GammaLib - Action #3248

Add rectangular sky regions

06/15/2020 09:43 PM - Specovius Andreas

Status:	Closed	**Start date:**	06/15/2020
Priority:	Normal	**Due date:**	
Assigned To:	Specovius Andreas	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2.0.0		

Description

Adding a rectangular sky region class to the ctools would eventually enable a classical box-style spectral analysis with the ctools.

An example in which such an analysis approach has already successfully been applied is e.g. this https://arxiv.org/pdf/1609.08671.pdf H.E.S.S. publication.

You may also refer to my slides https://indico.cta-observatory.org/event/2604/ from the ctools user call in march 2020.

History

#1 - 06/15/2020 10:19 PM - Specovius Andreas

- File demo.png added

During the call you were open for input, so I present what I did so far:
My implementation of a GSkyRegionRect class can be found in my repo in the branch 3248-sky-region-rectangle.

The implementation stores a centre, a width and a height and a position angle posang (CCW from North).
The rectangle can be read and formatted from/to ds9 format ("box").

The actual rotation is carried out for queried sky directions (e.g. in contains() and overlaps() methods) during a coordinate transformation into a local coordinate system of the rectangle, which simplifies the containment computations.
The rectangle itself is X/Y-aligned.

I implemented a method GSkyRegionMap::set_region_rect() to build WCS region maps for the rectangle.
Currently, a resolution "goal" is defined (0.01deg hardcoded), with which the number of bins along the ra/dec axes are computed. In-turn, a particular resolution per axis is computed and chosen for the skymap resolution:

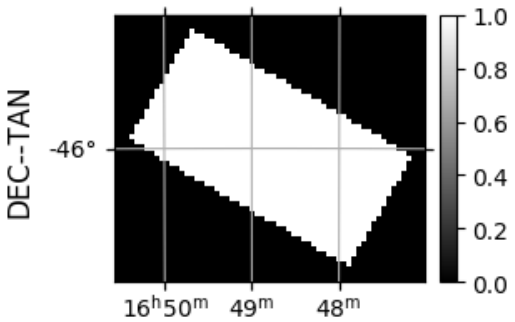
```
int nbins_x = int(map_width  / resolution_goal +0.5);
int nbins_y = int(map_height / resolution_goal +0.5);

double resolution_x = map_width  / nbins_x;
double resolution_y = map_height / nbins_y;
```

map_width and map_height are dynamically computed to obtain an optimal map extension for the particular rotated rectangle.

I wonder what the best choice for the resolution goal is, and whether you see any issues with this approach.

An example of a WCS region map with a resolution of ~0.01 deg produced from a rectangle with w=0.25, h=0.5, PA=60 can be seen in the attached image.



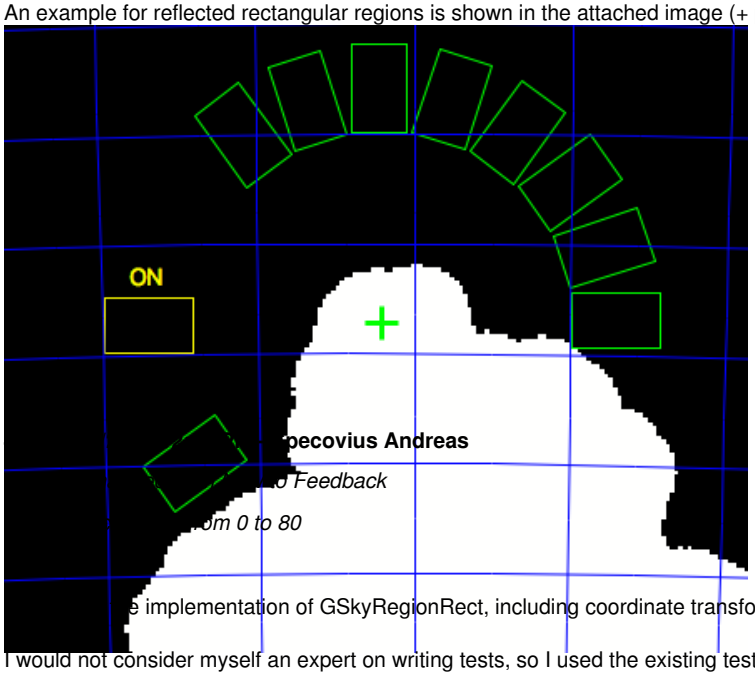
#2 - 06/16/2020 06:55 PM - Specovius Andreas

