ctools - Feature #1853

Create a tool or script that simulates a plausible zenith angle distribution for a list of pointings

09/28/2016 12:48 PM - Knödlseder Jürgen

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

In the context of the preparation of the first CTA Data Challenge, a tool or script is needed that simulates for a given list of pointings a plausible zenith angle distribution and that assigns appropriate response function for each pointing

Related issues:

Related to ctools - Feature # 1632: Create ctpntsim tool	**New**	
Related to ctools - Feature # 1633: Create ctirfsim tool	**New**	
Related to ctools - Feature # 1115: Create pointing simulation tool	**New**	**01/29/2014**

History

#1 - 09/28/2016 12:49 PM - Knödlseder Jürgen

- Related to Feature #1632: Create ctpntsim tool added

#2 - 09/28/2016 12:49 PM - Knödlseder Jürgen

- Related to Feature #1633: Create ctirfsim tool added

#3 - 09/28/2016 12:49 PM - Knödlseder Jürgen

- Related to Feature #1115: Create pointing simulation tool added

#4 - 09/28/2016 01:09 PM - Knödlseder Jürgen

So far we have two preparatory scripts that could be extended to deal with the zenith angle dependence:

- examples/make_pointings.py
  - script that generates for some KSPs the pointing sequence and that attributes North and South IRFs
  - generates input ASCII file to csobsdef
- cscripts/csobsdef
  - converts input ASCII file into an observation definition XML file
  - to be used as input to ctobssim for event list generation

A number of questions need to be answered:

Where to implement the zenith angle computation?

make_pointings.py is an example script and not a proper cscript, and one may argue that the example script should be limited to defining the pointings and the duration per pointing. However, the actual script assigns the response function of either the North or the South array. This makes sense as the arrays which are to be used are part of the KSP specification.

The csobsdef script just takes the IRF information provided in the ASCII file generated by make_pointings.py and forwards this information to the XML file.

We could enhance csobsdef and implement the zenith angle simulation in the csobsdef script, which would mean that no new script is actually needed to accomplish the task. The csobsdef script would then take the geographical location of the array and compute from this *somehow* (see below) an appropriate zenith angle, and based on that zenith angle, assigns the appropriate response function and writes the response function information into the XML file.

This would mean that for a KSP that involves both CTA arrays two csobsdef runs would be needed, and an additional csobsmerge (similar to csmodelmerge) would be needed to merge the Southern and the Northern XML files.

How to implement the zenith angle simulation?

I think the following options exists:

- implement a scheduler
- draw from a plausible distribution of zenith angles as function of celestial position
- use a typical zenith angle as function of celestial position

Implementing a scheduler is probably a complicated task, but could be envisioned. For the second option one would need to define what plausible means. Distributions could be based on zenith angle distributions for existing observatories (MAGIC for North, H.E.S.S. for South). Using a typical zenith angle is probably the simplest option (could be used as starting point).

How to set the appropriate IRF

I think there are two options:

