Deep Processing

Update in progress

See S15 Multi-Band Coadd Processing Prototype for a proposed change to some of the logic described herein.

Overview

This page attempts to capture at a high level the software and algorithm development necessary to implement the processing of objects detected at the full survey depth (at the time of a particular data release), including the detection, deblending, and measurement of sources too faint to be detected in any individual visit. The algorithms to be used here are generally poorly understood; we have many options for extending well-understood algorithms for processing single-epoch data to multi-epoch data, and considerable research is needed to find the right balance between computational and scientific performance in doing so. Unfortunately, different algorithmic options may require vastly different parallelization and data flow, so we cannot yet make assertions about even the high-level interfaces and structure of the code. We do, however, have a good understanding of most of the needed low-level algorithms, so our goal should be to implement these as reusable components that will allow us to quickly explore different algorithmic options. This will also require early access to parallelization interfaces, test data, and analysis tools that will be developed outside the DRP algorithms team.

Inputs

- Calibrated Exposures from Visit Processing
- Final relative astrometric calibration
- Final relative photometric calibration
- Moving and transient sources from Image Differencing and MOPS*
- External Catalogs (e.g. Level 3 inputs or known bright stars)

* We don't need Image Differencing outputs to start the Deep Processing (e.g. we can probably do Image Coaddition first), and there may be some value to doing the DRP Image Differencing at the same time as some parts of the Deep Processing (Deep Background Modeling, in particular).

Stages/Components

The next few sections contain the various components of the deep processing, in *rough* order - the exact flow is very much TBD. In fact, the lines drawn between several components are also somewhat arbitrary; the distinctions between detection, peak association, deblending, and measurement are based on a baseline algorithm that is derived largely from the SDSS processing, which was considerably simplified by only needing to process a single epoch for every band. Algorithms significantly different from this baseline would define these steps and the interfaces between them differently. Assuming that baseline, or something close to it:

- In Deep Detection, we compute above-threshold regions ("footprints") and peaks within them for all static or mostly static objects. There may be more than one footprint for a given patch of sky, corresponding to a different detection images, which will in turn correspond to different filters (different morphologies, SEDs, or variability), so each object may be represented by multiple peaks at this stage. We expect Deep Detection to operate entirely on coadds, though what kind of coadds (and how many) is TBD.
- In Peak Association, we combine the detection outputs from different images, merge footprints that cover the same patch of sky, and associating peaks that we believe correspond to the same object. At the same time, we also include peaks from other sources - transient and moving objects originally detected in Image Differencing, and potentially "known objects" from other surveys (e.g. Gaia). The outputs of this stage are a set of merged, non-overlapping footprints, each containing one or more peaks. We expect Peak Association to be a largely catalog-space operation - with the important caveat that these catalogs include footprints, and hence are in some sense halfway towards images, both in terms of the memory that will be required and the operations that will likely be performed.
- In Deep Deblending, we generate a "parent" object for each footprint, as well as a "child" object for each peak within it, along with a division of the pixel fluxes within each footprint amongst the children in that footprint. This generates a single "heavy" footprint (containing pixel values as well as describing a region) for each child object. The first stages of this will certainly operate on coadds, but we will likely need to have multiple coadds in memory at once, and it is possible that we may need to access the original input images at some point as well.
- In Deep Measurement, we run several algorithms on each source (both parents and children) to measure its properties. The heavy footprints generated by Deep Deblending are used to attempt to measure each child independently of its neighbors, but we may also run some algorithms in a mode that fits all children simultaneously. This will begin on coadd images (probably just a single direct coadd for each band), but will ultimately need to operate on all of the input images that contain the object, probably simultaneously.

Image Coaddition

We'll almost certainly need some sort of coadded image to detect faint sources, and do at least preliminary deblending and measurement. We'll use at least most the same code to generate templates for Image Differencing.

Because there's no single coadd that best represents all the data it summarizes, there are several different kinds of coadds we may want to produce. It's best to think of coaddition as a major piece of the algorithms below, but one which will be executed up front and reused by different stages of the processing; while there will be algorithmic research involved in determining which kinds of coadds we build, that research effort will be covered under the detection, deblending, and measurement sections, below.

