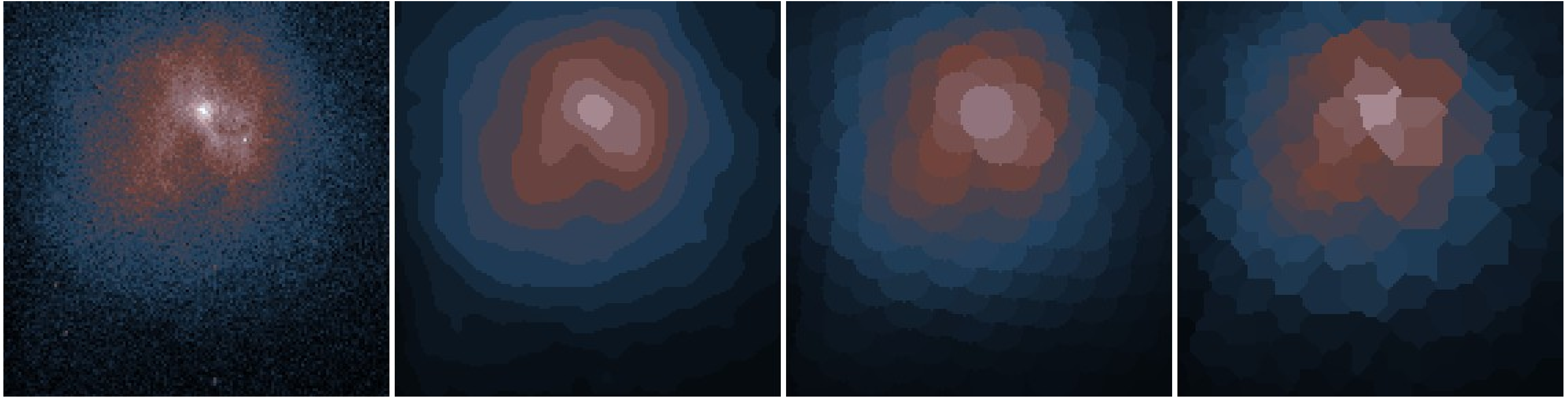
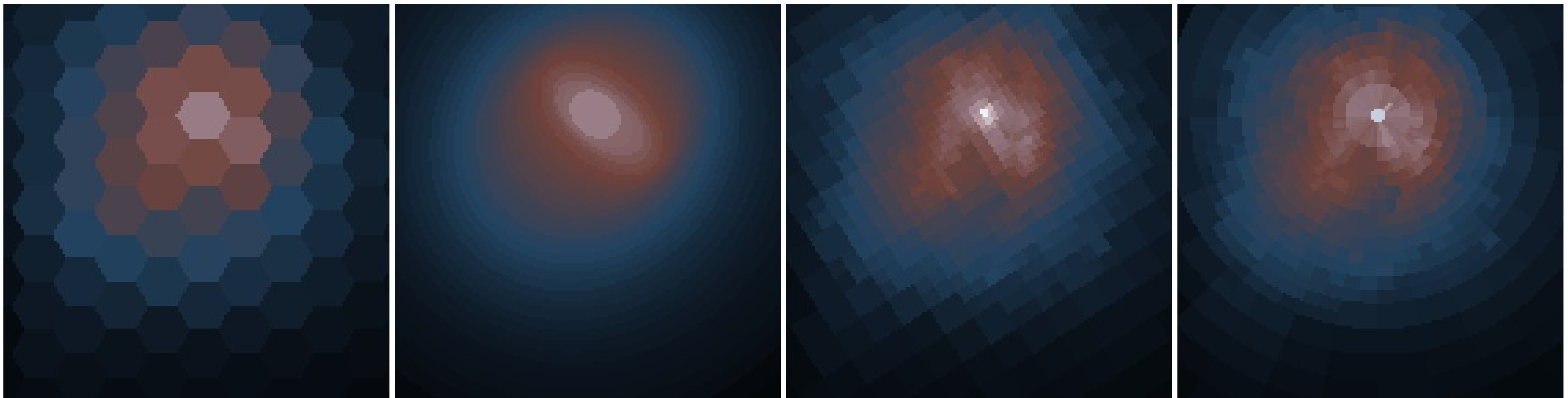


Summary of Adaptive Binning Routines
June 2015, circa CIAO 4.7



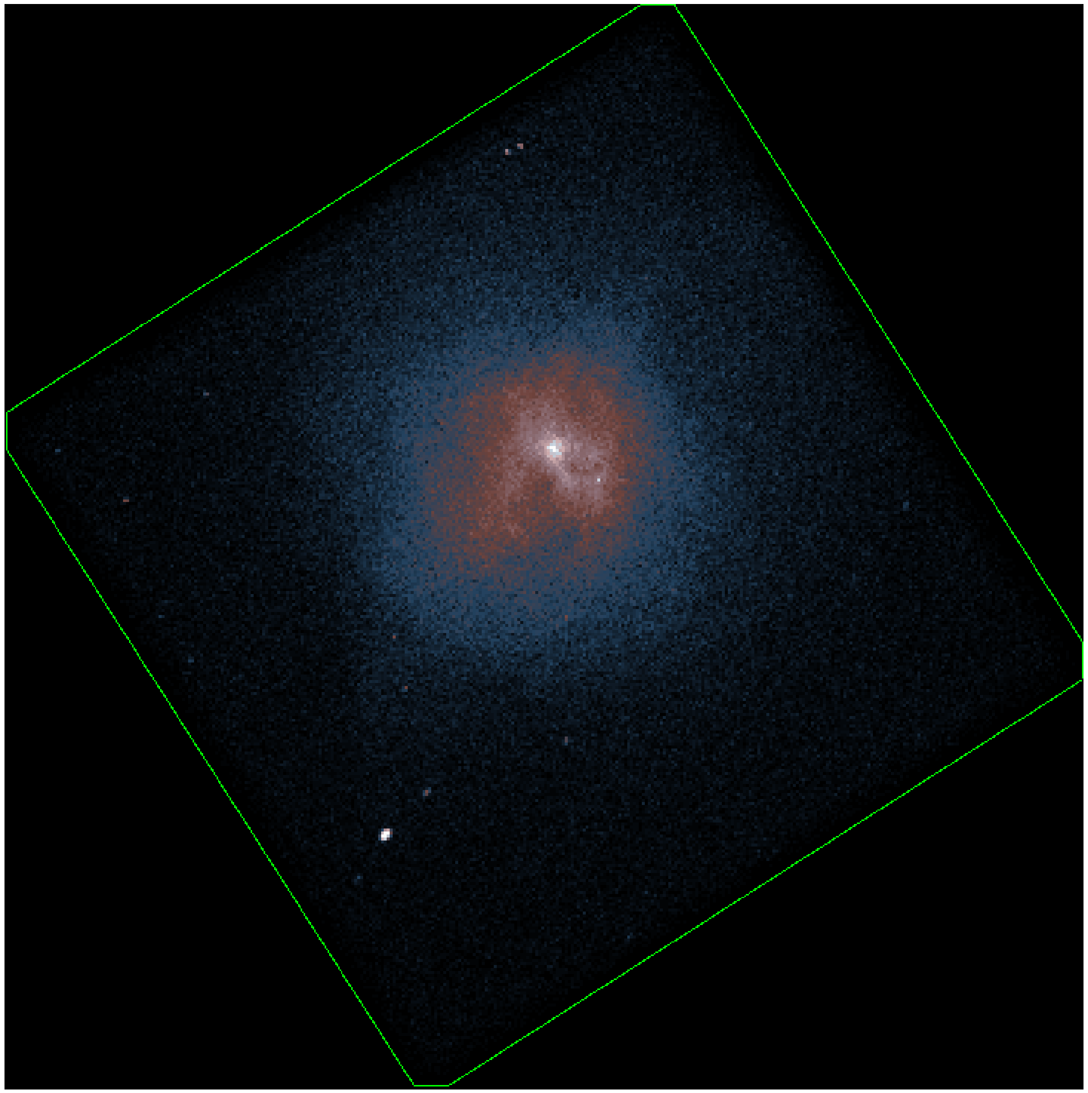
Available

- Released in CIAO
  - dmnautilus : quad-tree adaptive binning
- In Development
  - contour_bin : follows local contours
  - dragon_scales: overlapping shapes
  - grow_from_max : watershed tessellation
  - hexgrid : regular hexagonal grid
  - mkregmap : stack of regions
  - merge_too_small : utility to merge small regions
  - dmnautilus++ : updates to invert SNR logic
  - dmnautilus++++ : proof of concept uses polar grid

Getting started

```
% download_chandra_obsid 9399
% chandra_repro 9399 9399/repro
% apply_fov_limits \
     infile="acisf09399_repro_evt2.fits[ccd_id=7,energy=200:2000]" \
     outfile=img.fits binsize=4 cl+
```

NGC 5044



Counts image of ObsID 9399, NGC 5044, in the 0.2 to 2.0 keV band, ACIS-7 only with field of view shown.

dmnautilus

- Overview

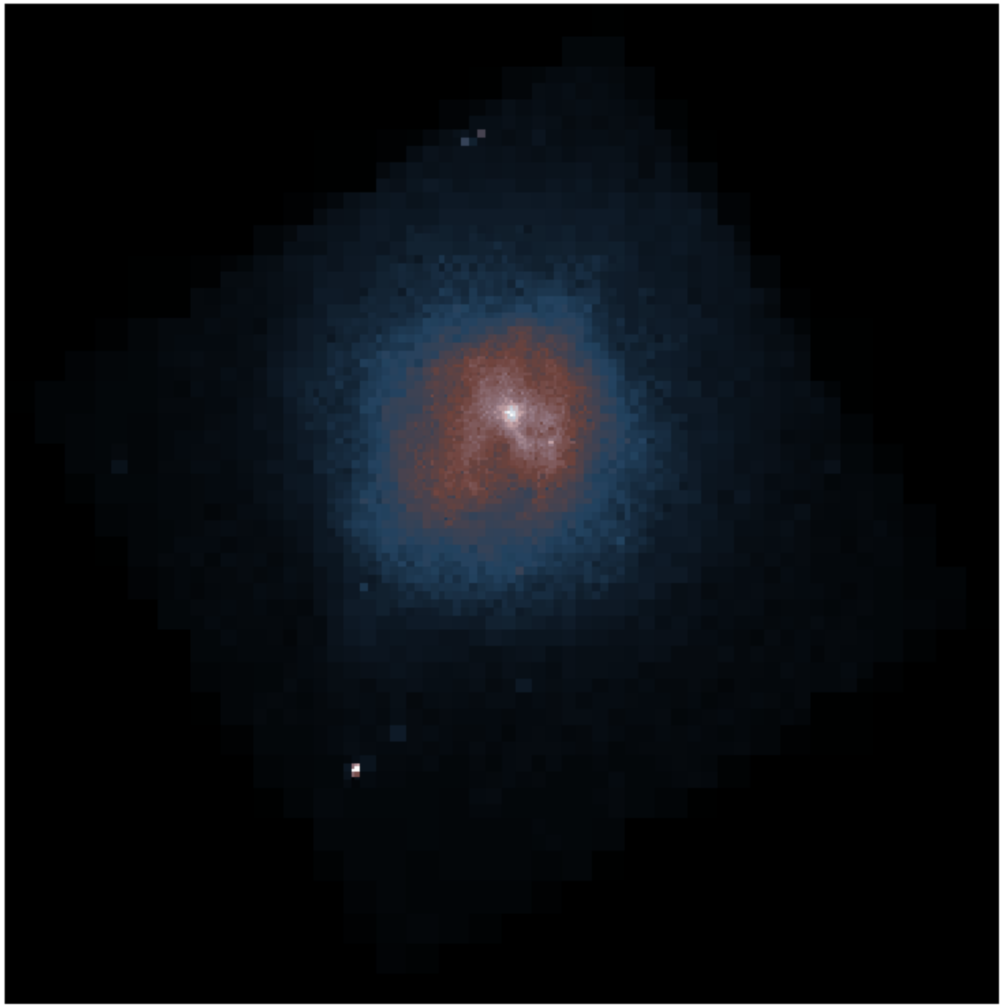
  Compiled tool in CIAO. Implements quad-tree adaptive bin. If the SNR in the image pixel values is above the input threshold, the image is divided into 2x2. Each sub-image is then checked. Algorithm stops when SNR threshold is no longer met and/or single pixel remains.