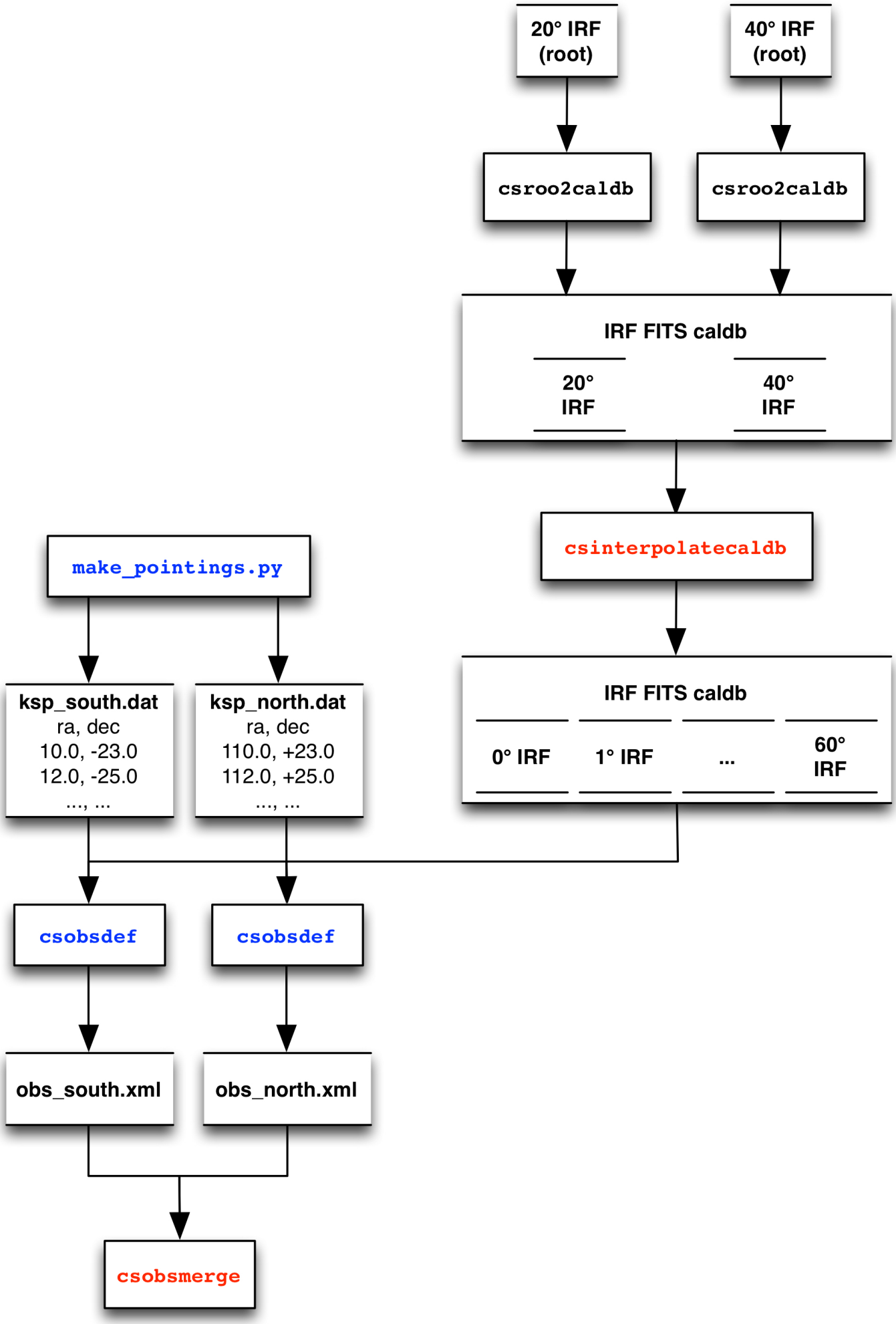
- chose nearest zenith angle IRF (20° or 40°)
- compute intermediate IRFs by interpolation

The first option would lead to a pretty coarse zenith angle dependence. The second option would be more realistic, but if we do this we have to find out how to compute the intermediate zenith angles from the 20° and 40° IRFs. If the second option is chosen, the computation of intermediate IRFs should be probably done by a separate tool, meaning that the problem falls back to choosing the nearest IRF from a now finely sampled IRF grid as function of zenith angle.

#5 - 09/28/2016 02:50 PM - Knödlseder Jürgen

- File zenith-angles.jpg added

Here a possible scheme for implementation (new scripts in red, scripts that need to be modified in blue):


20° IRF
(root)
40° IRF
(root)
csroo2caldb
csroo2caldb
IRF FITS caldb
20°
IRF
40°
IRF
csinterpolatecaldb
make_pointings.py
IRF FITS caldb
0° IRF
1° IRF
...
60°
IRF
ksp_south.dat
ra, dec
10.0, -23.0
12.0, -25.0
..., ...
ksp_north.dat
ra, dec
110.0, +23.0
112.0, +25.0
..., ...
csobsdef
csobsdef
obs_south.xml
obs_north.xml
csobsmerge

#6 - 09/28/2016 04:09 PM - Knödlseder Jürgen

One way to deal with the zenith angle computation could be to introduce a visibility cube which is a cube spanned by Right Ascension, Declination and zenith angle, and which gives for a given array location the numbers of hours per year during which a given celestial direction can be seen under a given zenith angle. In that way we would have full flexibility on how the cube will be computed, and csobsdef can simply use for a given celestial pointing direction the corresponding zenith angle profile to infer a typical zenith angle (or something else). Such a cube would also be very useful for observation planning. We could even have cubes derived from observations logs.

This means that a script csviscube (or something like that) would be needed to compute a visibility cube for each array in form of a FITS file. On input, the script would take

- geographic array longitude
- geographic array latitude
- minimum angular distance of sun below horizon
- minimum angular distance from moon (or something that encodes the brightness of the moon, taking care of moon phases)

csviscube would then compute from these quantities the visibility cube.