- File example_refl_reg_excl.png added

I extended csphagen to handle rectangular sky regions:

- Adjusted the parameter file: added width, height and posang and adjusted srcshape to accept "RECT"
- Added csphagen members _reg_width, _reg_height and _reg_posang, adjusted get/set_state() methods accordingly
- Adapted _get_parameters_bkgmethod_reflected() method to query rectangle parameters
- Slightly extended _reflected_regions() method to allow placing rectangular regions
- Outsourced the computation of the regions separation angle (alpha) to new method _compute_region_separation(). The current implementation for rectangles makes use of the corners of the rectangle to select a separation angle which is in some cases of initially rotated rectangles a bit overestimated. An optimised algorithm may yield slightly more reflected regions here.

The code can be found in my ctools repo in the same branch 3248-sky-region-rectangle.
An example for reflected rectangular regions is shown in the attached image (+ pointing, yellow ON, green OFF, white excluded).



e implementation of GSkyRegionRect, including coordinate transformations, and added unit tests.
I would not consider myself an expert on writing tests, so I used the existing tests for the other region types as orientation and hope that the main aspects of GSkyRegionRect are now covered.

I think the implementation is ready for a pull request now.
I would be happy to hear your opinion and whether you see open issues with the implementation before I initiate the pull request.

#4 - 12/02/2020 11:09 PM - Knödlseder Jürgen

- Assigned To set to Specovius Andreas

- Target version set to 2.0.0

Thanks a lot for the implementation of the rectangular sky regions.

I went over GSkyRegionRect and it looks fine. I propose the following adjustments:

- remove posang_deg() and use degrees in posang() in order to simplify the interface
- store internally the width, height and posang in degrees (avoid conversion forth and back)

Are the methods contains_local(), transform_to_local(), transform_to_global() and get_corner() actually needed in the public interface or can they be made private. They look quite low-level.

I also recognised that the code from which you started to implement the class was quite outdate, but I managed to rebase your branch after a number of adjustments.

I merged the code in the devel branch. In case you want to do any further code adjustments, please branch from devel so that you have an up-to-date code base.

I will also look into your changes to csphagen, hopefully tomorrow.

#5 - 12/03/2020 03:36 PM - Knödlseder Jürgen

I also checked csphagen which is okay. I rebased also that code, fixing a number of minor issues. I made the srcshape parameter public, since the user should now explicitly specify the shape of the region (before a hidden parameter made sense since the region was always circular). I adapted the unit tests to cope with that change, and added a unit test for the rectangular sky regions.

I also updated the reference documentation.

In addition, cslightcrv, csphasecrv and csscs make use of csphagen via the obsutils.get_onoff_obs() function. I adapted the function for regular sky regions (basically adding the width, height, and posang parameters) and added these parameters to all three scripts. I also adapted the unit tests.

I will merge the code into devel (at latest tomorrow), if you want to do any further changes, please start from the devel branch.

#6 - 12/04/2020 08:56 AM - Specovius Andreas

Thanks for revising the code!

You are right, I did not rebase the code to the latest version, sorry for the inconvenience!

The intention of internally storing half width/height and posang in radians was to reduce the amount of computations in the containment check method, which will be used much more frequently than setting/getting these parameters.
In case you think, speed is not an issue here, I am totally fine with your suggestions.

Concerning the methods you addressed:
I can imagine situations, where I might make use of the get_corner() method, e.g. for sanity checks. Hence I personally think, that it may be handy to keep this public.
For the contains_local() and transform_to_local/global() methods, I currently don't see a strong argument to keep them in the global scope.
In fact, this was mainly useful for debugging and testing.