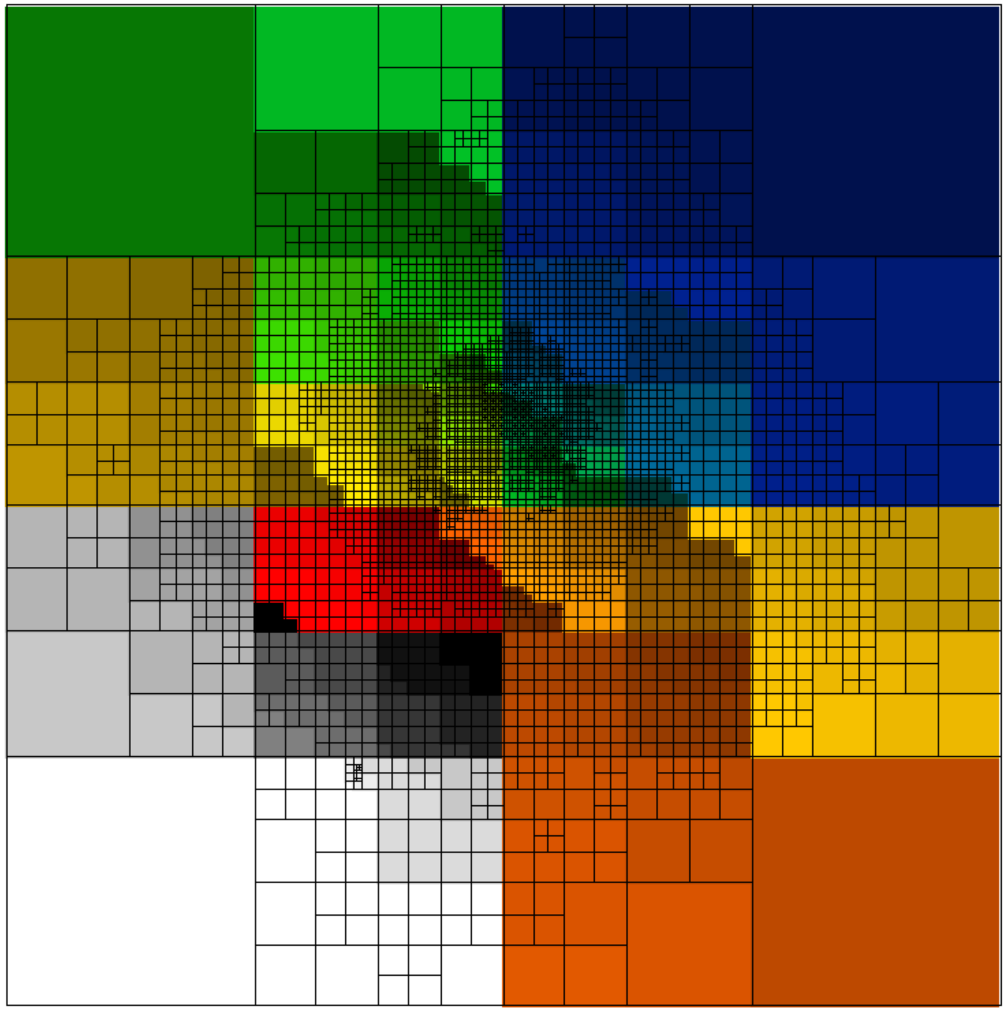
- Example command

  Keep sub-dividing the image until the SNR falls below 15.8 (~300 counts).

```
% dmnautilus img.fits nautilus.img 15.8 outmask=nautilus.map
```

dmnautilus results





(left) adaptively binned image created with dmnautilus with a SNR threshold of 15.8. (right) is the output mask/map file showing which pixels belong to each region.

dmnautilus parameters

Parameter	Description
infile	input counts image
outfile	output adaptively binned image
snr	target SNR limit
inerrfile	input error file; if blank uses sqrt(n)
outmaskfile	map of regions (pixel value is group number/ID)
outsnrfile	SNR achieved in each map region
outareafile	area of each map region

The counts image does not need to be smoothed.

contour_bin

- Overview

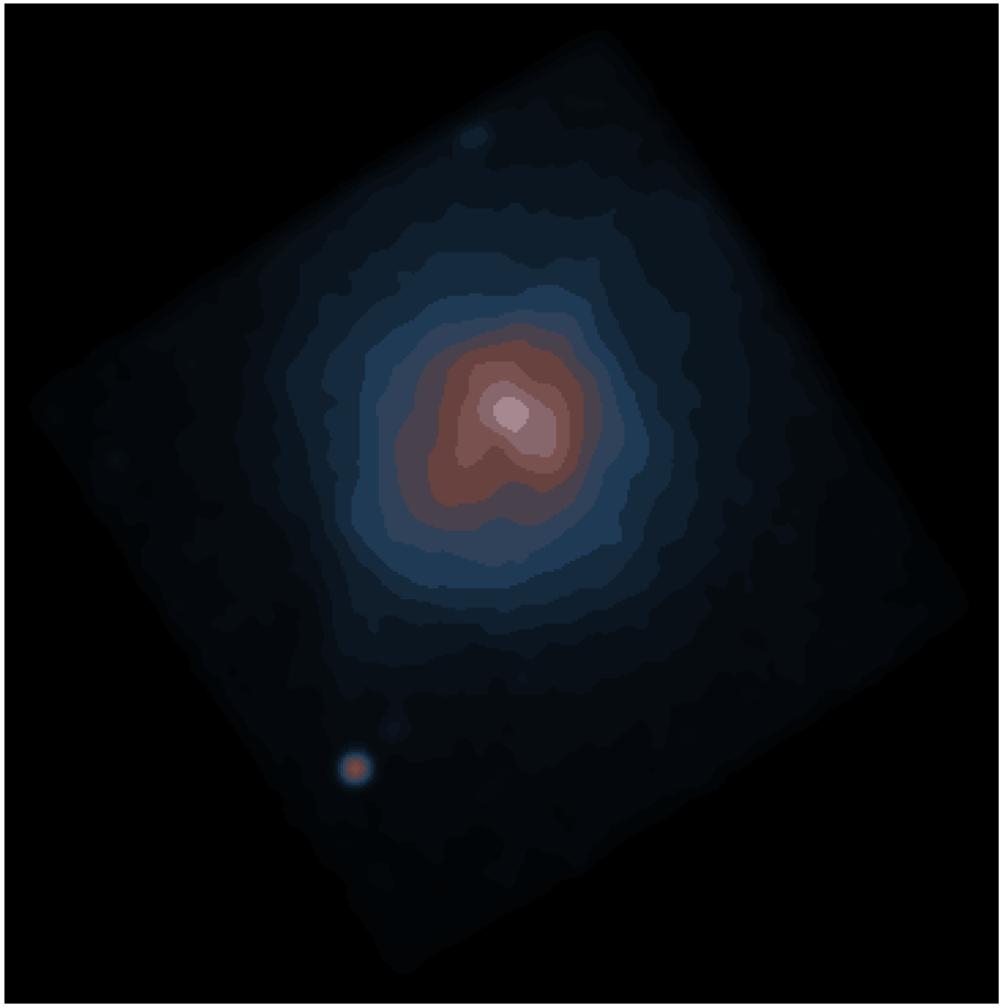
  A nod to Sanders' routine of same name (so yes, need to rename it). Uses dmimglasso to generate a contour around the max value down to next lowest value in the contour grid. Uses a maximum radius criteria to prevent thin, long “fingers”. Repeats until all pixels are included.

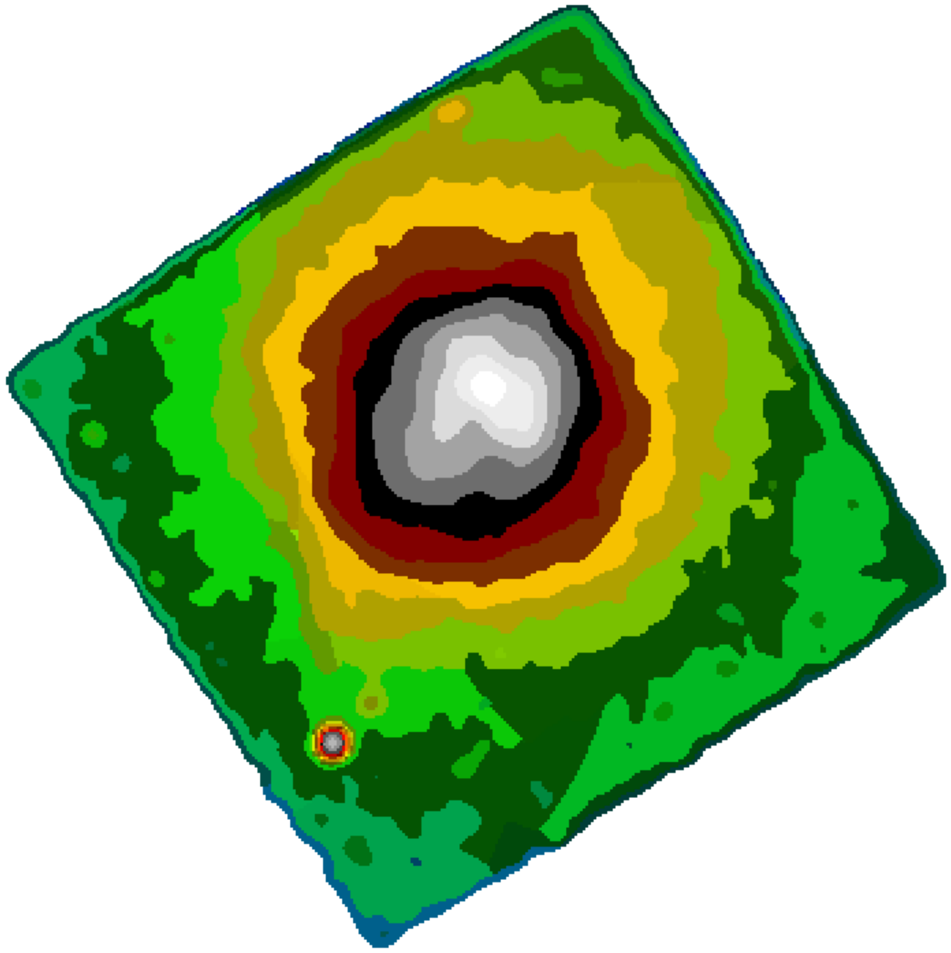
- Example commands

```
% aconvolve img.fits smimg.fits kernels="lib:gaus(2,5,5,3,3)"  \
  method=slide edges=constant const=0 clob+

% contour_bin infile=smimg.fits outfile=cbin.map binimg=cbin.img \
  distance=500 shape=circle levels=100 scale=log maxcontours=200 \
  verbose=2 clobber=yes
```

contour_bin results





(left) adaptively binned smoothed image created with contour_bin using a max radius of 500 pixels, with 100 logarithmically spaced contour levels. (right) the map showing which pixels belong to which group.

contour_bin parameters

Parameter	Description
infile	input counts image
outfile	map of regions (pixel value is group number/ID)
binimg	output adaptively binned image
distance	max distance contour should extend
shape	circle|box, distance measured as radius or offset
levels	number of contour levels between min and max pixel values
scale	linear|log, spacing between contour levels
maxcontours	maximum number of groups to create

Unlike dmnautils, the outfile is the map, not the binned image.

