GammaLib - Action #1113

Action # 1122 (New): Calculate IRFs for GSkyRegions

Use pointing table from a FITS file in GCTAPointing

01/28/2014 05:16 PM - Kosack Karl

Status:	Rejected	**Start date:**	01/28/2014
Priority:	Normal	**Due date:**	
Assigned To:	Kosack Karl	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

GCTAPointing should be able to read a table of START,STOP,ALT_PNT,AZ_PNT values, and store them internally in an interpolated list (a GNodesArray). Then, methods to get the pointing altitude (or zenith angle) and azumuth should be added that take a time as an input. They return the interpolated output.

This way, no RA/Dec to alt/az conversion needs to be done in ctools, the values are simply pre-computed in a housekeeping file.

This code can later be used in the ON/OFF analysis code to compute e.f. the average effective area over GSkyRegion that moves in alt/az as the telescope tracks during a run.

Related issues:

Related to ctools - Feature # 1115: Create pointing simulation tool	**New**	**01/29/2014**
Related to GammaLib - Change request # 1176: Introduce GCTAPointings containe...	**New**	**03/13/2014**

History

#1 - 01/28/2014 05:21 PM - Knödlseder Jürgen

We were just discussing this point also with Rolf. So do you think that a pointing table will be a standard CTA data product? Can you recall what the format should be? Would this also include deadtime as function of time?

Maybe we should start with writing a Python script that generates such a table. We could also have a ctool (something like ctpntsim) to simulate first a pointing table, and then use this as input to ctobssim for the simulations. ctpntsim would deal ultimately with observing constraints. Separating both steps is also useful for when we have real data. We can then use real pointing files to generate simulated data accordingly.

#2 - 01/29/2014 01:41 AM - Knödlseder Jürgen

- Project changed from ctools to GammaLib

- Target version deleted (2nd coding sprint)

Moved to GammaLib project as the implementation of this functionality will be in the library.

On the ctools side we need an executable that allows to simulate pointing lists. I created issue #1115 for this.

#3 - 01/29/2014 01:45 AM - Knödlseder Jürgen

- Target version set to 2nd coding sprint

We may create a class named GCTAPointings for this, although this class would not only be a simple container of GCTAPointing objects, but it would also allow to interpolate between pointings. Still, the class would be a logical container of GCTAPointing objects, and it may well be constructed as such.

It's not clear whether a simple GNodeArray can be used for interpolation, as this provides only linear interpolation, but when you interpolate between two pointings I guess you'd like to take the great circle on the sphere through these points and determine the respective intermediate point. Probably, GNodeArray could be used after appropriate coordinate transformations, but maybe there is a more clever way to move from point to another on a

great circle ...

#4 - 01/29/2014 01:48 AM - Knödlseder Jürgen

Kosack Karl wrote:

> GCTAPointing should be able to read a table of START,STOP,ALT_PNT,AZ_PNT values, and store them internally in an interpolated list (a GNodesArray). Then, methods to get the pointing altitude (or zenith angle) and azumuth should be added that take a time as an input. They return the interpolated output.

Would the table contain START and STOP, or just a TIME (pointing at a given time stamp)? If you want to take provision for drift scans, for example, I would expect that the pointing table contains pointing information for a given time (kind of snapshots), and that the code the takes provision for interpolating between the time stamps.

#5 - 01/29/2014 09:58 AM - Kosack Karl

I specified start+stop times only because these quantities are usually averages over a bin in time, and that makes the bin size explicit. One of the data products from the data reduction pipelines will be a housekeeping file, which is a set of tables in a similar format: START, START, <values>, that are "summarized" quantities from various instuments. The quantities are generally averaged in time between START and STOP, and thus can contain variances as well. Examples would be:

- array pointing summary: START, STOP, ALT, AZ (the pointing mode:track,fixed, slew is in the eventlist header). Deviations, etc, may also be stored in columns here
- trigger summary: START,STOP,RATE_AVG, RATE_VAR, DEADTIME_AVG, DEADTIME_VAR, etc.
- weather summary: START,STOP, TEMP_AVG, TEMP_VAR, PRES_AVG, PRES_VAR, etc.
- ...