All coadds will be produced by roughly the following steps:

1. Warp to the coadd coordinate system
2. Convolution (coadd type dependent)
3. Scale fluxes to coadd photometric system
4. Stacking

The overall processing flow for all coadds will likely be the same (though the final processing flow may or may not match our current prototype), and will share most of the same code.

Types of Coadds

- Direct: no convolution is performed, which further requires that little or no rejection be used in the stacking phase. PSF models and aperture corrections must be propagated through the coaddition process, as there will be discontinuities on the coadd image that will make determining these quantities from the coadd essentially impossible. This is the coadd type most similar to a single long exposure, and we can run any of our usual single-frame algorithms on it with relatively little modification. It will have correlated noise due to warping, but less than any other type of coadd. Input images can be weighed by exposure time or photometric quality, but no optimal weighting exists in the presence of different seeing in different exposures; some information from the best exposures is always lost.
- PSF-Matched: convolve with a kernel that makes the PSF on the coadd constant. Noise on the coadd is highly correlated, and a tradeoff must be made between the desired image quality on the coadd and the exposures that can be included; exposures with a PSF more than a little broader than the target PSF cannot be matched. A large amount of information from the best exposures will be lost, making this a poor choice for deblending or blended measurement. Single-frame algorithms that are not influenced by correlated noise can be applied as usual, and the measurement of colors is considerably easier, as the target PSF will almost certainly be the same in all bands. Unlike direct or likelihood coadds, PSF-matched coadds may be created using a stack that does outlier-rejection.
- Likelihood: convolve with a kernel that creates a map of the likelihood of an source (for given morphology) at the center of pixels: this is a generalization of the "Kaiser coadd", which is a map of the likelihood of a point source centered at each pixel, created by correlating each exposure with its own PSF. For a given morphology, this coadd is optimal for detecting that sources with that morphology. This clearly makes a point source likelihood coadd an excellent choice for point source detection, and we might be able to find a computationally feasible way to build likelihood coadds that allow us to pick up more extended sources (without being strictly optimal for these). Naively, the noise on a likelihood coadd will be very correlated, but it also has a very different meaning from the noise on other coadds, and it is not clear whether or not this will require a modified version of the single-frame detection algorithm. Existing single-frame algorithms for deblending and measurement cannot be run on a likelihood coadd, though at least in some cases an equivalent algorithm may exist. Formally, all exposures can be included in a likelihood coadd without degrading the result, and there is no trade-off between depth and image quality, but of course this still requires that all inputs have good PSF models and astrometric and photometric calibration.

For any of these coadd types, we may also want to coadd across bands, either to produce a χ^2 coadd or a weighting that matches a given source spectrum. It should be possible to create any multi-band coadd by adding single-band coadds; we do not anticipate having to go back to individual exposures to create a multi-band coadds.

Input Images

We will also need different coadds for different sets of input images. As discussed above, including images with broad PSFs in direct and PSF-matched coadds degrades the image quality of the coadd even as it adds depth, so we will likely want to produce coadds of these types that are optimized for image quality in addition to those that are optimized for depth. We will also want to build coadds that correspond only to certain ranges in time, in order to detect (and possibly deblend and measure) moving sources with intermediate speeds (i.e. offsets that are a small but nonnegligible fraction of the PSF size between epochs) that are below the single-frame detection limit.

Status/Challenges

We have a relatively mature pipeline for direct coadds that should only require minor modification to produce at least crude PSF-matched coadds. We need to finish (fix/reimplement) the PSF-matching coadds, and do some testing to verify that they're working properly. We haven't put any effort at all into likelihood coadds yet, and we need to completely reimplement our approach to χ^2 and other multi-band coadds.

Our propagation of mask pixels to coadds needs to be reviewed and probably reconsidered in some respects.

Our stacker class is a clumsy and inflexible, and needs to be rewritten. This is a prerequisite for really cleaning up the handling of masks.

We do not yet have a settled policy on how to handle transient, variable, or moving objects in coadds. Downstream processing would probably be simplified if transient and moving objects that can be detected at the single-epoch limit using Image Differencing were excluded entirely from the coadds, but some types of coadds do not handle rejecting masked pixels gracefully (at least in theory; this level of masking may work acceptably in practice), and we may want to adopt some other policy if this proves difficult. Similarly, it would probably be convenient if coadds were defined such that variable sources have their mean flux on the coadd, but this may not be worthwhile if the implementation proves difficult.