The algorithm can get stuck at the edge making lots of small contours, thus the maxcontours parameter.

The input image should be fairly smooth (no internal smoothing).

dragon_scales

- Overview

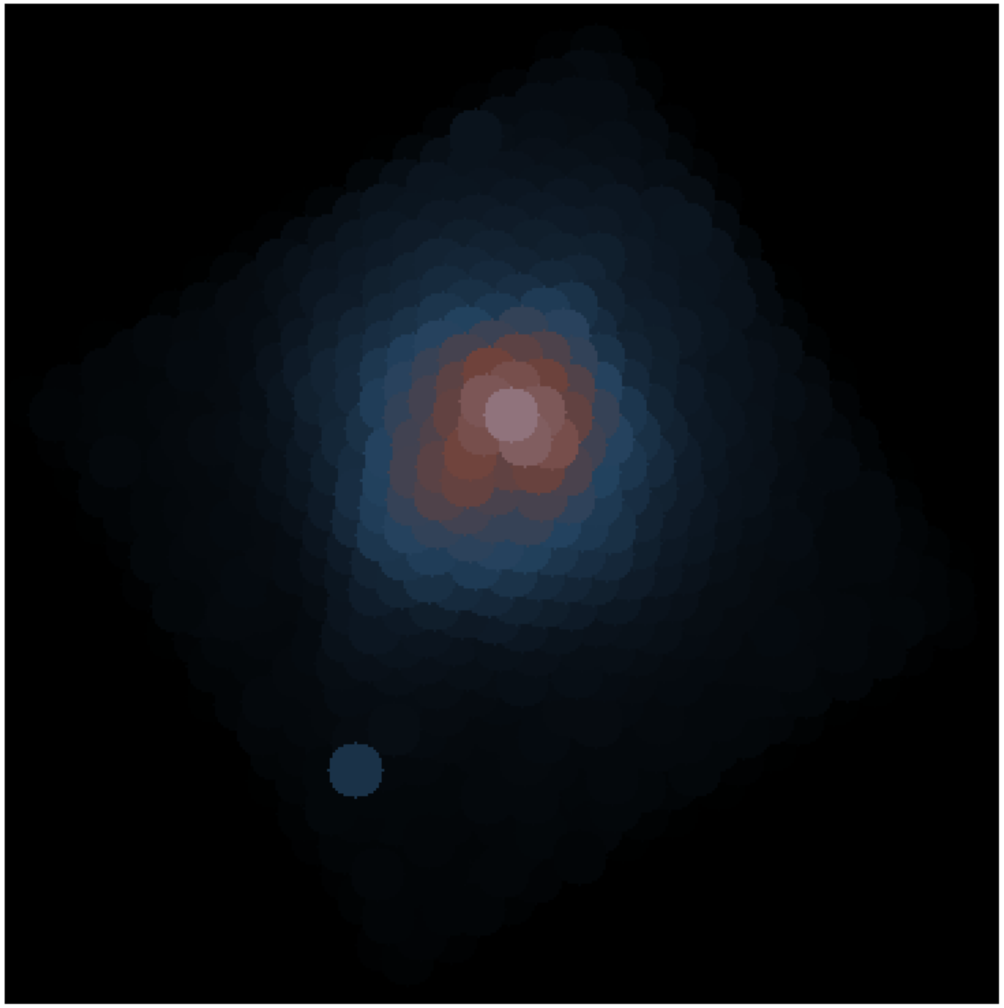
  Start at max pixel value in image and group the ungrouped pixels within radius. Goto the next highest pixel value not already included in a group and repeat. Radius can be a fixed value used for all pixels or taken from input image.

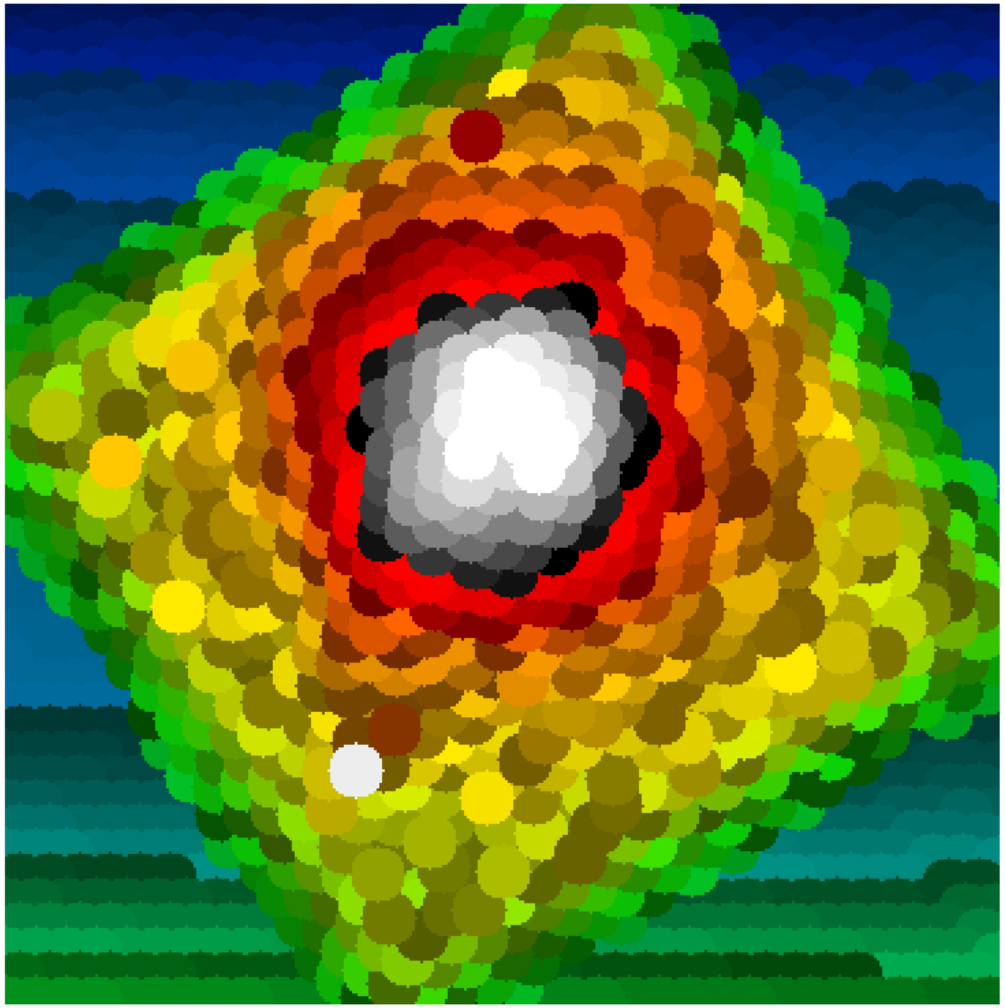
- Example commands

```
% dragon_scales smimg.fits psf=10 outfile=ds_10px.map \
  binimg=ds_10px.img shape=circle clobber=yes


% dmimgadapt img.fits img_500.asm tophat min=1 max=100 numrad=100  \
  radscale=lin counts=500 radfile=min500.map verb=3 clob+
% dragon_scales smimg.fits min500.map out=ds_500cts.map  \
  binimg=ds_500cts.img clob+
```

