Pixel Mask Filtering of CXC Datamodel

Helen He, Mark Cresitello-Dittmar, and Kenny Glotfelty

Smithsonian Astronomical Observatory, Cambridge, MA 02138, USA

Abstract. Data Model (DM) library, a CIAO X-ray analysis package, facilitates a wide range of "on-the-fly" data manipulation capabilities such as copy, merge, binning and filtering. The pixel mask filtering of image is integrated into the CIAO region filtering syntax and logic, and thus complements the conventional analytic shapes filtering of circle, box, etc. The integration allows for the standard operations on combining masks and analytic shapes such as include, exclude, intersect, and union. The mask is stored in a data block, referenced by the region filter string in the resulting file's data subspace. We will highlight the features and usage of pixel mask filtering in the CIAO software, and discuss some divergent effects between mask filtering and region shape filtering.

1. Introduction

The CIAO Data Model provides flexible filtering syntax which can be applied to events files. Pixel mask filtering or mask filtering is a new feature and extends the analytic shape filtering of DM library. The syntax of mask filtering is denoted by mask tag, 'mask()', and mask file in "mask(mask-file)".

The mask file is a 2D image/table storing binary data in FITS or an ASCII file containing 2 or more data columns. The data type of the mask file can be any numerical (integer or real) values. Therefore, any images can be used as mask file and any 2 numeric columns of a binary table file can be binned as image, so as mask file.

The mask filtering is to apply mask image in Boolean operation on events data array (FITS table columns or image), as any zero or non-zero values of mask data are treated as byte type 0 and 1. The mask is stored as part of the data subspace in the output. Regardless of the input data-type, the stored mask is always written byte type images in 0 or 1.

The Mask subspace in any output from mask filtering contains two parts, region string containing 'MASK' and the 'MASK' data block extension, as illustarted in following example. string is expressed

Table 1. 'sky' Subspace Region Filter String

sky	Real4	TABLE MASK MASK(MASK)||MASK(MASK2) Field area = 1234567.8 Region area = 1234

The 'MASK' and 'MASK2' inside the parentheses of 'MASK()' above indicate the named blocks below at Block-5 and Block-6, following the EVENTS and GTIs blocks, store the data in bytes, respectively.

Table 2. Data Blockss

Block 2:	EVENTS	Tabel	15 cols x 985	rows
Block 3:	GTI3	Tabel	2 cols x 7	rows
Block 4:	GTI1	Tabel	2 cols x 15	rows
Block 5:	MASK	Image	Byte(20x25)	
Block 6:	MASK2	Image	Byte(40x35)	

2. Masks versus Shapes Filtering

DM's filtering can be outlined in 3 formats but essentially falling into two types, shapes and masks. Both shapes and masks have 2-D attributes, but one is parametric geometry and other's arbitrary geometry in pixels. Taking 'sky' as filter descriptor, the 3 filter formats are

- 'sky=shape(parameters)': the shape of 'shape()', as a general term, can be the recognized shapes, circle, box, rectangle, etc. A 'circle' filtering is specified as 'sky=circle(x,y,r)', where '(x,y,r)' is the circle's center position and radius, respectively.
- 'sky=region(regfile)': the region tag, 'region()', is used to let DM library to read 'regfile', a region file, which stacks varieties of shapes with include or exclud.
- 'sky=mask(mskfile)': the 'mask()' tag, like the region syntax, instructs DM library to read the mask file, mskfile, which defines pixels region. Similarly, mask filtering also has masks operation in union and intersect when complex filterings are applied (to be discussed shortly).

Table 3. Mask Filtering Syntax

Mask	Region	Shape
sky=mask(f)	sky=region(f)	sky=circle(x,y,r)
exclude sky=mask(f)	exclude sky=region(f)	exclude sky=circle(x,y,r)
sky=bounds(mask(f))	sky=bounds(region(f))	sky=bounds(circle(x,y,r))
sky=mask(f1), det=mask(f2)	sky=region(f1), det=region(f2)	sky=circle, det=box
sky=mask(f1)&&sky=region(f2) ‖sky=box	same as mask	same as mask

List above is the mask filter syntax, aligned are region/shape filter for comparison. The 'f' in the 'mask(f)' and 'region(f)' represents mask and region file, respectively. The symbol '!' in front of filter syntax indicates the syntax is not allowed or unsupported.