The coadd datasets are a bit of a mess, and need a redesign.

The data flow and parallelization is currently split into two separate tasks to allow for different parallelization axes, and the stacking step is parallelized at a coarser level than would be desirable for rapid development (though it may be acceptable for production). Background matching may also play a role in setting the data flow and parallelization for coaddition, as it shares some of the processing.

Choices about which images to include in a coadd are still largely human-directed, and need to be automated fully.

We have a flexible system for specifying coordinate systems, but we have not yet done a detailed exploration of which projection(s) we should use in production, or determined the best scales for the "tract" and "patch" levels. We have no plans yet for how to deal with tract-level overlaps, though it is likely this will occur at the catalog level, rather than in the images.

We have put no effort so far into analyzing and mitigating the effects of correlated noise, which will become more important when we regularly deal with more than just direct coadds (but may be important even for these). A major question here is how well we can propagate the noise covariance matrix; exactly propagating all uncertainty information is not computationally feasible, but there are several proposed approximation methods that are likely to be sufficient.

Dependencies

- *New Butler:* Needed to clean up coadd datasets and existing tasks.
- *Astrometric Self-Calibration*: There's really no point to building coadds (or doing any significant deep processing) as long as our astrometry is limited to the accuracy we expect from just visit processing. Without this, we'll be limited to building coadds from test datasets where we can obtain improved astrometry elsewhere (e.g. SDSS, or simulated images with associated "true" WCSs).
- *Parallelization Middleware*: we can get by with the current tasks during algorithm development, but we'll need slightly more sophisticated parallelization to implement production-ready pipelines, and development work would be sped up by having finer-grained (i.e. sub-patch) parallelization in the stacking step.
- *Deep Background Modeling:* we need to at least have a sense of the processing flow this will require before working on production-ready pipelines.

Skills Required

The algorithms here are relatively well-understood, at least formally, and most of the research work is either in coming up with the appropriate shortcuts to take (e.g. in propagating noise covariance, handling masks) or is covered below in detection/deblending/measurement. Almost all of the coding we need to for coaddition is pure Python, with the only real exception being the stacker (unless we need to make drastic changes to the convolution or warping code to improve computational performance or deal with noise covariances).

Scheduling

PSF-matched coadds are a requirement for any serious look into options for estimating colors. The measurement codes themselves need some additional testing before this becomes a blocker, however.

Likelihood coadds should be implemented as-needed when researching deep detection algorithms.

Dataset and task cleanup should be done as soon as the new butler is available.

The stacker rewrite and mask propagation should be fixed up before we spend too much time investigating measurement algorithm behavior on coadds, as it will make our lives much easier there if we start with sane pixel masks.

Noise covariance work may need to come up early in detection algorithm research, but if not, it's a relatively low priority, as it may not end up being important unless we decide we need PSF-matched coadds for colors, *and* we've determined that the measurement algorithms we want to use there are affected by correlations in the noise.

Deep Background Modeling

Algorithm

Traditional background modeling involves estimating and subtracting the background from individual exposures separately. While this will still be necessary for visit-level processing, for deep processing we can use a better approach. We start by PSF-matching and warping all but one of the N input exposures (on a patch of sky) to match the final, reference exposure and subtract these exposures, using much of the same code we use for Image Differencing. We then model the background of the N-1 difference images, where, depending on the quality of the PSF-matching, we can fit the instrumental background without interference from astrophysical backgrounds. We can then combine all N original exposures and subtract the N-1 background difference models, producing a coadd that contains the full-depth astrophysical signal from all exposures but an instrumental background for just the reference exposure. We can then model and subtract that final background using traditional methods, while taking advantage of the higher signal-to-noise ratio of the sources in the coadd. We can then also compute an improved background model for any of the individual exposures as the combination of its difference background relative to the reference and the background model for the reference.

Status/Challenges

We have prototype code that works well for SDSS data, but experiments on the HSC side have shown that processing non-drift-scan data is considerably more difficult. One major challenge is selecting/creating a seamless reference exposure across amplifier, sensor, and visit boundaries, especially in the presence of gain and linearity variations. We also need to think about how the flat-fielding and photometric calibration algorithms interact with background matching, as the fact that the sky has a different color than the sources makes it impossible to for a single photometric solution to simultaneously generate both a seamless sky and correct photometry - and we also need to be able to generate the final photometric calibration using measurements that make use of only the cruder visit-level background models.