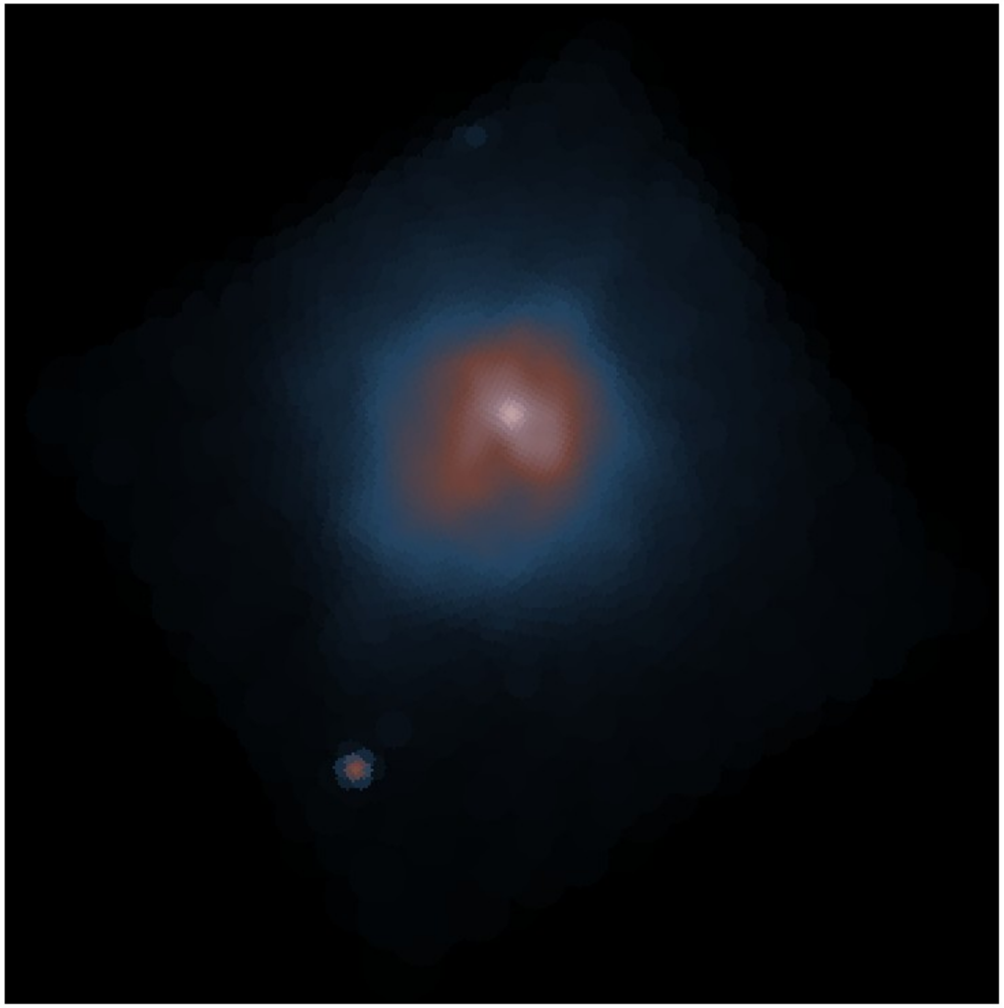
dragon_scales results

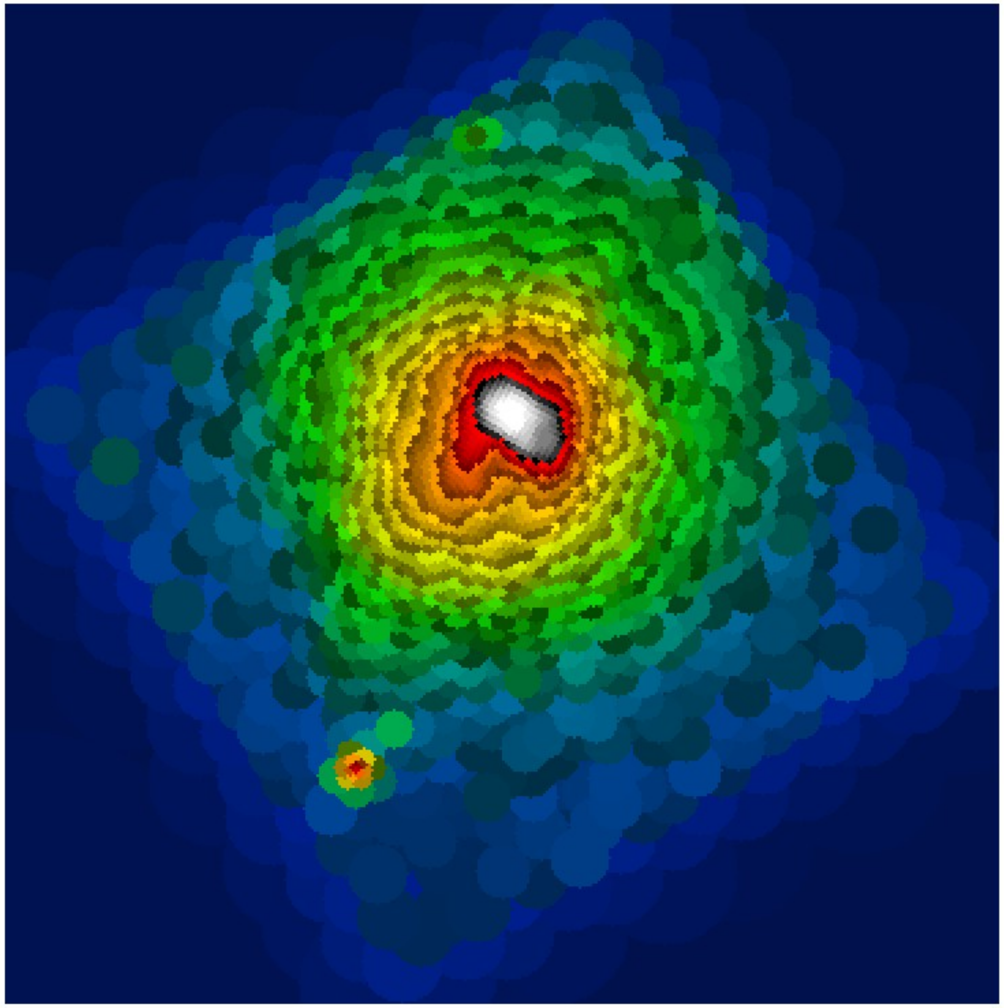




(left) dragon_scales' adaptively binned image using a fixed 10 pixels (logical) circular scheme.
(right) image showing the pixel group ID numbers.

dragon_scales results (2)





(left) dragon_scales adaptive binning using a variable radius. dmimgadapt was used to create an image whose pixel values are the size of a circular region needed to enclose 500 counts. That image was input to dragon_scales to adjust the size of the regions to group. (right) shows the group ID map – the size of the circle varies across the field.

dragon_scales parameters

Parameter	Description
infile	input counts image
psffile	input single radius value or image who value is radius of regions
outfile	map of regions (pixel value is group number/ID)
binimg	output adaptively binned image
shape	circle|box|diamond, distance measured as radius or offset or city-block

infile should be smoothed.

psffile needs to have radius be in logical|image coordinates.

grow_from_max

- Overview

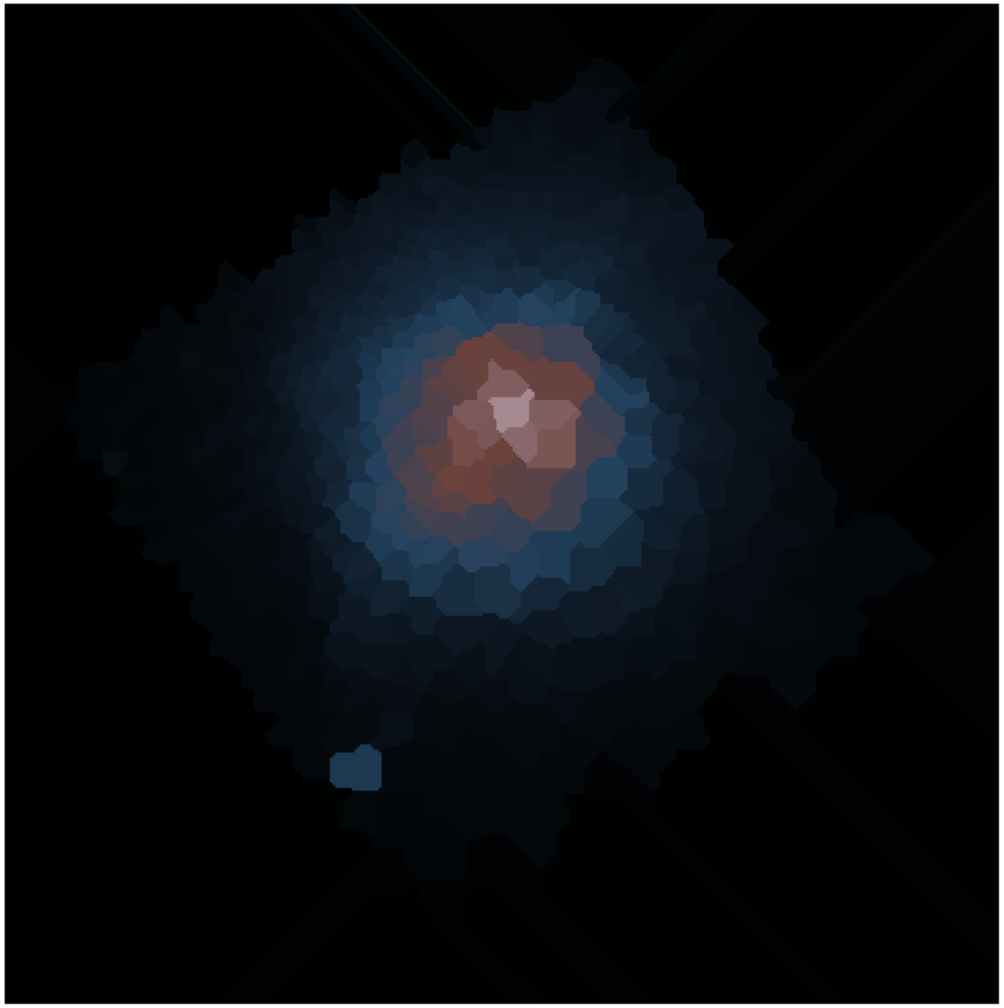
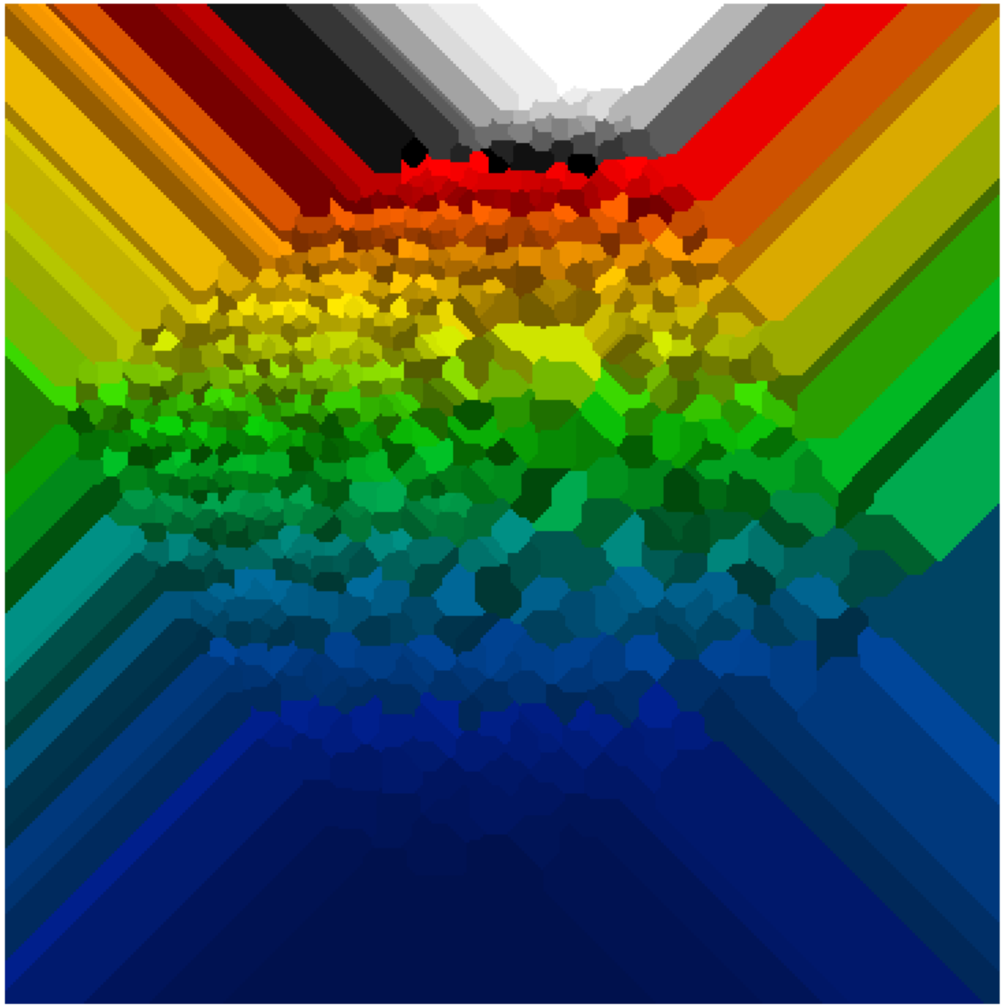
  This script starts by using dmimgfilt to find the location of the local maximums. Each local max pixel is given a unique ID (dmimgblob), and then is dilated until it hits its neighbors.   This is basically a watershed transform or can also be thought of as a tessellation.

- Example commands

```
% mkpsfmap img.fits psf.map energy=1 ecf=0.95 units=logical
% dmimgcalc infile=psf.map infile2=none outfile=ipsf.map\
  operation="imgout=((int)(img1*10.0))/10.0" clob+
% dmimgadapt infile=img.fits outfile=img.psf_asm function=gaussian \
  inradfile=ipsf.map mode=h clob+ verb=3

% grow_from_max img.psf_asm out=watershed.map binimg=watershed.img \
  shape=box rad=2.5 verb=2 clob+
```

grow_from_max results

(left) adaptive binned image where the local max in a 5x5 box in the PSF smoothed image are used to seed the dilation process. (right) is the map of region numbers.

grow_from_max parameters

Parameter	Description
infile	input counts image
outfile	map of regions (pixel value is group number/ID)
binimg	output adaptively binned image
shape	shape of the structuring element used to search for the local max. Ie local max within box|circle
radius	size of the structuring element.

