VISGEN: Radio Telescope Array Observation Simulator

Leonid Benkevitch

December 20, 2010

Abstract

The program `visgen` is the main part of the MAPS (MIT Array Performance Simulator) package. On input, it reads a visibility file prepared by `MAPS_im2uv` program, a file of radio telescope array specifications, and parameters of observation to produce a simulated visibility image as if it were a result of real observation with the specified array. Further processing requires `maps2uvfits` program to convert the native `visgen` visibility format into the standard `uvfits` file, which is readable by `AIPS` or `MIRIAD` packages.

This document is not a complete `visgen` description. It rather documents recent improvements and bug fixes based on the buglist and private communications with the `MAP` users and developers.

1 `VISGEN` Command line options

Usage: `visgen [options]`

`-n name`	name of the simulation
`-s name`	name of the site
`-A name`	name of file with the array definition (station types and locations)
`-G name`	name of a UV grid data file If multiple input polarisations are required specify multiple -G options. Assumed order is I,Q,U,V
`-M name`	name of a metafile listing gridfiles Format: pol identifier [I,Q,U or V] and filename per line [over-rides -G options]
`-V name`	name of the observation specification file

`-N`	do not add noise
`-S SEFD`	set global station system equivalent flux density. Default: 500
`-Z`	do not compute visibilities from UV grid data (does not apply to oob)
`-a`	turn on accounting
`-I name`	name of the ionospheric model file. Turns on ionospheric modelling
`-i name`	name of the ionospheric settings file. Default: $TEXTDIR/iono_settings.txt
`-O name`	include 'out-of-beam' (oob) sources listed in file 'name'
`-R name`	name of the RFI model file. Turns on RFI modelling (currently not implemented)
`-m num`	turn debugging output to level num. (-2 lots thru 2 little)
`-l`	treat the array as localized (non-VLBI). Only the baselines formed by the most remote stations are checked before the actual simulation.
`-v`	treat the array as global (VLBI). All the baselines are checked before the actual simulation.
`-D [IOMBN]`	turn on module-level dubugging (copious output) for one or many modules: `I` ionosphere `O` oob `M` makebeam `B` compute_beamconvl `N` add_noise

2 Brief description of `visgen` operation

The visgen program allows to "look" at the sky with a radio relescope or radio interferometer which consists of a number of receivers (an "array"), scattered over an area on the surface of the earth, or even distributed globally, as in case with the Very Large Baseline Array, or VLBI, array. The output of `visgen` is the image that the array can "see". At the moment of simulation the array may not exist, being at the stage of development or prototyping, thus `visgen` can provide valuable information related to the characteristics of the finished array design.

On input `visgen` requires the following information (the corresponding command line options are given for each item):

- Names of the simulation (option `-n`) and the site (option `-s`);
- The "ideal" sky image: the UV-grid data file generated by `MAPS_im2uv` program, option -G;
- The coordinates of array receiver stations (dishes, tiles, frames etc.), option `-A`;
- Parameters of each station (no option; obtained from array specification file);
- Specifications of the observation (RA, DEC, frequency, time, duration etc., in "obs_spec" file), opton `-V`.

There are also a few optional files that may be required dependent on the observation:

- File describing the out-of-beam sources, option `-O`;
- Files specifying the ionospheric parameters, option `-I`;
- The metafile listing UV-gridfiles for the case of several polarizations, option `-M`.

The `visgen` output is a file in native format containing the visibilities calculated over all the possible baselines of the array. Its name is not specified on the command line, and it is generated automatically as <simulation name>.vis. Typically, the similar observational data are acquired during real operation of the radio telescopes and interferometers.

Roughly, `visgen` operation is implemented as five nested loops, indicated below in the order of increasing depth:

- Loop over scans. An observation can include several scans, differing by frequencies, celestial coordinates, or integration parameters.
- Loop over integration time slices.
- Loop over baselines. Consists of two nested loops, bringing up all the possible baselines between the station pairs.

- Loop over observing frequency IFs.
- Loop over correlator frequency channels within an IF.