The problem of generating a seamless reference image across the whole sky is very similar to the problem of building a template image for Image Differencing, and in fact the Image Differencing template or some other previously-built coadd may be a better choice for the reference image, once those coadds become available (this would require a small modification to the algorithm summarized above). Of course, in this case, the problem of bootstrapping those coadds would still remain.

We also don't yet have a good sense of where background matching belongs in the overall processing flow. It seems to share many intermediates with either Image Coaddition or Image Differencing, depending on whether the background-difference fitting is done in the coadd coordinates system (most likely; shared work with coaddition) or the original exposure frame (less likely, shared work with Image Differencing). It is also unclear what spatial scale the modeling needs to be done at, which could affect how we would want to parallelize it.

Dependencies

- *Photometric Self-Calibration*: the problems we're likely to see in background matching fully-calibrated inputs may be completely different, so we need to have this in place before we settle on a final algorithm
- *High-Quality ISR for Precursor Datasets*: performance will depend on the details of both the astrophysical background and the camera, so tests would be carried out ideally on a combination of HSC, DECam, an PhoSim data. But trying to do background matching on any of these without doing a very good job on ISR first would be a waste of time.

- *Parallelization Middleware*: putting all of the steps together will require at least some sort of scatter-gather, though we could continue with our current approach of manually starting tasks that correspond to different parallelization units while prototyping. Some algorithmic options may end up requiring even more complex interprocess communication, but it's hard to say at this point.

Skills Required

This is a difficult algorithmic research problem that interacts in subtle ways with ISR, Photometric Self-Calibration, and the data flow and parallelization for Image Coaddition. It should not be a computational bottleneck on its own, but it will likely need to piggyback on some other processing (e.g. Image Coaddition) to achieve this.

Scheduling

Because we can use traditional background modeling outputs as a placeholder, and the improvement due to background matching is likely to only matter when we're trying to really push the precision of the overall system, we can probably defer the complete implementation of background matching somewhat. It may be a long research project, so we shouldn't delay too long, though. We should have an earlier in-depth design period to sketch out possible algorithmic options (and hopefully reject a few) and figure out how it will fit into the overall processing.

Deep Detection

Algorithm

Detection for isolated static sources with known morphology and SED is straightforward:

- Build a likelihood coadd for the given SED and morphology.
- Find above-threshold pixels (Footprints).
- Find peaks in the Footprints.
- Grow the Footprints to match the target morphology.

We may need to take some care with noise covariances introducing in warping, and the details of finding peaks and growing Footprints are more heuristic than statistically rigorous. Nevertheless, the real challenge here is expanding this procedure to a continuous range of morphologies and SEDs. There are a spectrum of options here, with the two extremes being:

- We create a separate likelihood coadd for each point on a grid of morphology and SED, and run detection separately on each (with merging to happen later, at the Peak Association stage).
    - This approach puts a greater burden on the Peak Association stage by providing more inputs, but it also provides more information with each peak that can help resolve conflicts.
    - A key piece of making this sort of approach feasible is the ability to build certain kinds of likelihood coadds quickly from other likelihood coadds. For instance, each SED-specific likelihood coadd is just a different linear combination of of the single-band likelihood coadds, and we should be able to convolve point-source likelihood coadds to create likelihood coadds for other morphologies (or we may just bin them as an approximation).
    - We may want consider a variant where we actually do this continuously, by writing a thresholding code that can operate on a linear combination of images while allowing their weights to vary.
- We run detection with a lower threshold on one image (which may not need to be a likelihood coadd at all), then evaluate the significance of each detected peak afterwards, culling those that don't have the significance desired when evaluated using the range of morphologies and SEDs we're searching for.
    - This obviously requires much less effort in building detection images, but it may require a lot of effort in evaluating peak significance, especially if we end up with a lot of "garbage" peaks due to the low detection threshold.
    - It's not clear how well we can evaluate significance before we actually deblend, which may make it impossible to cull garbage peaks early, which in turn would make both Peak Association and Deep Deblending significantly harder.
    - Using fewer images for detection also means fewer Footprints, even if we end up with the same number of peaks, which should decrease memory and temporary storage usage.

We'll almost certainly use some mixture of these approaches; we'll certainly have more than one detection image, but it's not clear how many we'll have.