The infile should be smoothed but not overly smooth. The input should have a large number of local maximums but not so many that every other pixel is one. Using an adaptively smoothed image at the size/scale of the PSF seems to work nicely.

The off-chip/edge behavior is especially dramatic for this algorithm. Dealing with pixels outside the FOV should be reviewed.

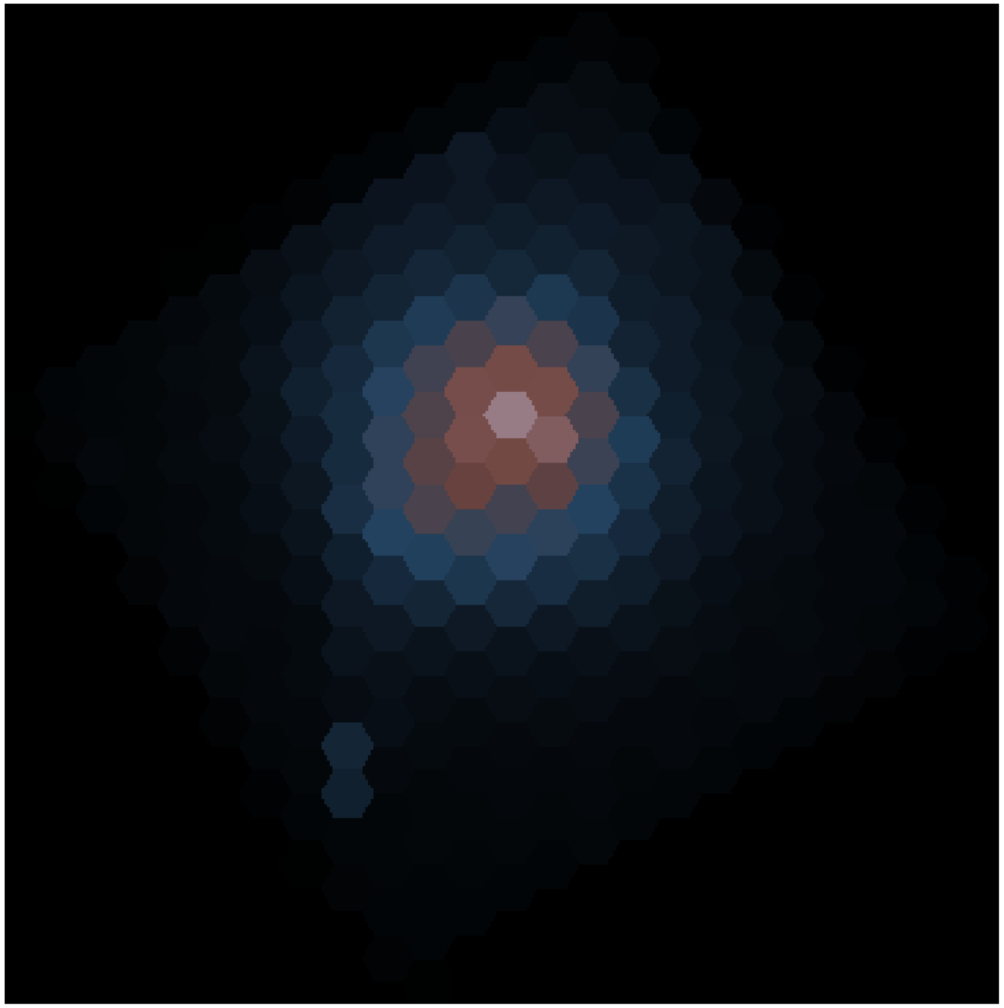
hexgrid

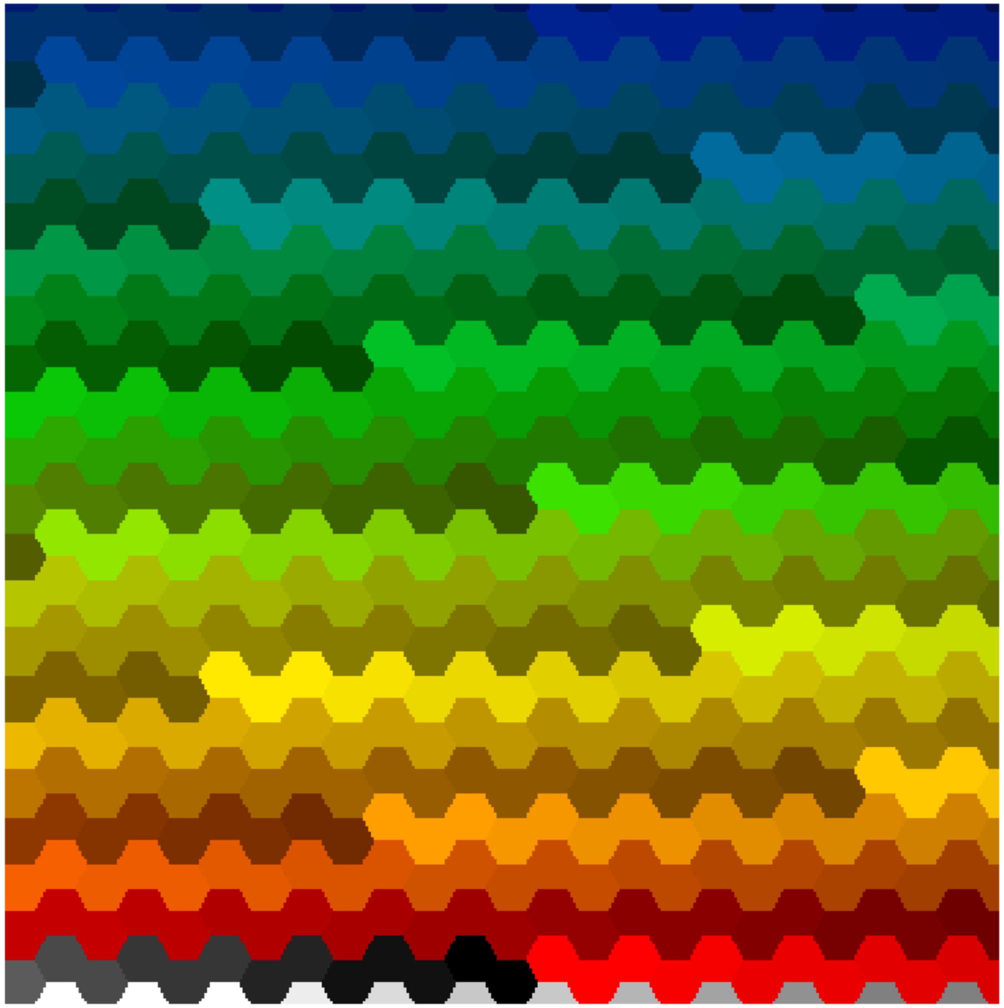
- Overview
  - The hexgrid routine is not adaptive but simply creates a grid of hexagonal shaped masks. Hexagons are a natural tiling element. They are the regular polygon with the most number of sides that can uniformly tile an image. As such they are the closest approximation to circular grid.
- Example commands

```
% dmstat smimg.fits cen-
% set px = `stk_read_num ")dmstat.out_max_loc" 1 echo+`
% set py = `stk_read_num ")dmstat.out_max_loc" 2 echo+`
% dmcoords smimg.fits op=sky x=$px y=$py

% hexgrid img.fits hex.map side=10 bin=hex.img \
  xref=")dmcoords.logicalx" yref=")dmcoords.logicaly" clob+
```

hexgrid results





(left) image rebinned on a hexagon grid (right). The center of one of the hexagon is forced to be at the input (xref,yref) location – which was set to the max pixel location.

hexgrid parameters

Parameter	Description
infile	input counts image
outfile	map of regions (pixel value is group number/ID)
sidelen	side length of hexagons (logical pixels)
binimg	output adaptively binned image
xref	x location of the center of 1 of the hexagons
yref	y location of the center of 1 of the hexagons

The pixel values in the infile are not used to create the grid; only the image dimensions.

As sidelen becomes small, there can be some quantization like effects/rough edges. A practical limit of 3 pixel is used.

The xref,yref location basically sets the “phase” of the hexagonal grid. It provides a way to keep the grid from spiting a single feature into multiple adjoining grids.

