the path to XForms header files. This is necessary on all builds unless the headers are installed in a standard location.
- –enable-no-threads - Don't start multiple threads
- –enable-no-ui - No gui code

Configure creates all of the makefiles used during the build, a top level header file, and several executable scripts. On tess

```
LIBS="-L/usr/X11R6/lib $LIBS"
```

must be added to the top of configure.

If you're not sure what options to use, look at the comments at the top of configure.status. For example,

```
./configure --enable-libc5 --with-FORMS-LIB=/home/jpmellor/xforms/FORMS
      --with-FORMS-INC=/home/jpmellor/xforms/FORMS
```

might be used to configure the system on tess and

```
./configure  --with-FORMS-LIB=/u2/graphics/XForms/FORMS_irix_n32
      --with-FORMS-INC=/u2/graphics/XForms/FORMS_irix_n32
```

might be used on IRIX.

The files created by configure are generated from .in files using variable substitution. For example, ./Makefile is created from ./Makefile.in. The files currently built are

- src/config.h
- src/analyze/Makefile
- src/calibrate/Makefile
- src/control_panel/Makefile
- src/examiner/Makefile
- src/get_surfel/Makefile
- src/graph/Makefile
- src/grow/Makefile
- src/image_info/Makefile
- src/list/Makefile
- src/misc_utilities/Makefile
- src/misc_xforms/Makefile
- src/mods/Makefile
- src/process/Makefile
- src/region_viewer/Makefile

- src/review/Makefile
- src/rgb_image/Makefile
- src/stereo_viewer/Makefile
- src/test_surfel/Makefile
- src/texture_viewer/Makefile
- src/threed_info/Makefile
- src/threed_viewer/Makefile
- src/tool/Makefile
- src/utilities/Makefile
- src/Makefile
- Makefile
- Makefile.opt
- Makefile.std
- testit
- testit_545
- testit_549
- testit_565
- testit_575
- testitb
- testitb1

1.2 Make

After configuring the system, all that is needed is to run make in the top level directory.

2 Running the System

2.1 Tool

tool is the main executable of the system. It loads images and cameral calibration data and allows interactive exploration, as well as, viewing the results of automatic reconstruction. All of parameters set by command line options can be set interactively on a control panel. In general, the inputs are image and calibration data and the output is a list of surfels. Some of the more useful command line options are:

- -d <debug level> | --debug <debug level> - Turn on some debugging messages.
- -g | --no_gui - Turn off the gui. Useful for automatic reconstructions.
- -p <number of processes> | --processes <number of processes> - The number of threads to use for automatic reconstruction.
- -c <control file> | --control_file <control file> - File listing image and calibration data to be loaded.
- --calibration_file <calibration file> - Updates to calibration data loaded in control file.
- --log_file <log file> - File in which to record hypotheses tested.

- `--output_file <output file>` File in which to record recovered surfels.
- `--model_file <model file>` - Model to load. Used only for visualization.
- `--surfel_file <surfel file>` - Surfels to load.
- `--dl_file <dl file>` - File containing dynamically linked functions. Used to modify how hypotheses are selected. See `./src/mods`
- `--upper_left <x y z>` - One corner of the reconstruction volume.
- `--lower_right <x y z>` - Opposite corner of reconstruction volume.
- `--deltas <x y z a>` - Spacing between hypotheses.
- `--min_cameras[_final] <minimum number of cameras>`
- `--min_nodes[_final] <minimum number of nodes>`
- `--max_match[_final] <maximum match error>`
- `--min_unique[_final] <minimum uniqueness>`
- `--min_config[_final] <minimum configuration score>`
- `--min_interest[_final] <minimum interest score>`
- `--max_cameras <max cameras>`
- `--max_tries <max tries>`
- `--max_color_shift <max color shift>`
- `-w <region width> | --region_width <region width>` - Width of surfel in samples.
- `-h <region height> | --region_height <region height>` - Height of surfel in samples.
- `-t <region translate> | --region_translate <region translate>` - Maximum translation to compensate for camera error.
- `--region_sample_size <region sample size>` - Sample size in world coordinates.

2.2 Calibrate

calibrate imposes a consistent set of shifts and updates the camera calibrations. The input is a list of raw surfels and the output is a list of consistent surfels and calibration updates. The command line options are:

- `-d <debug level> | --debug <debug level>` - Turn on some debugging messages.
- `-c <control file> | --control_file <control file>` - File listing image and calibration data to be loaded.
- `-f <calibration file> | --calibration_file <calibration file>` - Updates to calibration data loaded in control file.
- `--surfel_file <surfel file>` - Surfels to load.
- `--model_file <model file>` - Model to load. Used only for visualization.
- `--output_file <output file>` File in which to record recovered surfels.
- `-i <update1 file> | --update1_file <update1 file>` - Initial calibration update.
- `-u <update2 file> | --update2_file <update2 file>` - Final calibration update.
- `-t <max consistent shift> | --max_shift <max consistent shift>` - Maximum consistent shift.