While most transients and fast-moving objects will be detected via traditional Image Differencing, we'll also want to build coadds that cover only a certain range in time to detect faint long-term transients and faint moving objects. We can use the same approaches mentioned above on these coadds.

Status/Challenges

Most of the low-level code to run detection already exists, though it may need some minor improvements (e.g. handling noise correlations, improving initial peak centroids, dealing with bad and out-of-bounds pixels). A notable exception is our lack of support for building likelihood coadds.

We have a large algorithmic landscape to explore for how to put the high-level pieces together, as discussed in the Algorithm section. This is essentially a lot of scripting and experimenting, with attention to the computational performance as well as the scientific performance. We probably need to put some effort into making sure the low-level components are optimized to the point where we won't draw the wrong conclusions about performance from different high-level configurations. Good test datasets are extremely important here, with some sort of truth catalog an important ingredient: we probably want to run on both PhoSim data and real ground-based data (ideally HSC or DECam) in a small field that has significantly deeper HST coverage.

We should spend some time looking into how we might evaluate the significance of a peak, post-detection; if we can do this efficiently, it makes the low-threshold/cull approach much more attractive. On the other side, we put at least a little effort into seeing how a threshold-on-linear-combination algorithm would look, in terms of operations per pixel and memory usage.

We'll need to pass along a lot more information with peaks, and we have a prototype for doing this using `afw::table` on the HSC side. Even once that's ported over, there will be more work to be done in determining what the fields should be.

The position of Deep Detection in the overall processing flow is moderately secure; we'll need to build several coadds in advance, then process them (at least mostly) one at a time, while probably saving the Footprints and peaks to temporary storage. We may build some additional coadds on-the-fly from the pre-built coadds (and it's unlikely we'll need to write any of these to temporary storage).

Dependencies

- *Image Coaddition*: we won't be able to make much progress analyzing the options here until we can make likelihood coadds.
- *Peak Association:* we can make a lot of progress without this, but the final stages of development will have to be done with the final stages of development here, as the algorithms are closely intertwined.

Skills Required

Mostly experiment and analysis work, with significant high-level Python coding and a bit of low-level C++ coding (which could be done by separate developers). Algorithms work involves using empirically-motivated heuristics to extend a statistically rigorous method to regimes where it's not formally valid, and it's at some level stuff that's all been done before for previous surveys - once we substitute likelihood coadds for single-visit likelihood maps, there's nothing special about multi-epoch processing here. It's the fact that we want to do a better job that drives the algorithm development here.

Scheduling

There's no sense scheduling any real development here until likelihood coadds and high-quality test data are available, and the fact that the algorithmic options here all fit within the same general processing flow means that changes here won't be that disruptive to the rest of the pipeline, so it's not a top priority from a planning standpoint. But we will want it to be relatively mature by the time we get to end-game development on the (harder) association, deblending, and measurement tasks, so we can feed experiments on those algorithms with future-proof inputs.

Peak Association

Algorithm

In Peak Association, we combine all of the detection outputs that cover a patch of sky, including not just Deep Detection outputs but peaks and Footprints that correspond to transients and moving objects, as obtained from Image Differencing. Each set of Footprints and peaks from a different origin - which we'll call "detection layers" - represents a different (and conflicting) view of the objects present in that patch of sky. Footprints from different layers that overlap can probably be straightforwardly merged, which can of course result in Footprints that were separate in some layers being combined in the end. Within each merged Footprint, the algorithm must also associate peaks from different detection images that correspond to the same object. This is much harder, as a peak in one layer may correspond to multiple peaks in another layer.

Our baseline plan (derived from the SDSS approach) involves an algorithm that does this without access to any actual pixel data; our hope is to pass enough information in the peaks themselves from detection to allow this stage to proceed entirely at the catalog level (though these catalogs will include Footprints, so they do contain some pixel-like information).

While generic algorithms for spatial matching may be useful for determining sets of overlapping Footprints, the work of merging their peaks is essentially a heuristic string of decisions based on thresholds and empirical testing, based on our understanding of the origins of the peaks and the properties of the peaks themselves. As such, it is unlikely we will be able to provide rigorous guarantees about its performance, aside from what we can get by running the algorithm on test datasets with associated truth catalogs.