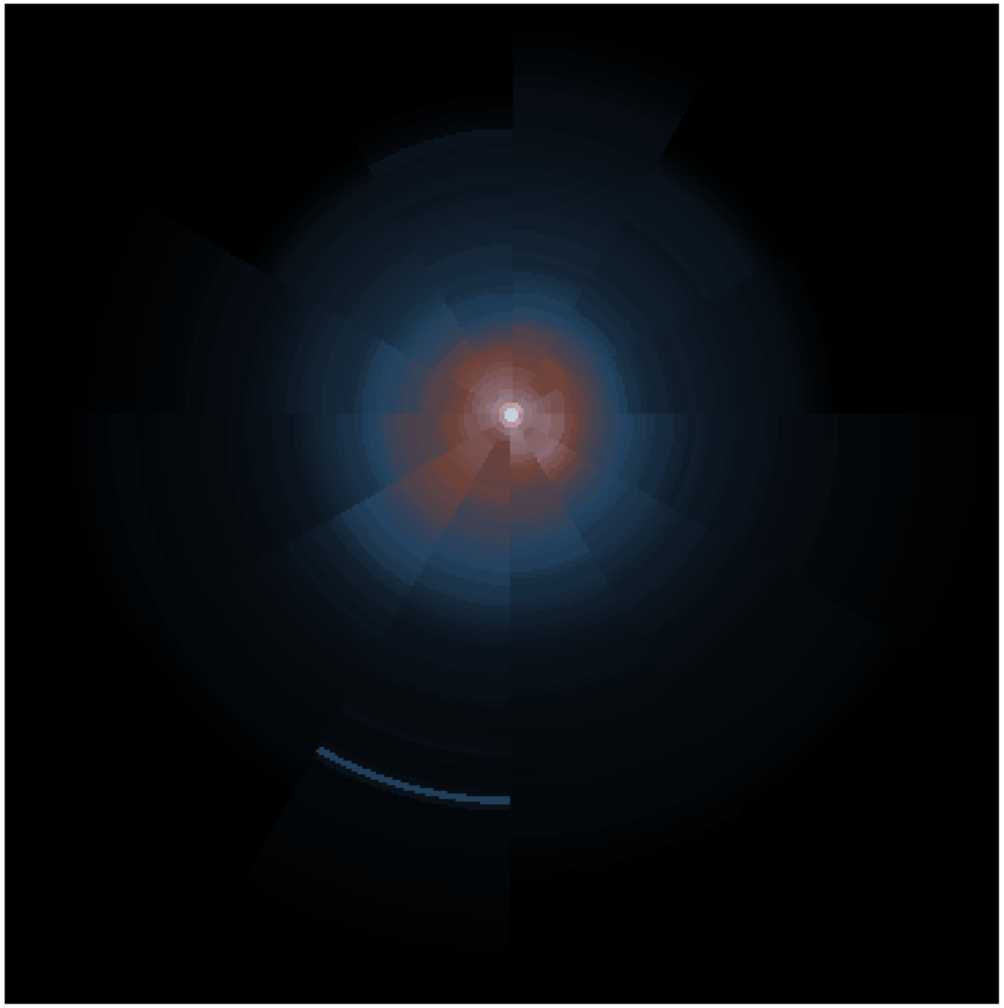
mkregmap

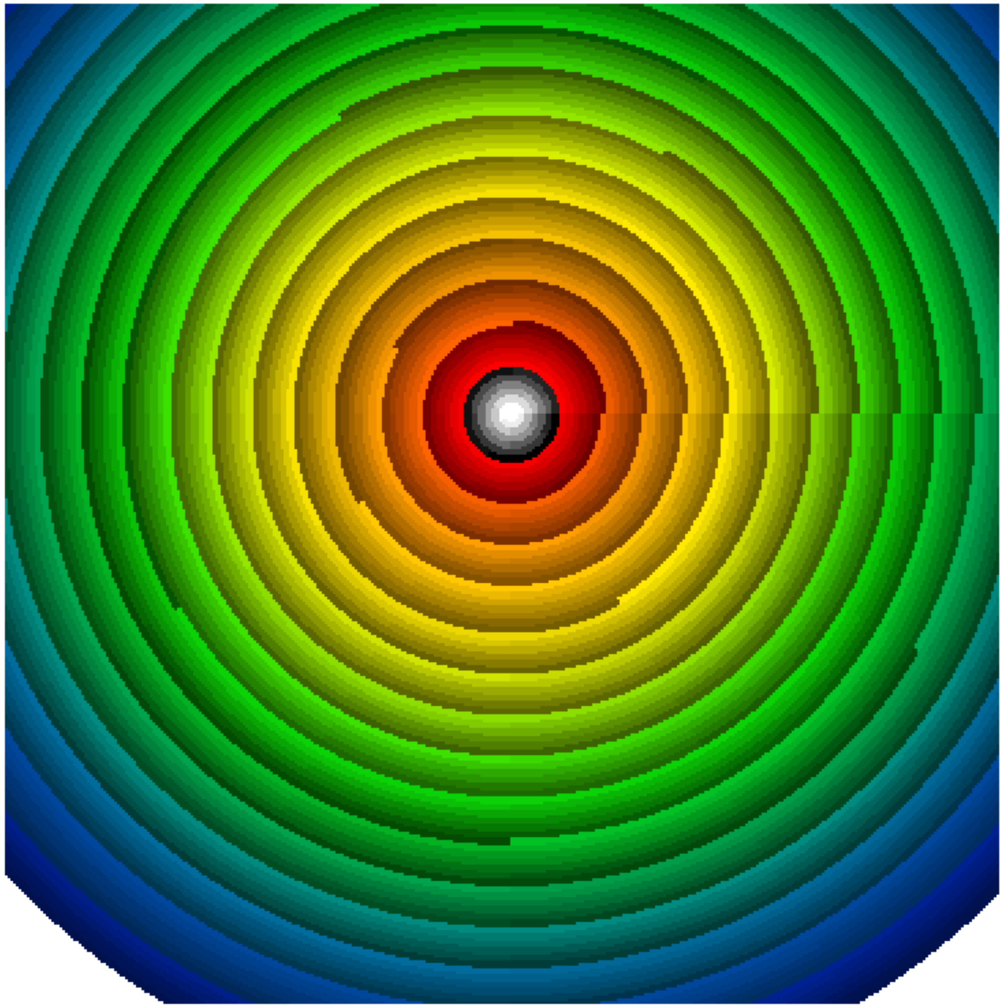
- Overview
  - Formerly called srcmap, this script reads in a stack of regions (1 to N) and labels the pixels in the output map based on which region they belong to. In the case of overlaps, the last (highest N) is used.
- Example Commands

```
% dmstat smimg.fits cen-
% set px = `stk_read_num ")dmstat.out_max_loc" 1 echo+`
% set py = `stk_read_num ")dmstat.out_max_loc" 2 echo+`
% mkregmap img.fits out=pie.map bin=pie.img coord=sky clob+ \
  regions="circle(${px},${py},10);pgrid(${px},${py},10:1000:10,0:360:30)" \


% dmellipse infile=img.fits outfile=ellipses.fits  \
  fraction="lgrid(0.05:1.0:0.025)" shape=ellipse clob+ step=20 verb=2
% dmsort ellipses.fits ellipses_sort.fits -component clob+
% mkregmap infile=img.fits outfile=ellipse.map clob+ \
  regions='ellipses_sort.fits[#row=igrid(1:100:1)]'
```

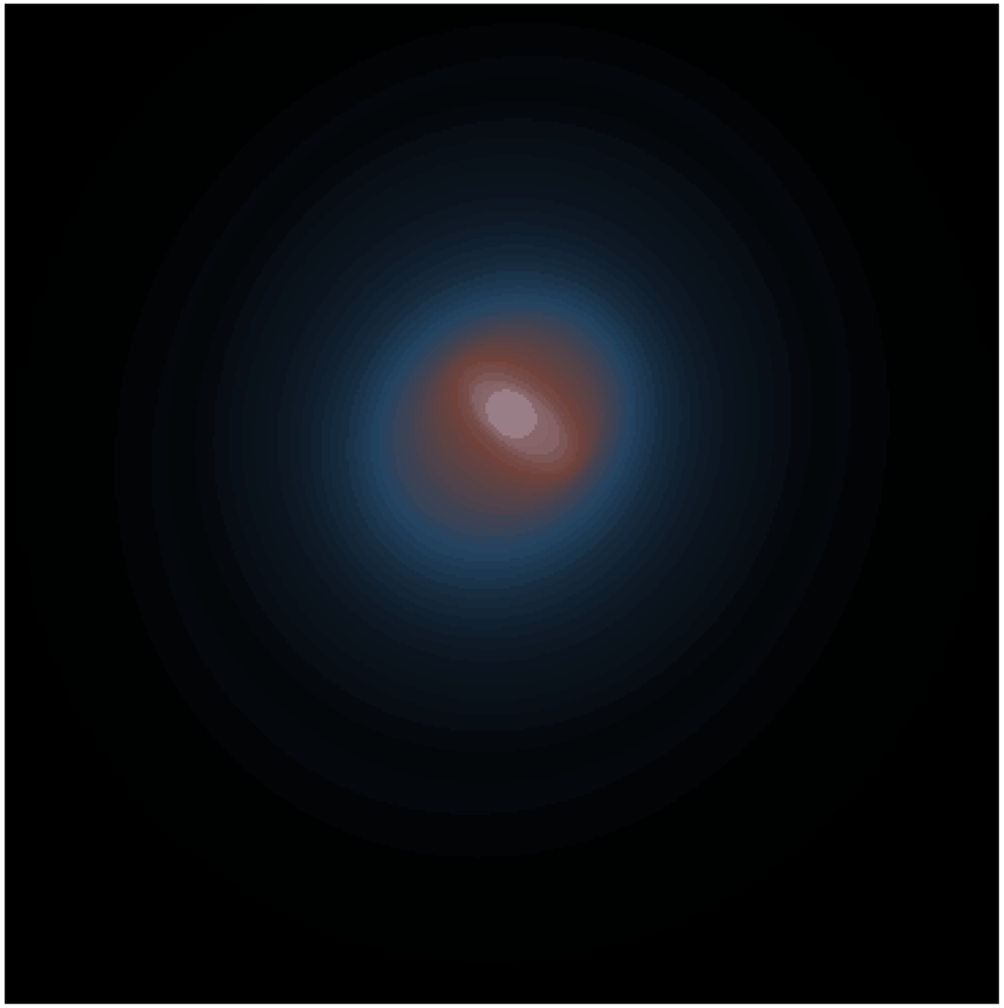
mkregmap results

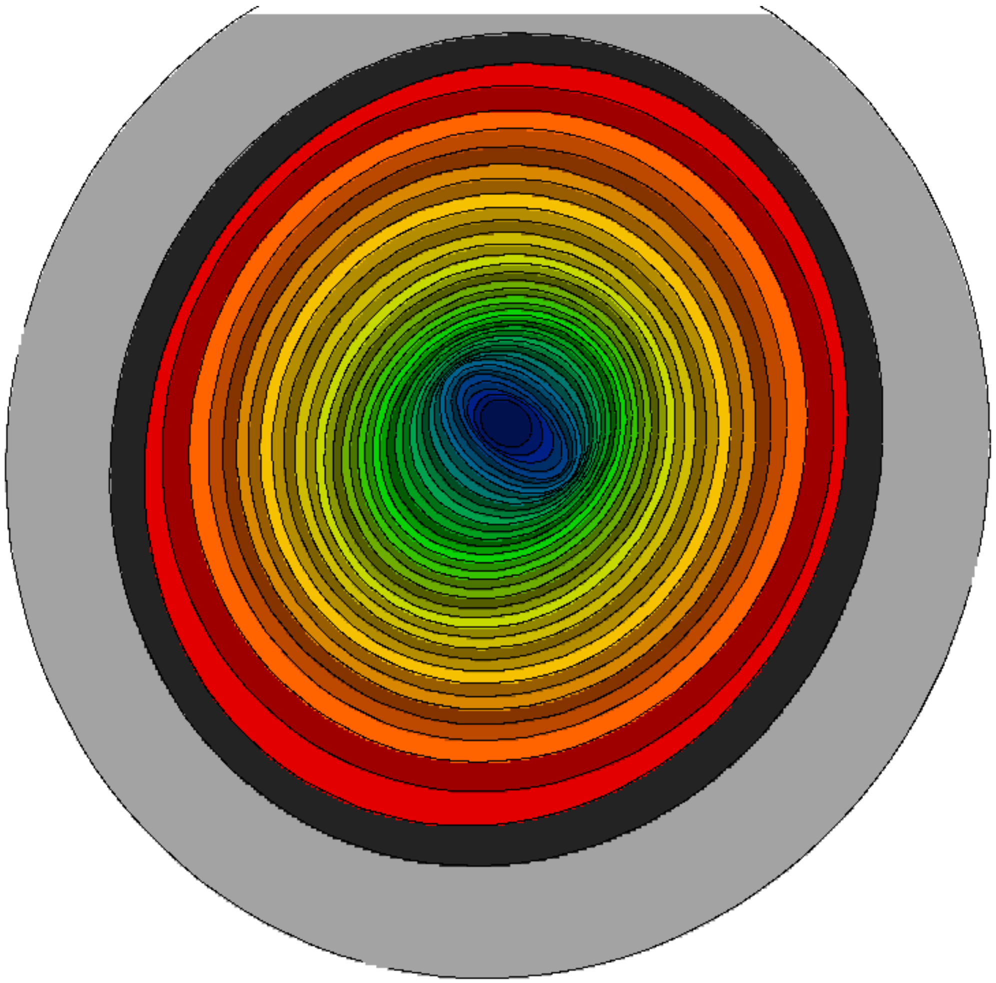




(left) image regridded using a polar grid via the stack library's “pgrid()” syntax with a circle() used around the center. Note the bright off-axis source is smeared into a large pie-wedge. (right) shows the region ID map.

mkregmap results (2)





dmellipse was used to create a grid of ellipses that enclose from 5% to 100% of the counts in 2.5% increments. The regions were reverse sorted and then fed into mkregmap to create the binned image (left) and region map (right).

mkregmap parameters

Parameter	Description
infile	input counts image
regions	stack of regions
outfile	map of regions (pixel value is group number/ID)
binimg	output adaptively binned image
coord	coordinate name for region files

The coord parameter is helpful eg with XMM data that does not contain sky column descriptor (either pos or (x,y)).