All quantities can be used for making GTIs or for tracking the status of the array. The exact format has not been decided, but it will be something like what is shown above. The time bins for each table may be different, hence they are each in a separate HDU rather than one big table.

#6 - 01/29/2014 10:08 AM - Kosack Karl

I also don't quite think we need a GCTAPointings container, since this really still describes one pointing (e.g. a fixed point in RA/Dec), but when expressed in ALT/AZ coordinates you move in time. So I would say it's still just a method of GCTAPointing, like GCTAPointing::altaz(time).

Generally, once should also ask for a time for the RA and dec coordinates. In the tracking-mode case, that just returns a fixed ra/dec for any given time. In the fixed-mode pointing, alt/az doesn't change, but RA/Dec does. The only case where I could see a good use for a set of pointings would be a slew-mode.

#7 - 01/29/2014 11:02 AM - Kosack Karl

Proposed steps:

1. add a class GHorizDir, similar to GSkyDir, but representing an Alt/Az (Horizontal) coordinate
2. change the interface of GCTAPointing to take:

   pnt.dir(time); // returns GSkyDir
   pnt.dir_horiz(time); //returns GHorizDir
   pnt.rot(time); // field-rotation matrix at time t

3. modify GCTAObservation to look for a housekeeping file in the XML definition
4. create pointing housekeeping table with START,STOP,ALT,AZ,RA,DEC (any maybe errors)
5. have GCTAPointing load the pointing housekeeping file and create an interpolation function for alt/az and ra/dec over time. The interpolation can be done assuming spherical coordinates with vector quantities (for best accuracy) or just as a simple linear interp (which is probably fine for looking up IRFs in alt/az)
6. use this interpolation function to get pnt.dir(t) or pnt.dir_horiz(t)

#8 - 01/29/2014 04:50 PM - Kosack Karl

- Parent task set to #1122

#9 - 01/30/2014 10:11 AM - Kosack Karl

- Target version deleted (2nd coding sprint)

#10 - 01/30/2014 10:11 AM - Kosack Karl

- Target version set to 2nd coding sprint

#11 - 01/30/2014 10:17 AM - Kosack Karl

- Target version deleted (2nd coding sprint)

#12 - 02/17/2014 10:18 PM - Knödlseder Jürgen

- Target version set to 2nd coding sprint

#13 - 02/20/2014 03:13 PM - Kosack Karl

- Status changed from New to Pull request

- Estimated time set to 0.00

- Position deleted (245)

some of this was partially implemented at the last coding sprint and could go into the devel branch. The branch is "1113-implement-altaz-pointing".

The main changes were:

- Added a GHorizDir class that stores Alt/Az coordinates
- modified GCTAPointing to be able to store a pointing table (a set of points in horizontal coordinates)
- added a method GCTAPointing::load_pointing_table() to load a pointing FITS file and set up an interpolator (currently linear)
- added function GCTAPointing::dir_horiz(const GTime &time) that returns the horizontal pointing direction for a given time by interpolating the table.
- added some test cases

These functions do not interfere with anything else at this point, and are not yet called automatically. In teh future, the ability to load up a pointing table in GCTAObservation should be added, and the functions zenith(), and azimuth() in GCTAPointing should be replaced with the dir_horiz(time) function.

In the future, the interpolation should be done correctly (using 3D position vectors on a sphere), and not linearly between alt/az positions, but the linear method is probably fine if the pointing table is finely spaced, and high accuracy is not needed (generally the alt/az pos is just used to look up the correct value in an IRF table, so only needs to be as accurate as the IRF binning).

#14 - 03/13/2014 12:34 PM - Knödlseder Jürgen

- Status changed from Pull request to Resolved

- % Done changed from 0 to 100

Sorry for being slow with this. I now merged the code into devel.

I did some revision of the source code, renaming for example load_pointing_table to a simple load, as it exists for other classes. I also splitted the load into a load and read method, the latter is directly operating on an existing FITS table. I added also an extname optional parameter to the load method so that the extension name could be changed by the user (as in other classes). Finally, I added a load constructor which allows loading the pointing table upon construction.