Status/Challenges

While we have a very crude prototype on the HSC fork, this stage is unimplemented on the LSST side, and our best choice is probably to start from scratch, using the SDSS implementation as a guide.

We'll probably use the same test datasets as in Deep Detection, and probably run a lot of experiments for the two stages jointly. Defining metrics for performance will be an important early step, especially once we get beyond fixing the egregious failures, and have to start making tradeoffs that can improve behavior for some blends while making others worse.

We can probably write a preliminary version that will handle ~80% of objects (isolated or simple blends) correctly; the last 20% (or 19.9%, or whatever we do get correct in the end!) will take 80% of the effort.

It is unlikely we will get much use out of the existing Source Association code or Footprint merge code.

We should make some effort to make Peak Association consistent across patch boundaries.

A major algorithmic question is whether to allow any conflicts to remain to be resolved by the deblender and/or measurement; if we view the output of Peak Association as a hypothesis on the number of objects in a blend (and their rough positions) to be evaluated by the deblender, it may be valuable to allow multiple hypotheses to be passed to the deblender. We already intend to do this, in a small way: we intend to measure each the "parent" (everything in a footprint) as a single source, in addition to measuring the "child" sources individually. It may make sense to allow this at multiple levels, though clearly this has performance implications for later stages of the pipeline, and we'd want to reject as many hypotheses as possible early. This can be seen as a way to ensure the full Peak Association algorithm does not require pixel data; if we run into problems that *do* require pixel data, we'll punt them off to later stages of the pipeline.

Dependencies

- *Deep Detection*: we need a better sense of the number and types of detection layers before we can make much progress here.
- *Deep Deblending*: the dependency mostly flows in the other direction, but feedback from the deblender will certainly inform the development of Peak Association at its later stages.

Skills Required

This will be almost entirely C++ work, with a lot of trial-and-error experimentation involved.

Scheduling

Most work should happen after we've made significant progress on Deep Detection, but we probably want to sketch out high-level interfaces and put together some sort of placeholder quite early. That placeholder could probably take us quite far in terms of keeping lack of effort here from blocking work elsewhere.

Deep Deblending

Baseline Algorithm

In single-frame deblending, we attempt to allocate flux from each pixel to all of the sources who have contributed it, which can later be used to measure these objects individually. In deep deblending, we need to extend this procedure to multiple epochs and multiple bands (possibly using coadds), while extending the notion of separating "per-pixel" fluxes into whatever we need to separate objects for measurements. We start with the consistent set of peaks and footprints output by Peak Association; this gives us the complete set of sources (assuming, for now, that Peak Association produces only a single hypothesis, as discussed above).

Here's a proposal for the baseline algorithm - essentially the same as used SDSS, but operating on per-band direct coadds: (TODO: add mathematical definitions)

- In each band, for each peak, construct a template for that peak using symmetry arguments: the template is the minimum of each pixel value and its 180-degree reflection about the peak. If the template is sufficiently similar to the PSF model, or we know the peak is a point source for some other reason, instead use the PSF model as the template. Templates are different for each band, but the decision about whether to use the PSF for the template should be consistent across bands.
- Discard and/or merge any templates that are too similar in all bands (must be done consistently in all bands).
- In each band, fit the templates for all peaks within a footprint to the pixel data simultaneously.
- Compute the fraction each peak contributes to each pixel as the value of that peak's template for that pixel multiplied by its fitted amplitude. These are saved in "HeavyFootprints" - a description of the pixel region a source covers, plus its deblended pixels.
- Transform the HeavyFootprints to other epochs using the the same sort of matching kernels used in Image Differencing.

Algorithmic Variations

- Instead of transforming HeavyFootprints using difference kernels, fit PSF-convolved models to the deblended pixels, then use those models to subdivide the flux in other bands (after reconvolving with the appropriate PSF).
- Instead of generating templates separately in each band, create a single template in some linear combination of bands (e.g. a χ^2 coadd). Probably needs to be the same linear combination for each blend family, but we might find clever ways around that.
- Use sparseness-regularized deconvolution methods (e.g. Jean-Luc Stark's work) to generate PSF-independent templates (still using e.g. symmetry), which we'd then convolve with the PSF when fitting to the original data and apportioning flux.
- Generate templates in all bands simultaneously, with a Bayesian prior or other ansatz for how much similarity to require between bands.
- Use Gaussian mixture methods and to create templates, using Gaussian or double-Gaussian PSF models to convolve and deconvolve. Might be machine learning algorithms we can use here.