Regions must be a stack – so need to use the 'igrid' stack-expander to turn a single FITS region file into stack which requires knowing the number of regions.

merge_too_small

- Overview

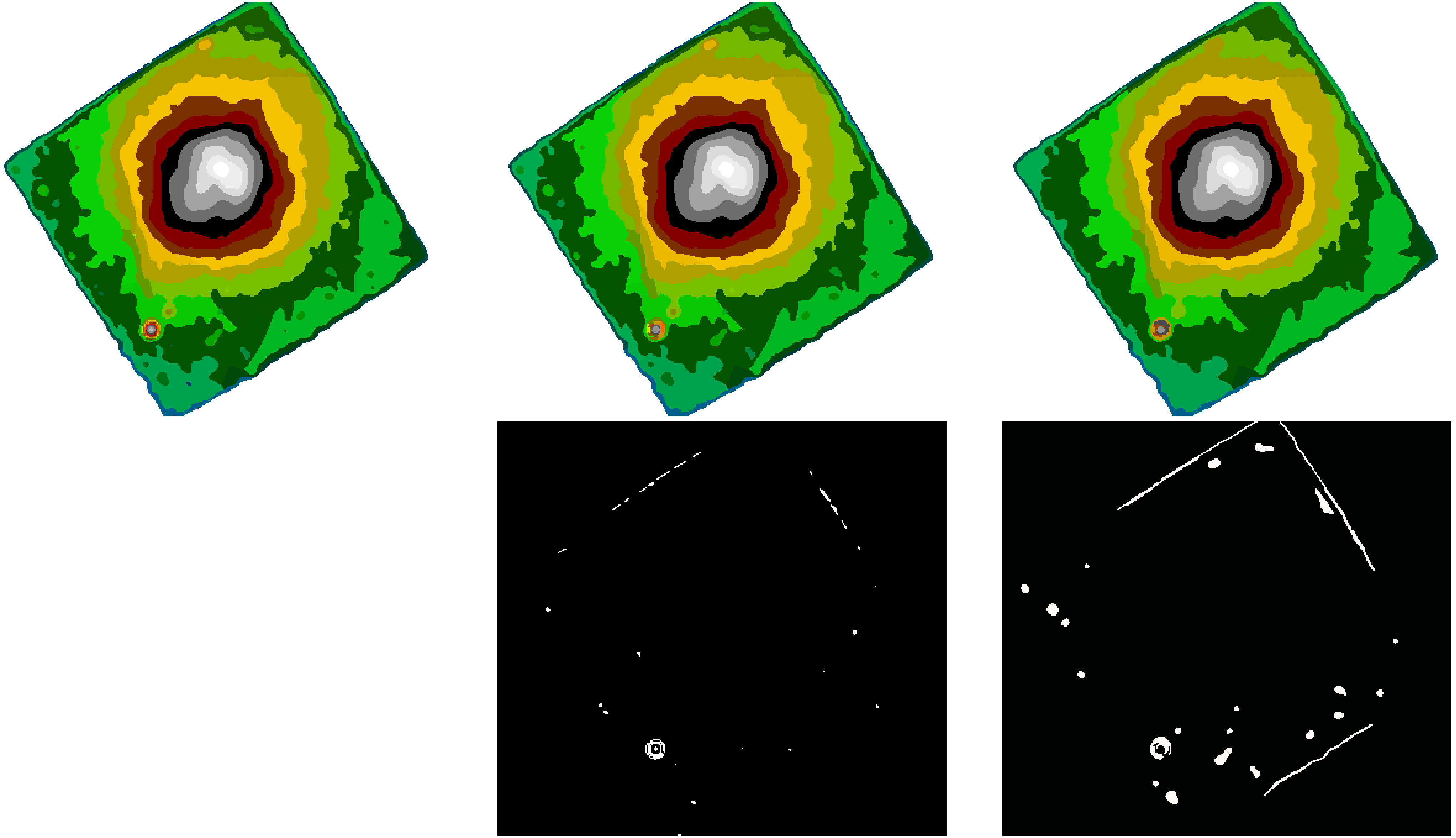
  Several of the methods shown can small produce spurious regions that contain very few pixels and/or counts. Basically whatever is needed to fill in the gaps. The tool looks at the input map and will reassign regions to the neighboring group with the least counts or pixels.

- Example commands

```
% merge_too_small infile=cbin.map outfile=cbin_10px.map \
  method=area  minvalue=10 verbose=2 clobber=yes imgfile= binimg=

% merge_too_small infile=cbin_10px.map outfile=cbin_10px_500cts.map\
  method=counts imgfile=smimg.fits binimg= minvalue=500 verbose=0\
  clobber=yes
```

merge_too_small results



The results of applying the merge_too_small routine to the contour_bin map. (Left) is the original output from contour bin. (Center) is merge_too_small set to require at least 10 pixels per region. (Right) takes the (Center) output an requires at least 500 counts per region. Bottom row shows diff images from previous image.

merge_too_small parameters

Parameter	Description
infile	input counts image
outfile	map of regions (pixel value is group number/ID)
method	counts or area
imgfile	counts image (reqd for method=counts)
binimg	output adaptively binned image
minval	reasign groups with counts/area less than this value.

area is in logical pixels.

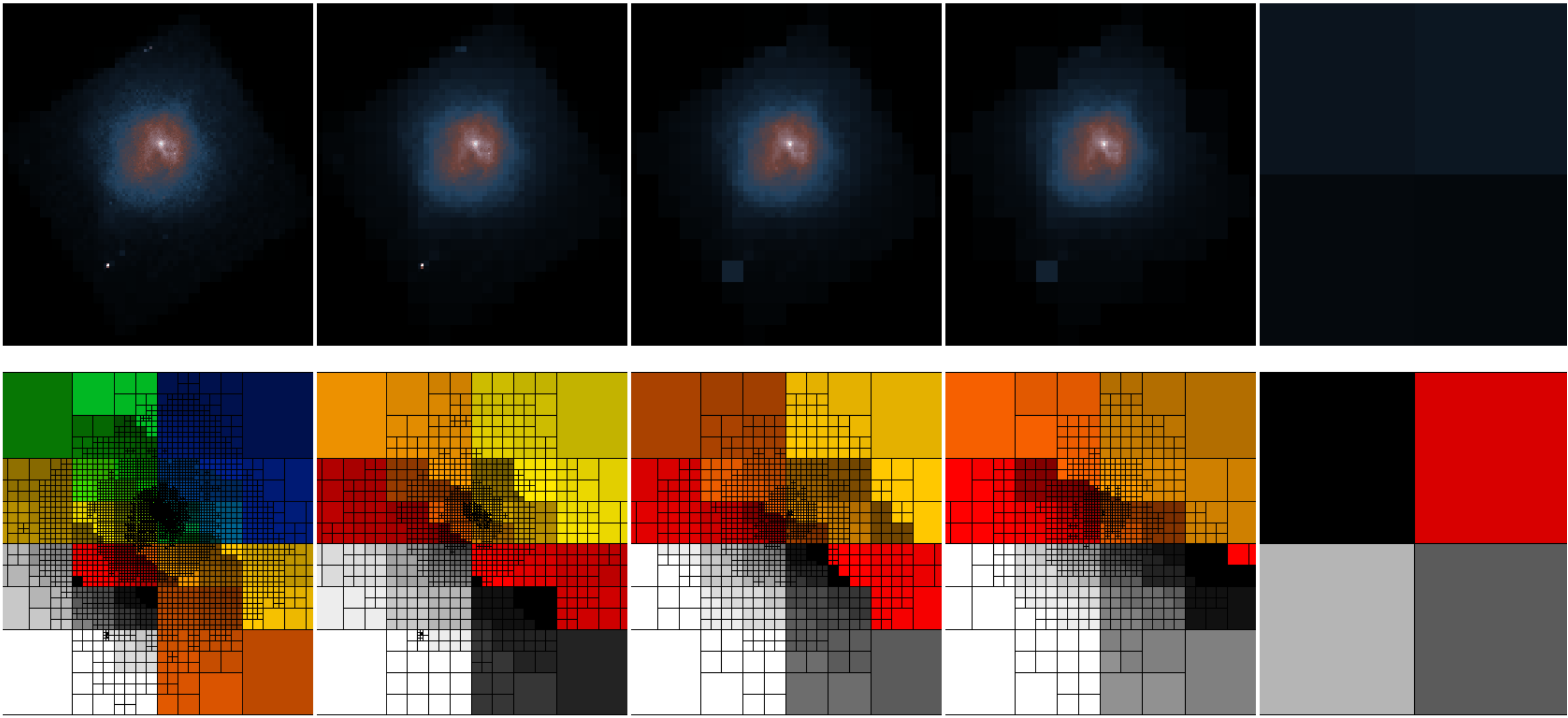
Groups with less-than-or-equal-to minval are reassigned.

Groups are always reassigned to neighboring group with lowest area|counts.

dmnautilus ++

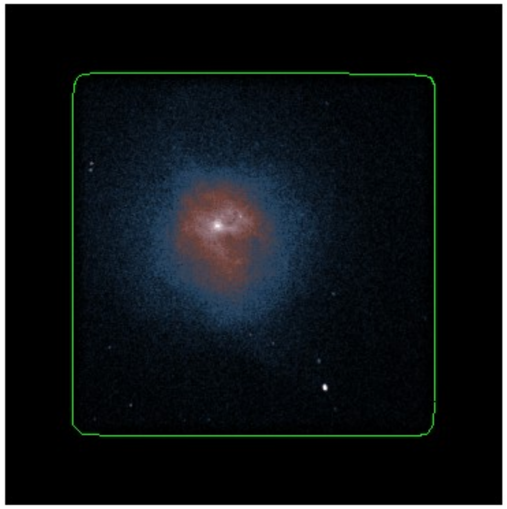
- Overview
  - Prototype code was delivered to DS/Warren on 6/5/2015 to invert the SNR threshold logic – that is to make the SNR be a lower-limit rather than an upper limit. New algorithm will only split the region into 2x2 if one, or more of the sub-regions exceeds the SNR threshold. New method parameter controls behavior. method=0 is the original algorithm. method=4 requires all 4 sub images to exceed SNR limit.
- Examples