In the course of simulation `visgen` samples the visibility data from the uv-plane, provided by the `MAPS_im2uv` program, in the form of patches. The patch is located roughly at (u, v), where $u \leq (x_2 - x_1)/\lambda$, and $v \leq (y_2 - y_1)/\lambda$. Here x and y are coordinates of two stations, $(x_2 - x_1)$ and $(y_2 - y_1)$ determine the baseline length, and λ is the wavelength. Clearly, (u, v) must be within the uv grid, otherwise the simulation fails. The patch size is determined by the frequency bandwidth and the integration time, and the patch must also fit in the uv-grid, otherwise the simulation cannot continue. Simulations with high resolution images and arrays with large numbers of stations can require a significant amount of computing time, from hours to days, and a baseline or patch failure can happen when a significant time has already been spent. To avoid this, it is necessary to check if all the baselines and patches fit in the uv-grid before the simulation starts. This check has been implemented in the current version of `visgen`.

3 Latest improvements to `visgen`

3.1 Preliminary baseline checking

In order to check if all the baselines and patches fit in the uv-plane *before* the simulation starts, `visgen` launches a new subroutine, `check_baseline()`, which is actually a "mini-`visgen`" in the sense that it reproduces all the `visgen` operations for the baseline and patch generation and checking, but makes neither disk memory access nor integrations. The time needed for such a "dry run" is order(s) of magnitude less than that required for an actual simulation. However, in the case of arrays with large number of stations the number of baselines and patches is big enough to take several minutes to complete the check.

A good example is the SKA array with planned 2400 stations. Simple calculation gives the number of baselines to be analysed at $2400(2400-1)/2 = 2,878,800$.

3.2 Array convex hull computation

We have implemented a technique that dramatically reduces the number of the analysed baselines. Not all the baselines deserve testing, but only the longest. The longest baselines are produced by the most remote from each other station pairs, i.e. the stations located on the periphery of the array. We define the array periphery as the convex hull of all the stations considered as a set of points on the earth surface. In the case of localized (non-VLBI) arrays the earth surface can be approximated by a plane. The convex hull of a set of points can be defined as the minimal polygon whose vertices are the the points belonging to the set, and all the other non-vertex points belong to this polygon's interior.

After some analysis of the published convex hull algorithms we designed our own algorithm and implemented it in the C language as the subroutine `convex_hull_2d()` in the file `$SIM/visgen/convex_hull_2d.c`. The algorithm is very efficient. Its speed is apparently at the level $O(n \ln n)$ or even better (however, it still needs to be proven mathematically). The algorithm `convex_hull_2d()` uses ring lists, the list structures closed into ring, so that the last atom in such a list points at its first atom. The primitives for list creating, deleting, inserting, copying and printing are saved in the file `visgen/list_proc.c`. The lists are utilized to store indices of the points that are the poligon vertices. In a few steps the points belonging to the set of convex hull vertices are found and inserted into the vertex list.

The convex hull vertex indices of the analyzed array and the array $(east, north)$ coordinates are stored in the files `bd_idx.txt` and `points.txt`. To visualize the array and its convex hull a simple python script, `plot_hull.py`, has been written and stored in the `$SIM/visgen/scripts` directory. To use it, install the packages `matplotlib` and `ipython`. After `visgen`, simply run

```
$ plot_hull.py
```

to open the window with the view of array and its minimal polygon. For example, for the SKA array it shows that the whole array lies inside of the minimal poligon with 11 vertices. Therefore, for only 11 of 2400 stations the baselines are to be analyzed, or just $12 * 11 = 132$ baselines of the total of 2,878,800. This takes a split of a second.

Note that the convex-hull technique cannot be used if the array stations are

globally distributed as in the VLBI case. Therefore, for the VLBI arrays *all* the baselines are analyzed.

3.3 "Old" and "new" styles of array coordinates

The array convex hull calculation required to make the array coordinate input more consistent in the subroutine `$SIM/visgen/read_array_spec.c`. Now, undependent of "old style" $(east, west)$ and "new style" (x, y, z) of the station coordinates specification, both representations are stored in the stricture `struct array_specification arrayspec`. If the array coordinates in an array specification file from the directory `$SIM/array/` are given in the form of global Cartesian triplets (x, y, z), they are recalculated into the local $(east, north)$ coordinates relative to the projection of the array *centroid* on the earth surface.

