Issues in applications and the framework classes -- post-DC2 cleanup and changes for DC3

from: DC3 Management

vw: RHL has questioned whether we want to continue to use VisualWorkbench. (RHL) I don't think that we're gaining much from it, and I find the bits that we are using (pixel accessors) too limited for my applications. If we do stick with vw, then I think that we should inherit from the image classes, not require the user to follow a pointer to get to the vw objects --- this is a classic IsA relationship.

fw::Function:

- Consider not storing parameters in the function; make parameters the final argument of operator() instead. If we do this we have to decide how to check the length of the argument vector for each function invocation. (comment by Russell Owen) The two proposals I've seen are:
  - Live with the cost of checking the length for each function invocation -- in most cases it is cheaper than copying the parameters. This is simple but does add overhead.
  - Make the number of function parameters a template parameter. That resolves the speed issue but it makes polynomial functions trickier to use, especially those for which the number of parameters is a nontrivial function of the polynomial order. It also complicates the Python interface because one needs to explicitly instantiate a separate version for each allowed number of parameters.

fw::Convolve:

- Add separable convolution. Be sure to consider the effects on ticket 231 (but if we can live with 231 as it was resolved then there is no issue).

fw::MaskedImage:

- Add "origin" method (returns a MaskedPixelIAccessor) (comment by Russell Owen)
- Consider a single iterator that acts as both a pixel accessor and also an STL iterator. If that is not practical then consider making pixel_iterator truly STL-compatible; this would allow us to remove PixelProcessingFunction? and use the standard std::foreach function instead (comment by Russell Owen)

fw::Mask:

- The UW group would like to eliminate the use of bit naming. This would greatly simplify the Mask code and use of the Mask class. The main use case, as we understand it, is have a few planes available for temporary internal use so could we leave a few unassigned planes for that purpose?

fw::image classes (Mask, Image, MaskedImage, and in some cases Exposure):

- Fix python versions of +=, etc. (ticket #244)
- An efficient way to copy image data to and from numpy arrays would help unit tests (comment by Russell Owen). N.b. the hard part of this is making python Images and Masks behave as numpy arrays; this is doable (I did it in swig 1.3.27 for ACT), but the implementation got broken by some swig change. This is scary. RHL.
- Need unit tests
- subimage metadata needs some work. FITS files written from subimages have incorrrect headers (or at least are not usable by ds9 -see ticket #302). I'm not sure if there are any issues with subimages

being persisted in other ways. (comment by Russell Owen)
- Handle FITS metadata more gracefully. For example at present an Image and a MaskedImage read from a FITS file have the all the fITS header data as metadata. But that medata is not updated if a subimage is taken. Similarly an Exposure read from a FITS file has WCS information in the FITS metadata and in the WCS object.
- (RHL) The subimage model is, I think, probably wrong. At present, a subimage is a COPY of an image, not a pointer into the same object. This makes it impossible to write code that e.g. builds a mosaic. I'd like to see copies to be deep, and use subimage explicitly to get shallow copies. I find the current situation confusing.
- (RHL) Handle image offsets consistently. This is quite hard, and I don't know a good OO way to do it --- basically, pixel manipulations don't seem to map to OO very well. An example is that the detection::Footprint code needs to set the Footprint's coordinate systems allowing for the image's origin.
- (RHL) We need to make the iterators much more powerful, allowing relative subscripting.

fw::

- Put standard types into a .h file
- We are not consistent in our standard for pixel coordinates, ie what physical pixel coordinate is meant by image index (0,0).
    - This decision is partially realized by the choice for value of fw::image::PixelZeroPos? (currently 0.5). We need not only to choose the value for LSST (RHL and TSA agreed that 0 made a lot of sense), but also to ensure that it is used uniformly throughout the code, in particular by WCS (comment by T. Axelrod)
- Reorganize the source code as per mwi and VW (comment by Russell Owen)
    - Put image stuff subdir image, etc.
    - Add higher level .h files such as image.h for stuff in image/
    - Consider one file fw.h that includes everything
- Standardize on int or unsigned int for all of the following (comment by Russell Owen):
    - Setting cols and rows in constructors (int in Mask, Image but not Kernel or vw)
    - getCols/getRows (always unsigned now) RHL and a pain -- I usually end up casting it away.
- RHL Be consistent about our "left" and "right" conventiontions (e.g. do we want to follow the vw convention that bbox.max().x() is one after the last pixel?). My vote is "yes", but this involves changes to e.g. detection::Footprint.

dps:

- Overhaul the code that runs pipelines (this may already have been done)
    - Allow nodelist to be an input
    - Allow nodelist=None for running one copy on the local machine to support all the example code

detection:

- Detection needs to do better cosmic ray rejection.

imageproc::wcsMatch:

- Can speed it up by using two tricks from swarp:
    - Reduce the kernel size by 1 and offset the center by one pixel depending on the fractional pixel value.

- Use a separable kernel