Concerning GHorizDir, I was wondering whether we should name this more explicitly GHorizontalDir. Other alternatives would be GAltAzDir. I agree that spherical coordinate classes should be derived from a common class, such as GSphereDir (or equivalent), yet at the end I'm not sure that they would share a lot of code as we want to have the method very efficient to reduce computing time. Before going there, maybe we want to have a full picture of all coordinate systems we need and how they relate, and the flesh out how we should organize them.

#15 - 03/13/2014 01:11 PM - Knödlseder Jürgen

Another thing: conceptually, GCTAPointing is a single pointing, but now it contains an entire table of pointings. It would be better to introduce a pointing container class GCTAPointings to handle pointings as function of time. The GCTAPointings class would then deal with loading and saving of pointing tables, and interpolation, and could have an operator that returns a GCTAPointing object as function of time or index:

```
const GCTAPointing& operator[](const int& index) const;
GCTAPointing&       operator[](const int& index);
GCTAPointing        operator()(const GTime& time) const;
```

I created issue #1176 for this.

#16 - 03/13/2014 03:25 PM - Kosack Karl

It's somewhat of a philosophical question as to whether or not the pointing table refers to one pointing or multiple pointings: I would say it refers to a single pointing.

Do you call a pointing "a fixed position in RA/Dec"? In that case the pointing table is a table of Alt/Az values that corresponds to a single pointing.

If a pointing is called "a fixed position in AltAz", then, each element in the table is a pointing (and then a Fixed RA/Dec is no longer a single pointing either)...

I thought it was a bit simpler thinking of a pointing as a fixed direction on the sky, and the table is just needed to translate that pointing into earth-based coordinates (therefore there is only one pointing).

If you want to be really general (and also allow the case of a drift scan where Alt/Az is fixed, or a slew scan where everything changes), than a concept of Pointings could be introduced, however you then need to differenciate between RA/Dec and Horizontal pointings.

#17 - 03/13/2014 03:36 PM - Kosack Karl

actually, thinking about it more, I do like your GCTAPointings class, but I was confusing it with just a container of GCTAPointings (which it isnt: it's more of a factory class that genreates a GCTAPointing given a time). It may give the same GCTAPointing over and over again, with only the horizontal coordinate changed, for example. That does work generally, and there is no longer a concept of tracking/drift/slew, etc.

It might be nice to give it a better name Than GCTAPointings though, which to me implies a container, not an interpolator. GCTAPointing is the return value, and is an **instantaneous** pointing direction. Maybe GCTAPointingEvolution or something like that for the class that describes the time-based pointing, rather than the instantaneous one? or you could use GCTATracking for the collection, and GCTAPointing for the instantaneous value (since tracking implies a change in time)

GCTATracking::get_pointing(time); // returns a GCTAPointing given a time
CCTATracking::get_mode(); // returns GCTATracking::FIXED_RADEC, ::FIXED_ALTAZ, ::SLEW?

#18 - 03/13/2014 11:26 PM - Knödlseder Jürgen

Yes, my idea was that GCTAPointing is an **instantaneous** pointing direction, while GCTAPointings is a collection of pointing directions. I guess it still could act as a container (so that it can also be used to simply create or concatenate point tables), while it will have interpolation methods that allow to derive interpolated **instantaneous** pointing directions. In this sense it would be more a factory, but is this conceptually bad that a container has factory-like methods?

#19 - 03/14/2014 03:47 PM - Kosack Karl

Ok, I'm happy with it then. It was just a question of terminology: normally when we say there are multiple "pointings", we mean multiple fixed-positions on the sky (e.g. the galactic center, the crab nebula, etc). During a pointing, the telescopes track in some way. So normally, for a single observation there is a single pointing, but during that pointing the tracking position may change in either alt/az, ra/dec, or both in the case of slewing. I think if it's well documented what you mean by GCTAPointings, its ok, but just by reading the name of the class, I would expect it to be a list of fields of view, not the instantaneous position during an observation. Again, it's another case where we need a good CTA dictionary to define terms...

#20 - 07/19/2014 02:10 AM - Knödlseder Jürgen

- Target version deleted (2nd coding sprint)

#21 - 07/20/2014 11:25 PM - Knödlseder Jürgen

- Status changed from Resolved to Rejected