3.4 Distinguishing VLBI and localized arrays

In the new `visgen` version a global VLBI flag has been introduced. Now the array is treated as VLBI or non-VLBI independently of the "old-style" or "new-style" station coordinate specifications. Two levels of the VLBI indication exist:

- VLBI line in the array specification file in directory `$SIM/array/`;
- the command line options `-v` and `-l` (higher priority).

To specify the array as globally distributed, insert line with the VLBI (or vlbi) word at the beginning of the array specification file. If there a no `-v` or `-l` options in the `visgen` command line, the array is processed as the VLBI or global array.

The command line option `-l` has higher priority: it will make `visgen` treat the array as localized, and it will suppress the VLBI line action. *Vice versa*, the presence of command line option `-v` makes `visgen` treat the array as localized, or non-VLBI.

The only difference in processing VLBI and non-VLBI arrays in the new `visgen` version is in the baseline checking. For the VLBI, or global, arrays the array convex hull is not calculated, therefore all the baselines are analysed before the simulation. Thus, to check not only the baselines for the

peripheral stations, but all the possible baselines, simply add option `-v` to the command line.

3.5 Diagnostics and recommendations

If a specific simulation did not pass the baseline check, `visgen` issues the recommendations how to correct the conditions for the simulation could be continued. A particular example of the diagnostics and recommendations is shown below:

```
CHECK_BASELINE: 11 bad baselines found
CHECK_BASELINE: umin_len, umax_len = -230157 107140 (meters)
CHECK_BASELINE: vmin_len, vmax_len = -277383 -28364 (meters)
CHECK_BASELINE: umin, umax = -107493 50039 (wl, at 140.016 MHz)
CHECK_BASELINE: vmin, vmax = -129550 -13247.2 (wl, at 140.016 MHz)
CHECK_BASELINE: Absolute maximum dimensions along u or v:
                  uvmax = 129549 (wavelenghts)
CHECK_BASELINE: The longest baseline is 136363 (wavelengths)
CHECK_BASELINE: The array resolution is 7.33336e-06 (radians)
                  or 0.000420184 (degrees) at frequency 140.112 MHz
CHECK_BASELINE:
CHECK_BASELINE: Recommendations:
CHECK_BASELINE: In order to continue simulation with this array,
CHECK_BASELINE: EITHER increase the patameter 'Visibility grid x-log2N'
                  in file
CHECK_BASELINE: 'test01_Description.txt' from current 10 to 11,
CHECK_BASELINE: and then run LOsim and MAPS_im2uv programs,
CHECK_BASELINE: OR decrease the parameters FOV_size in file
CHECK_BASELINE: 'obs_spec_test01_gha' from current 1000 down
                  to 815.22 (arcseconds) or less.
```

4 How to try the new version

The improved versions of the MAPS package are currently being contained in a repository separate from that of the standard MAPS. To try the new version, we recommend the following sequence.

1. Install the packages `matplotlib` and `ipython` using the means of your

operating system.

2. Temporarily rename your working copy (we assume it has the name MAPS; otherwise use your own name):
   ```
   $ mv MAPS MAPS_orig
   ```

3. Checkout the new version from the repository:
   ```
   $svn checkout svn+ssh://mwa-lfd.haystack.mit.edu/svn/MAPS2/mapscorr
   MAPS
   ```

4. Try to run your tasks and see the output. We also recommend to try the test01 on the SKA variant with 1964 antennae:

   ```
   $ cd MAPS/test/test01
   $ LOsim test01 > /dev/null
   $ MAPS_im2uv -i test01_Brightness.fts -o test01_Visibility.dat \
          -n 0.0304617
   $ visgen -n test01 -s SKA_1 \
          -A $SIM/array/ska1964_star_clumps.txt \
          -G test01_Visibility.dat -V obs_spec_test01_gha -N -m 0
   ```

5. Visualize the array and its convex hull:
   ```
   $ plot_hull.py
   ```

6. After you have finished testing the new MAPS version, remove it:
   ```
   $ rm -fr MAPS
   ```

7. Restore your MAPS version:
   ```
   $ mv MAPS_orig MAPS
   ```