GammaLib - Feature #1732

Work out implications for changing DETX and DETY from sky coordinates to instrument coordinates

03/04/2016 02:01 PM - Knödlseder Jürgen

Status:	Feedback	**Start date:**	03/04/2016
Priority:	Normal	**Due date:**	
Assigned To:		**% Done:**	90%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

So far DETX and DETY are given in a sky coordinate system. Current Cherenkov Telescopes use DETX and DETY in an instrument system, aligned with altitude and azimuth. In case we want to switch to such a system, the CTA module would need to be modified. Before doing so we should make a list of things that would need to be changed. One obvious change is that time becomes an explicit parameter of the pointing (implying probably that a pointing table is needed).

This feature is for collecting ideas and the implications for such a change.

History

#1 - 10/26/2016 11:13 AM - Mayer Michael

- Status changed from New to Feedback

- Assigned To set to Mayer Michael

- Target version set to 1.2.0

- % Done changed from 0 to 90

I have given some more thoughts to this topic during the past weeks.

In principle, the instrument system and sky system are only different by a time-dependent rotation angle between the coordinate system axes (and a tangential projection). This rotation angle, however, can be simply inferred from the event list, provided that we have DETX and DETY information available. The following code inside GCTAPointing would do the job.

```
// Compute position angle in sky system
double posang  = m_dir.posang(atom->dir().dir()); //m_dir is pointing direction, Atom is @GCTAEventAtom@ pointer

// Compute rotation angle in instrument system
double instang = std::atan2(atom->dir().dety(), atom->dir().detx());

// Compute relative rotation angle between two systems
double rot_angle = posang - instang;

// Map rotation angles into continuous range
if (rot_angle < -gammalib::pi) {
   rot_angle += gammalib::twopi;
}
else if (rot_angle > gammalib::pi) {
   rot_angle -= gammalib::twopi;
}
```

I have played around with the code and came up with a solution how we could incorporate this without affecting the current scheme. It all boils down to computing the rotation angle for each event after reading in the pointing information. In GCTAPointing, I added a function compute_rotation_angles(GCTAEventList*) that is called from within GCTAObservation::read() (after reading the events). The angles are stored in a vector that can be used to interpolate between event times (stored in a protected GNodeArray in the pointing object). This way we can easily access and compute the rotation angle between the two coordinate systems for each point in time during the observation.
I have thus enhanced the functions GCTAPointing::instdir() and GCTAPointing::skydir() by additionally taking a GTime argument. In e.g. GCTAIrfBackground::eval, we then use the event time to compute the correct intstrument coordinates. The other way round, this feature then also enables us to add a time loop in GCTACubeBackground::fill() (which is used by ctbkgcube). Accordingly, we can account for the time-dependent rotation between the two coordinate systems also in a binned analysis.

Here a list of the changes that come with the above solution:

- Modify GCTACubeBackground::fill to take the additional argument delta_rot_angle (default = 5.0 degrees)

- Add GCTAEventList::has_detxy() to check if DETX/DETY were loaded
- Add GCTAPointing::compute_rotation_angles(GCTAEVentList*)
- Add GCTAPointing::rotation_range() returning the range of rotation angle for the parent observations
- Add GCTAPointing::rotation_angle(GTime) returning the rotation angle for a given point in time via interpolation
- Modify GCTAIrfBackground::eval() to use the event time to compute the correct instrument coordinates
- Modify GCTAObservation::read() to call m_pointing.compute_rotation_angles() after events have been read.

The last point can probably be handled better (i.e. to support in memory pipelines without IO). However, for now ctobssim doesn't simulate a changing rotation angle (it is always zero). For analysing real data the function GCTAObservation::read is called anyway.

The modifications are open for discussion on branch *1732-detxy-rotation-angles*.

Note that I also modified ctbkgcube by adding a hidden parameter drotangle. That allows to specify the change in rotation angle to make a new time bin in GCTACubeBackground::fill. These changes are in ctools on branch *1732-rotation-angle-in-ctbkgcube*

#2 - 10/26/2016 02:21 PM - Mayer Michael

By the way, I also fixed an issue that caused event list with different MJDREF (e.g. from HESS) to be loaded incorrectly:
in GCTAEventList::read, the function@read_events@ was called before the GTIs were loaded implying that always the standard CTA reference time was used. I changed the order such that GTIs are loaded first. There was a comment in the code that this had to be done her (for some unknown reason). It seems however my change didnt cause any problem.