2.1. Masks Intersect in AND-Operator

Run dmcopy events mask filtering in 'sky' descriptor, and the output is new events with a mask data subspace, mask1. Run dmcopy the new events another mask filtering, mask2, where the consolidation takes place on mask1 (the existing mask) and mask2 (the run-time mask),

```
dmcopy   ' evt[sky=mask(mask1)]'        evt_m1
dmcopy   'evt_m1[sky=mask(mask2)]'      evt.copy
```

the output, evt.copy, has a 'MASK' block storing the data of mask1&mask2, which are overlapped. The AND/OR operations can be illustrated through two 3x3 images/matrix as followed.

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \&\& \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} -> \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \| \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} -> \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

2.2. Masks Union in OR-Operator

DM merge tool, dmmgerge, merges several event tables of the same columns into a single output table. When the event files contain masks subspaces, the masks are combined to one mask subspace in the output table, assuming all the events have mask extensions as 'evt1[mask1]', 'evt2[mask2]', 'evt3[mask3]'.

```
dmmerge   infile= 'evt1,evt2,evt3'   outfile=evt.merged
```

If all the masks are overlapped, the consolidated output, 'evt.merged', has one 'MASK' data subspace, whose mask bounds may be broader to enclose the input masks being union-ed. Howeever the completed union of this may not always happen as in following cases.

partial union

Should mask1 and mask3 be overlap but mask2 singled out, the merged output would have 2 mask subspace components as expressed below, where MASK is the union of mask1 and mask2, while MASK2 is the copy of mask3.

```
Component 1:   MASK(MASK)
Component 2:   MASK(MASK2)
```

no-union at all

Should the events have multiple GTIs per CCD, say GTI1, GTI2,GTI3, regardless masks condition, the merged output would have 3 mask subspace components, MASK, MASK2,MASK3 are simply the copies of mask1, mask2 and mask3.

```
Component 1: MASK(MASK)
Component 2: MASK(MASK2)
Component 3: MASK(MASK3)
```

3. Mask Binning

Run DM tool on an event file can have both filtering and binning, including mask filtering. In the masking-and-binning syntax, the binning still follows all the current CIAO binning syntax, for example, the bin scale must be matching to that of an existing file (image only) or must be the multiples of the existing image's scale. Different from the region/shape filtering, the mask of the mask filtering is also binned accordingly and stored as an new extension of the output. The algorithm of mask re-binning is to use logical AND-operation on the the sub-image (or the pixel cell), or should all pixel values in the pixel cell is 1, the binned pixel/cell is 1 otherwise is 0.

Matching Coordinates

Generate a mask image file, mask.scale2, in bin scale 2. Create two other events images in bin scales 1 and 2.

```
dmcopy   'anyevt[bin sky=2]'   mask.scale2
dmcopy   'evt[bin sky=1]'      img1
dmcopy   'evt[bin sky=2]'      img2
```

Command dmcopy the events, img1 and img2, with the same mask filterings,

```
dmcopy   'img1[sky=mask(mask.scale2)]'   error.img
dmcopy   'img2[sky=mask(mask.scale2)]'   okay.img
```

The first masking run fails in mis-matched coordinates of the mask (mask.scale2) from the events (img1). The second masking run, it succeeds as matched coordinates of the mask with the the events (img2).

Bin Scale Multiples

Run dmcopy events masking-and-binning with mask.scale2 file in various bin scales,

```
dmcopy   'evt[sky=mask(mask.scale2)][bin sky=2]'   evt_es1.img
dmcopy   'evt[sky=mask(mask.scale2)][bin sky=4]'   evt_es2.img
dmcopy   'evt[sky=mask(mask.scale2)][bin sky=6]'   evt_es3.img
```

All the runs above are succeeded as the bin scales are the multiples of 2 of the mask's scale, or 1, 2, 3. But, following binning specs will cause errors because the bin scales are not the 2's multiples,

```
dmcopy   'evt[sky=mask(mask.scale2)][bin sky=3]'   error.img
dmcopy   'evt[sky=mask(mask.scale2)][bin sky=5]'   error.img
```

Acknowledgments. The work has been supported by NASA under contract NAS 8-03060 to the Smithsonian Astrophysical Observatory for operation of the Chandra X-ray Center.