2.3 Grow

grow imposes the "1 pixel, 1 surfel" constraint and groups surfels. It will also fit planes to groups and extract textures. The input is a list of surfels. The output is a list of (possibly textured) surfels. The command line options are:

- `-c <control file> | --control_file <control file>`
- `-f <calibration file> | --calibration_file <calibration file>`
- `-s <surfel file> | --surfel_file <surfel file>`
- `-m <model file> | --model_file <model file>`
- `-z <max dz for neighbor> | --delta_z <max dz for neighbor>`
- `-p <max dp for partial occlusion> | --delta_p <max dz for partial occlusion>`
- `-x <max spacing for neighbor> | --delta_x <max spacing for neighbor>`
- `-n <max dn for neighbor> | --delta_n <max dn for neighbor>`
- `-r <root for storing texture> | --texture_root <root for storing texture>` - Directory in which to store textures and masks.
- `-o <output file> | --output_file <output file>`
- `--raw` - Do not fit planes to groups.
- `-t | --no_texture` - Do not extract textures.
- `-g | --no_gui`
- `--no_prune` - Do not impose the "1 pixel, 1 surfel" constraint.
- `--min_cameras <minimum number of cameras to retain surfel>`
- `--min_surfels <minimum number of surfels to retain surface>`
- `--prune_groups <perform pruning on groups?>`
- `--prune_global <perform pruning on global?>`
- `--no_groups <output group files?>`

3 How I got the Results in the Thesis

1. Compile `./tess/test_ts_scripts/make_total.c` using the compile line at the top of the file.

2. Create a symlink in `./tess/test_ts_scripts/` from `make_test_ts` to the executable compiled above.

3. Make sure there is a symlink in `./tess/` pointing from `start_new_job` to `start_new_job_passive`.

4. Modify `./tess/crontab` as appropriate and add it to your user crontab, see the man page for more details.

5. Run `./tess/init_tess`.

6. Results and log files will accumulate in `./results_new`. Currently running jobs will be listed in `./tess/test_ts_running`. See below for more discussion about managing tess.

7. When the run is complete, run `./tess/check_output` in `./results_new` to check the individual output files. If any are missing or corrupt, rerun those portions.

8. When all of the output is satisfactory, run `./tess/combine_output` in `./results_new` produce single surfel file (`./results_new/output_combined_001`).

9. Run `./bin/linuxbin/calibrate` on the combined surfel file with max shift of 1 to produce a set of consistent surfels (`./test/output_consistent_combined_001_n1`).

10. Run

```
./bin/linuxbin/grow -c data/control\_all
    -s test/output\_consistent\_combined\_001\_n1
    -z 15 -x 300 -n 0.92 -g --prune\_groups 1 --prune\_global 1
    --raw -o surfdump11
```

    to produce a set of pruned and grouped surfels. Files of the form `surfdump11_group`*???* should be created in top level directory.

11. Compile `./tess/test_ts_scripts/make_scripts.c` using the compile line at the top of the file.

12. Create a symlink in `./tess/test_ts_scripts/` from `make_test_ts` to the executable compiled above.

13. Make sure there is a symlink in `./tess/` pointing from `start_new_job` to `start_new_job_passive`.

14. Modify `./tess/crontab` as appropriate and add it to your user crontab, see the man page for more details.

15. Run `./tess/init_tess`.

16. Results and log files will accumulate in `./results_new`. Currently running jobs will be listed in `./tess/test_ts_running`. See below for more discussion about managing tess.

17. When all of the output is satisfactory, run `./tess/combine_output` in `./results_new` produce single surfel file (`./results_new/output_consistent_combined_001_n1_grow`).

18. Run

```
./bin/linuxbin/grow -c data/control\_all
    -s test/output\_consistent\_combined\_001\_n1\_grow
    -z 15 -x 300 -n 0.92 -g --prune\_groups 1 --prune\_global 1
    --raw -o surfdump13
```

to re-prune and re-group the surfels.

19. Run

```
./bin/linuxbin/grow -c data/control\_all
    -s test/output\_consistent\_combined\_001\_n1\_grow
    -z 15 -x 300 -n 0.92 -g --prune\_groups 1 --prune\_global 1
    -t -o surfdump14
```

    to produce a model.

20. Run

```
./bin/linuxbin/grow -c data/control\_all
    -s test/output\_consistent\_combined\_001\_n1\_grow
    -z 15 -x 300 -n 0.92 -g --prune\_groups 1 --prune\_global 1
    -r texutures -o surfdump12
```

    to produce a textured model.

4 Modifying the System

The source code is divided up into several sub directories.

- `calibrate/`
- `control_panel/` - Routines to interactively change parameters.
- `get_surfel/` - Utility to interactively enter surfel parameters.
- `grow/`
- `image_info/` - Routines for loading image and calibration data.
- `list/` - Routines for doubly linked lists (can be thread safe).
- `misc_utilities/`
- `misc_xforms/`
- `region_viewer/` - Routines for displaying interactive surfel tests.
- `review/` - Routines for reviewing surfel solutions.
- `rgb_image/` - Routines for reading and writing rgb images.
- `stereo_viewer/` - Stereo pair view.
- `test_surfel/` - Routines for testing surfels.
- `threed_info/` - Routines for manipulating model, surfels, etc.
- `threed_viewer/` - Routines for displaying model, surfels, etc.
- `tool/`

5 Tools for Using tess

Several tools for automatically managing lots of small jobs on tess can be found in `./tess`. The basic idea is that the scripts to be executed are placed in `test_ts_scripts` and symlinks to them are placed in `test_ts_todo`. *start_new_job* starts scripts on unloaded nodes of tess, moving the link from `test_ts_todo` to `test_ts_running`, and checks for completion, moving the link from `test_ts_running` to `test_ts_done`. *check_output* will check all of the individual output files. *combine_output* will combine all of the individual output files into a single file.