#7 - 09/28/2016 04:48 PM - Knödlseder Jürgen

- File viscube.fits.gz added

#8 - 09/29/2016 09:20 AM - Knödlseder Jürgen

- File viscube_north.fits.gz added

- File viscube_south.fits.gz added

Attached some very preliminary visibility cubes for CTA North and CTA South (do not include any moon constraints):
attachment:viscube_north.fits.gz
attachment:viscube_south.fits.gz

#9 - 09/29/2016 09:20 AM - Knödlseder Jürgen

- File deleted (viscube.fits.gz)

#10 - 09/29/2016 04:00 PM - Knödlseder Jürgen

- File North_decs.png added

- File North_min_zenith.png added

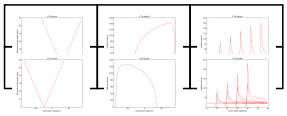
- File North_zeniths.png added

- File South_decs.png added

- File South_min_zenith.png added

- File South_zeniths.png added

Maybe this is trivial for some (in particular the left column), but here a number of plots extracted from the visibility cubes:



#11 - 09/29/2016 05:08 PM - Knödlseder Jürgen

The above plot (left column) suggests that we may as a simple model implement a zenith angle dependence that is based on the declination of the pointing:

```
zenith = |declination - geographic latitude|
```

Of course, this would be the *best* observation scheduling possible. Reality would be worse.

#12 - 10/04/2016 09:44 AM - Knödlseder Jürgen

- File zenithangle.py added

- File show_zenithangle.py added

#13 - 10/04/2016 04:19 PM - Knödlseder Jürgen

I looked a bit into how the zenith angle information could be managed by the CALDB.

While writing the zenith angle information in the CAL_CBD column of the index file is easy (and actually now done by csroot2caldb) the information can currently not be exploited because GCaldb does not provide any method to search the calibration database, return information about available IRFs and allow multiple selection criteria to be combined. Since it is not clear whether GCaldb will be used in the long run for CTA, it is maybe not worth to add such methods now for the needs of CTA.

I therefore decided that it is easier to encode the zenith angle selection in the names of the IRFs. This makes also sense since all tools only take the IRF name and not any other information. This means that the zenith angle names are not extracted from the CALDB, but hard coded. This should of course not be done in a cscript, but doing this in make_pointings.py seems ok.

For the moment, the zenith angle is computed from the simple formula

```
zenith = |declination - geographic latitude|
```

for each declination.

We may still think of using a visibility cube, and having a cscript that computes the visibility cube is still very valuable.

#14 - 10/05/2016 05:48 PM - Knödlseder Jürgen

- Status changed from New to In Progress

- Assigned To set to Knödlseder Jürgen

- % Done changed from 0 to 80

I added a csviscube script that computes the visibility cube using a Sun constraint.

Still need to add a Moon constraint and to handle time intervals that are shorter than a day.

#15 - 03/03/2017 10:36 AM - Knödlseder Jürgen

- Target version changed from 1.2.0 to 1.3.0

#16 - 06/07/2017 05:46 PM - Knödlseder Jürgen

- Target version changed from 1.3.0 to 1.4.0

#17 - 08/01/2017 09:51 AM - Knödlseder Jürgen

- Target version deleted (1.4.0)

- Start date deleted (09/28/2016)

#18 - 11/14/2019 03:09 PM - Knödlseder Jürgen

- Status changed from In Progress to Closed

- % Done changed from 80 to 100

The script was added a long time ago, hence I close the issue. There is now another issue for the addition of the Moon constraint (#3065).

Files

zenith-angles.jpg	673 KB	09/28/2016	Knödlseder Jürgen
viscube_north.fits.gz	5.04 MB	09/29/2016	Knödlseder Jürgen
viscube_south.fits.gz	4.83 MB	09/29/2016	Knödlseder Jürgen
North_decs.png	31.4 KB	09/29/2016	Knödlseder Jürgen
North_min_zenith.png	31.6 KB	09/29/2016	Knödlseder Jürgen
North_zeniths.png	46.1 KB	09/29/2016	Knödlseder Jürgen
South_decs.png	30.5 KB	09/29/2016	Knödlseder Jürgen
South_min_zenith.png	32 KB	09/29/2016	Knödlseder Jürgen
South_zeniths.png	50.6 KB	09/29/2016	Knödlseder Jürgen
zenithangle.py	11.4 KB	10/04/2016	Knödlseder Jürgen
show_zenithangle.py	4.53 KB	10/04/2016	Knödlseder Jürgen