#7 - 12/04/2020 10:05 AM - Knödlseder Jürgen

user#291 wrote:

> Thanks for revising the code!
>
> You are right, I did not rebase the code to the latest version, sorry for the inconvenience!
>
> The intention of internally storing half width/height and posang in radians was to reduce the amount of computations in the containment check method, which will be used much more frequently than setting/getting these parameters.
> In case you think, speed is not an issue here, I am totally fine with your suggestions.

I would not be worried about the speed for a single multiplication, it's probably faster than the memory access of the stored member. It's always better to keep the code simple for maintenance reasons.

> Concerning the methods you addressed:
> I can imagine situations, where I might make use of the get_corner() method, e.g. for sanity checks. Hence I personally think, that it may be handy to keep this public.

I have seen that you actually use the method in csphagen, hence it should remain public. Yet I recognised that get_corner() is not really GammaLib style (which does not use get_ or set_ for public methods), hence I propose to rename the method to corner().

> For the contains_local() and transform_to_local/global() methods, I currently don't see a strong argument to keep them in the global scope. In fact, this was mainly useful for debugging and testing.

I think they are not used anywhere, so I would propose to make them private. The advantage is that we don't have to worry about interface changes for private methods, hence my philosophy was always to make only public what is really needed. **I will move the methods to private.**

I think one outstanding issue is to make sure that the overlaps() and contains() method work for all types of regions. I added a bit of code to GSkyRegionCircle but I did not really check it. In principle, we have a matrix of 3x3 region types that we have to check. I have not yet found a clever concept that avoids implementing for each region type a check of all other region types. We should also add unit tests for all possible cases.

Finally, I was also considering of renaming the class GSkyRegionRect to GSkyRegionRectangle which is more explicit, and more compatible with the other region classes GSkyRegionCircle and GSkyRegionMap that fully spell out their type.

#8 - 12/04/2020 10:39 AM - Specovius Andreas

I personally would prefer to keep names short but I definitely see the point of being compliant with the Gammalib naming scheme.

Performing the overlap and containment checks via converting to GSkyRegionMap would definitely reduce the cases to distinguish here.
But this would mean a loss in performance and accuracy, hence I am not sure whether this approach is reasonable.

As mentioned above, I implemented a first, surely incomplete, unit test for GSkyRegionRect, which may serve as a starting point here.

#9 - 12/04/2020 10:50 AM - Knödlseder Jürgen

One other proposal: currently GSkyDir is used as object carrying the local coordinates x and y which is not fully what a sky direction is for. I think a GSkyPixel object would actually be more appropriate for that purpose since it has x() and y() methods for storing the attributes. This would also make the usage safe against mixing local and global coordinates, since they are stored in objects of different types.

#10 - 12/04/2020 01:57 PM - Specovius Andreas

user#3 wrote:

> One other proposal: currently GSkyDir is used as object carrying the local coordinates x and y which is not fully what a sky direction is for. I think a GSkyPixel object would actually be more appropriate for that purpose since it has x() and y() methods for storing the attributes. This would also make the usage safe against mixing local and global coordinates, since they are stored in objects of different types.

Indeed, this caveat is also documented in the methods descriptions.
Switching to GSkyPixel for local coordinates sounds great! (I have to admit, I was not aware of GSkyPixel) I don't see any issue in switching to GSkyPixel. Should be rather easy to change that.

#11 - 12/04/2020 06:28 PM - Knödlseder Jürgen

- % Done changed from 80 to 90

I implemented all changes and merged them into devel.

I complemented the unit tests, now the containment and overlap of all region types is checked. I discovered an error in the overlap computation between circle and rectangle which basically ignored that the overlapping region has roundish corners. This is now corrected.

Maybe you can check if everything works as expected at your side, once you give green light we can close the issue.

#12 - 12/07/2020 01:57 PM - Specovius Andreas

I checked out the latest code and ran a few tests on GSkyRegionRectangle.
I did not encounter any issues smile.png

#13 - 12/07/2020 02:17 PM - Knödlseder Jürgen

- Status changed from Feedback to Closed

- % Done changed from 90 to 100

Great, hence I close the issue now. Thanks again for providing the code.

Files

demo.png	10.4 KB	06/15/2020	Specovius Andreas
example_refl_reg_excl.png	25.7 KB	06/16/2020	Specovius Andreas