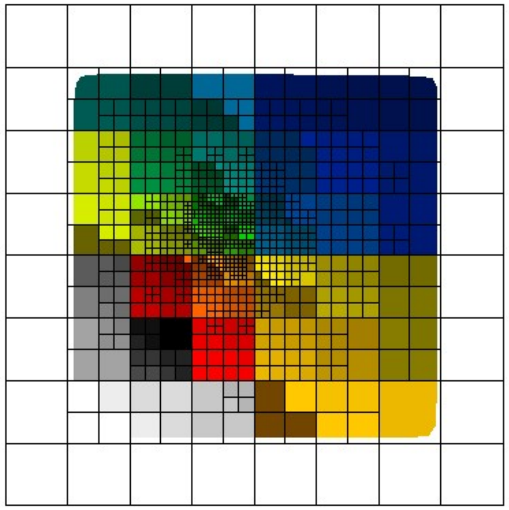
dmnautilus++ results

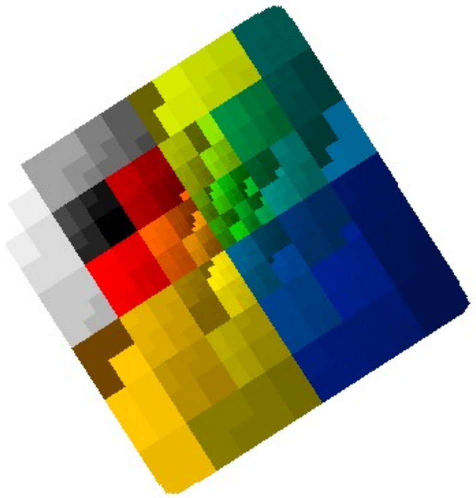


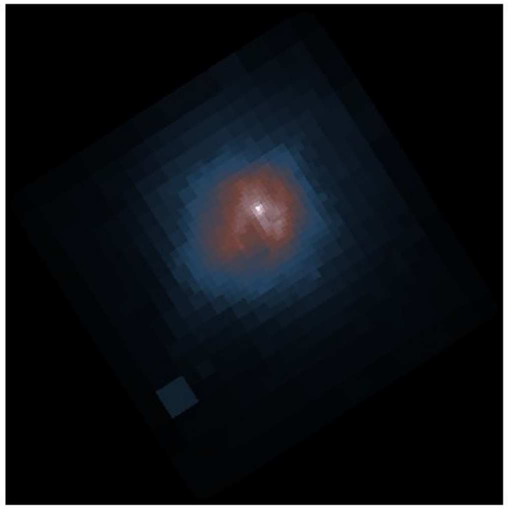
Top, Left to Right shows dmnautils++ with method=0, 1, 2, 3, 4. Bottom left-to-right shows the pixel maps. Due to the image rotation and lack of subspace info, the 0 padding around the image greatly affects method=4 (requiring all 4 sub-regions to meet SNR threshold).

dmnautilus++ results(2)


a


b


c


d

```
% dmregrid2 img.fits img.rot res=1 method=sum theta=57.328 rotx=182 roty=183
% dmimghull img.rot img.rot.hull
% dmnautilus++ @@dmnautilus "img.rot[sky=region(img.rot.hull)][opt full]" \
  img.rot.abin outmask=img.rot.map snr=15.8 meth=4 clob+
% dmregrid2 img.rot.map img.rot.rot.map res=0 method=sum \
  theta=-57.328 rotx=182 roty=183 clob+
% dmmaskbin img.fits "img.rot.rot.map[opt type=i4]" img.rot.rot.img
```

To overcome the problems with method=4, we need to establish the edge of the field and it also helps to rotate the image to make the chip edges parallel to the image axes. (a) shows the rotated image with a FOV created with the dmimghull tool. (b) shows the dmnautils++ output with method=4. (c) shows the map rotated back to the original orientation, and (d) shows the binned image.

dmnautilus++ parameters

Parameter	Description
infile	input counts image
outfile	output adaptively binned image
snr	target SNR limit
method	number of subimages required to be above SNR
inerrfile	input error file
outmaskfile	map of regions (pixel value is group number/ID)
outsnrfile	SNR achieved in each map region
outareafile	area of each map region

Getting the image with FOV and rotated to align the chip edges to the image axes is messy, but scriptable.

It is important that 2nd dmregrid uses resolution=0 which does not interpolate the value.

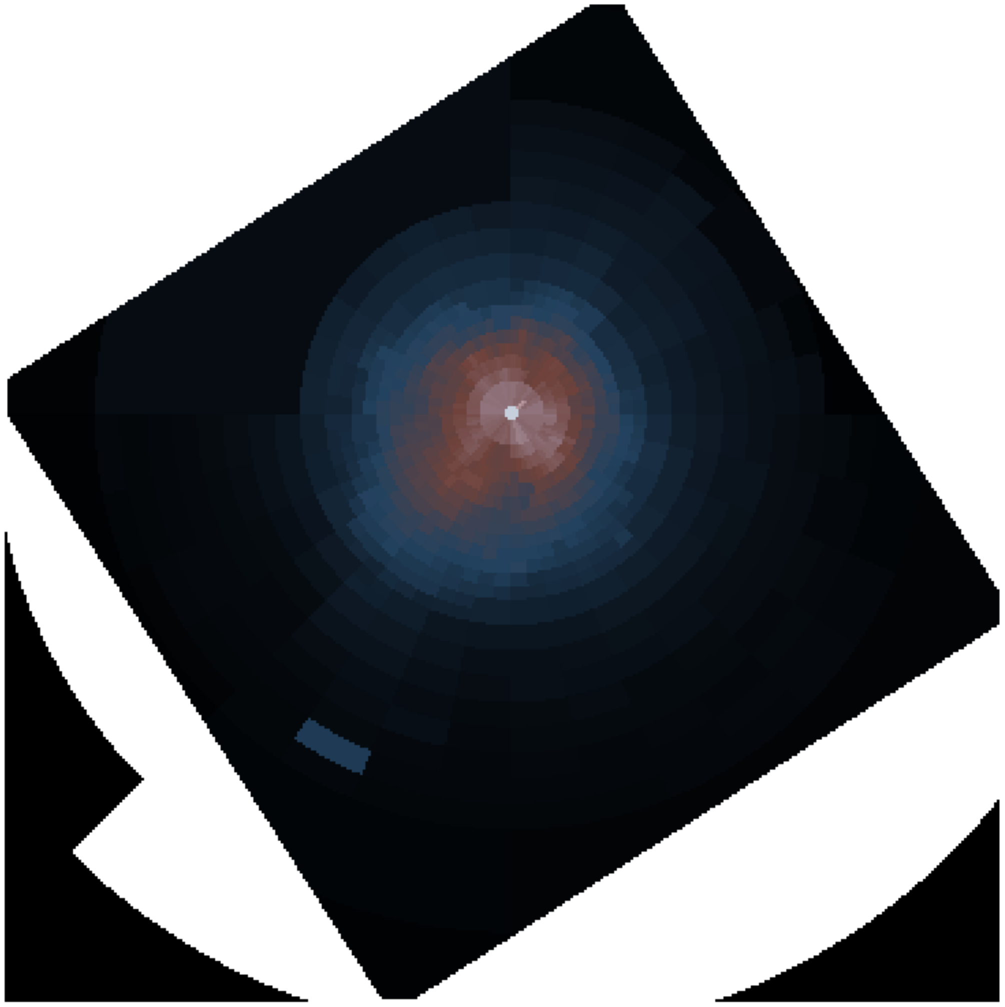
dmnautilus++++

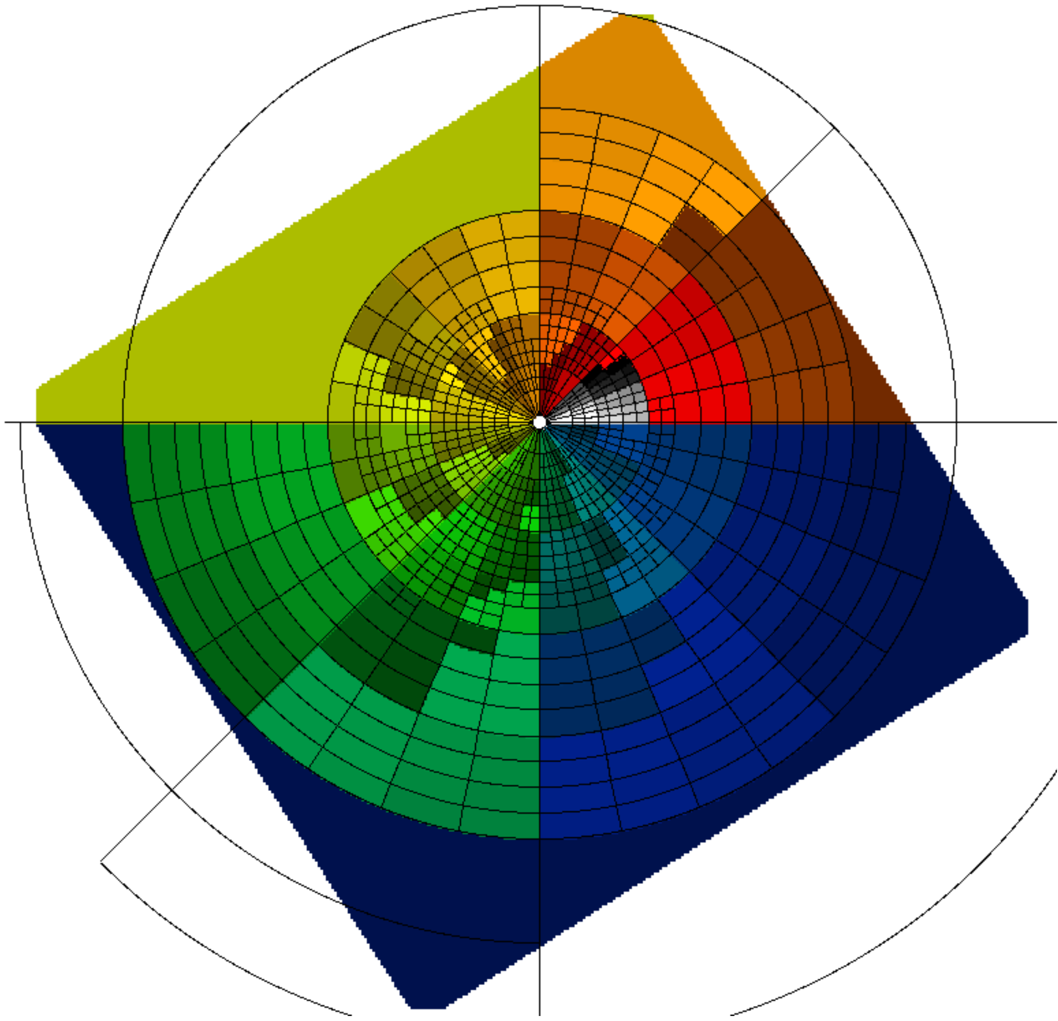
- Overview

  It occurred to me that rather than work in a rectangular system, dmnautilus could learn to work in polar coordinates. I've spent a day prototyping/playing with options/parameters. It looks promising but needs more work.

- Example

dmnautilus++++ results





dmnautils++++ parameters

- Details are still being worked.
- Requires at least
  - Location of center of pie slice (xcenter, ycenter)
  - A starting angle
  - A starting radius (uses a circle for inner region)
  - probably a min angle size and min radii size.
  - Can probably extend to elliptical pies with a fixed ratio of major to minor axis.

Parameter Comparison

	dmnautilus++	contour_bin	dragon_scales	grow_from_max	hexgrid	mkregmap	merge_too_small
infile	x	x	x	x	x	x	x
outfile	x (1)	x	x	x	x	x	x
snr	x						
method	x (3)						x (3)
inerrfile	x						
outmaskfile	x (1)						
outsnrfile	x						
outareafile	x						
verbose	x	x (4)	x	x	x	x	x
clobber	x	x	x	x	x	x	x
binimg		x	x	x	x	x	x
distance		x					
shape		x (2)	x (2)	x (2)			
levels		x					
scale		x					
maxcontours		x					
psffile			x				
radius				x			
sidelen					x		
xref					x		
yref					x		
regions						x	
coord						x	
imgfile							x
minvalue							x

(1) dmnautilus.outfile = {others}.binimg, dmnautilus.outmaskfile = {others}.outfile
(2) generic param name 'shape' is used very differently by different algorithms
(3) dmnautilus method = 0:4, m2s method=area|counts
(4) at verbose > 1, uses ds9 to display progress

Limitations

- no background
- no exposure variation
- most work best with smoothed inputs
  - each routine shows better/worse results with diff smoothing
- edge of image / subspace / NaN|NULL pixels can affect results
- no defined Figure Of Merit to rank quality of different algorithms.