These variations range from straightforward extensions of the baseline to very open-ended explorations of different ideas, and it is by no means exhaustive.

Deblending vs. Joint Fitting

In addition to attempting to apportion pixel flux between sources, we also plan to fit multiple sources simultaneously (in Deep Measurement). And, of course, those fits can also be used to reapportion fluxes. However, we strongly believe there will always be a need for flux apportioning:

- We'll want to fit objects individually before fitting them simultaneously, as a faster and possibly more stable way to initialize and approach convergence for high-dimensional fits.
- We'll need to perform some measurements (e.g. aperture fluxes) that are important for comparing to other datasets (especially historical ones) and/or make fewer model assumptions than higher S/N measurements.

It is also possible that a flux apportioning method followed by individual model fitting may outperform simultaneous model fits, if the models used in the fitting are relatively simplistic but the flux apportioning templates are more flexible.

Status/Challenges

We have a single-epoch deblender that implements the symmetry-based templates on a single epoch and apportions flux into HeavyFootprints using them. This code needs some refactoring to allow its components to be reused more easily for many of the the algorithmic experiments we need to do.

We also have a prototype for a multi-band deblender that has most of the features of the baseline plan, on the HSC fork of the codebase, which will be ported to the LSST side of the codebase in the very near future. This is not expected to survive to production, even if the baseline approach turns out to be very effective, as the data flow doesn't support some features of the baseline that are expected to be critical, particularly the requirement that PSFs are identified consistently in all bands.

The production data flow for deblending is completely undetermined right now, with a major open question being the data axis for parallelization:

- If we parallelize over sky patches (and these are the same patches used in single-coadd pipelines, such as detection and coadd-based measurement), we may be limited by memory - this may rule out algorithms that need to have coadds of multiple bands and/or coadds with different depth/resolution tradeoffs in memory at once. If we frequently have blends that cross patch boundaries, this could also produce a significant number of poor deblends.
- If we parallelize over blend families, the size of those blend families could vary dramatically (and at the high end, we could still run into memory problems), and the I/O would likely be much more complex.

- We can imagine parallelizing in a much more fine-grained way (e.g. using scatter-gather or OpenMP to use multiple cores for different templates or pixels with a template, all within one blend family), but this will require more complex code across-the-board, even if it eliminates the risk of memory constraints. It also increases the risk of having cores go unused for a significant fraction of the processing time.
- We can come up with a divide-and-conquer approach to large blends, which could modify any of the above.

It may be that my concerns about memory are essentially unfounded, but we need to do some number-crunching to figure that out.

There are many algorithmic possibilities that need to be implemented and investigated. There is a huge amount of work to do here.

We don't have a good metric for how good a deblender is, or good test datasets on which to evaluate them. We'll probably have to rely a lot on human inspection of real data, along with some use of simulations and requiring that we obtain sensible color-color diagrams from deblended fluxes.

Dependencies

- *Deep Detection*: we'll need realistic outputs from this to rigorously test different deblending methods
- *Peak Association*: we'll need realistic outputs from this to rigorously test different deblending methods
- *Deep Measurement*: we'll want to test flux apportioning methods against joint fitting, use joint fitting as part flux apportioning. And we'll want to use post-deblend measured fluxes as a metric for deblend quality.

Skills Required

Everything: we'll need to write lots of qualitatively new C++ code, design a big complex system (probably mostly in Python?), possibly with non-trivial parallelization, explore and expand on completely new algorithms, and do a ton of experimentation.

Scheduling

We need to start work on this early, and work on it essentially continuously after that. The first stages need to be:

- a refactor and cleanup of the current code, so we can reuse its low-components in exploring other algorithmic ideas
- do some projections on blend sizes and figure out the memory requirements, and hopefully settle on a parallelization and data flow scheme (or at least rule some out)

Once we have a sense of the data flow, we can put together a very high-level interface that should support any of the algorithmic options we want to try, which we can then write as plugins. We'll then probably want to implement multiple plugins before we can test them fully - because truly rigorous tests will depend on having very mature versions of other pipelines - while ruling out as many as we can using simpler tests.

Deep Measurement

Deep Aperture Corrections