#3 - 11/15/2016 11:39 AM - Mayer Michael

Are there any thoughts/comments to the above idea?

#4 - 11/17/2016 04:12 PM - Knödlseder Jürgen

user#77 wrote:

> Are there any thoughts/comments to the above idea?

Sorry for the slow reply.

This looks like an elegant solution.

I was just wondering whether something would need to be done also for the Npred computation, since there, no events would be available, hence the principle will probably not work.

I'm also wondering about the way you changed the GCTAPointing class: for the moment the concept is that an instance of GCTAPointing corresponds to a given point in time, hence GCTAPointing methods should in principle not be time dependent. Your implementation may be a good workaround for the moment, but I'm wondering whether this does not point to the fact that we should overthink the current scheme (and maybe meaning of GCTAPointing).

I will look into the changes you propose and will think a bit about this.

#5 - 11/17/2016 05:00 PM - Knödlseder Jürgen

I looked into GCTAModelIrfBackground. As far as I can see there is in fact no issue in Npred. In case that you are sure that DETX and DETY are actually filled for each event in detector coordinates, it looks to me that the pointing conversion is in fact not needed, am I right?

In that case, I would rather make sure that every event has DETX and DETY in detector coordinates and not sky coordinates, which I guess would solve the issue without touching the structure of GPointing.

For GCTACubeBackground, I was wondering whether the cube should not be a function of DETX and DETY instead of RA and DEC, which probably would also solve the issue.

#6 - 11/22/2016 01:03 PM - Mayer Michael

Thanks for your feedback.

> I'm also wondering about the way you changed the GCTAPointing class: for the moment the concept is that an instance of GCTAPointing corresponds to a given point in time, hence GCTAPointing methods should in principle not be time dependent. Your implementation may be a good workaround for the moment, but I'm wondering whether this does not point to the fact that we should overthink the current scheme (and maybe meaning of GCTAPointing).

I certainly agree. I realised while adding the changes that we already have some infrastructure to support pointing tables. The question is if we understand a GPointing object as one direction at a given point in time (i.e. having the attributes GTime, GHorizDir and GSkyDir) like you described. I think from the CTA perspective, it makes sense to have an additional pointing sequence class, maybe a container holding GPointing objects. This container might then be able to interpolate in time and provide the functionality to convert from sky to instrument coordinates at a given point in time. For the analysis, it is important to know the rotation angle between sky and instrument system at any point in time (especially for stacked analysis when you want to transform the other way round).

> looked into GCTAModelIrfBackground. As far as I can see there is in fact no issue in Npred. In case that you are sure that DETX and DETY are actually filled for each event in detector coordinates, it looks to me that the pointing conversion is in fact not needed, am I right?

For unbinned analysis it is not needed. But to be consistent with binned analysis, it might be good to use the back-and forth transformations throughout the code.

> For GCTACubeBackground, I was wondering whether the cube should not be a function of DETX and DETY instead of RA and DEC, which probably would also solve the issue.

That is in fact what we already have in the IRFs.
I am not sure about that since the whole binned analysis runs in sky cubes. If we had the background cube in detector coordinates, the information how to convert to sky coordinates would be lost. Which sky bin would correspond to which DETX/DETY bin if we stacked hundreds of observations?

#7 - 03/03/2017 10:17 AM - Knödlseder Jürgen

- Target version changed from 1.2.0 to 1.3.0

#8 - 06/06/2017 10:28 PM - Knödlseder Jürgen

- Target version changed from 1.3.0 to 1.4.0

#9 - 07/31/2017 11:08 PM - Knödlseder Jürgen

- Assigned To deleted (Mayer Michael)

- Target version deleted (1.4.0)

#10 - 10/27/2017 10:06 AM - Tiziani Domenico

Alexander and me also gave some thoughts to this topic in the past days.

It could also be a solution to state that the conversion from background data in instrument coordinates to sky coordinates is a step that has take place before the analysis. This would mean that (on the side of the data production) for each IACT observation an individual background template has to be built in which the background is integrated over the observation time. Like this, a RA/Dec-aligned background template could be generated, which would then make any changes to GammaLib unnecessary.