Problems and Solutions for LSST Shape Measurement

The Lighthouse People and The LSST Dark Energy Science Collaboration

ABSTRACT

LSST weak lensing science has unprecedented requirements for modelling the point spread function and for the accurate measurement of galaxy shapes in the face of blending. In this document, we describe the results of a workshop on these issues held at Point Montara in February 2017. We discuss available solutions for PSF modelling and shape measurement, lessons learned from their use in the Dark Energy Survey, remaining open issues, and progress and plans towards fixing those. In addition, we lay out a strategy for handling multi-epoch image data in an API useful with present weak lensing image analysis codes and a framework for validating PSF and shape measurement through image simulations.

1. Introduction

description of the things one needs for accurate shape measurement

This document describes progress made on several of these aspects:

- in Section 2.1, we describe the integration of complex world coordinate solutions into the multi-epoch data structures used by shape measurement codes in DES
- in Section 2.2, we outline the concept of an API that provides information similar to MEDS but fits the LSST data management structures
- in Section 3, we report on progress on the PSF modelling code PIFF
- in Section 4, we describe how to run the BFD shape measurement method on MEDS data structures
- in Section 5, we develop an image simulation framework suitable for testing shape measurement – both end-to-end and with some complications of real data turned off.

2. MEDS: Multi-epoch data structures

write why we need this: unified API for PSF modelling / shape measurement / photometry codes to access single frame image, weight, astrometry and PSF information

2.1. High-order instrumental astrometric distortions in MEDS

Gary, Troxel, Mike, Erin: describe how this is implemented

The MEDS python class would allow for flexibly swapping out the WCS in the input file FITS header by something more elaborate if we have the base class provide access functions for the `cutout_row/col` variables (in addition to the Jacobian) **(Erin)**. A derived class could then implement these differently, e.g. by evaluating Gary's WCS **(Gary)**. Both this and the way shape measurement codes find the matching PSFEx model files could be implemented by an external simple table that maps exposure and CCD IDs to auxiliary filenames.

2.2. A MEDS-Like API for LSST

MEDS has been an overwhelmingly successful approach for providing data to multple multi-epoch object characterization algorithms in DES, but it makes some assumptions that will make it impossible to use directly in LSST. But the MEDS experience is something LSST must learn from – multi-epoch fitting is a sufficiently complex task that we will not have the luxury of building a throw-away prototype to learn from before we build the final production system for LSST. From the LSST perspective, MEDS *is* that prototype, and collaboration with DES represents our best opportunity for learning from it in a production environment.

To that end, we have sketched out a design for a set of interfaces that generalize the MEDS approach to meet the needs of LSST. We're provisionally calling it `umemofi`: the Unified Multi-Exposure Multi-Object Fitting Interface. The design sketch can currently be found at `https://github.com/TallJimbo/umemofi`.[1] The goals of the new interface include:

- We need to separate the programmatic interface to data structures from their serialization and file formats, enabling drivers that repackage information from CCD-level images on the fly and serialize using LSST DM's "data butler" abstraction layer.
- We need to fully define all interfaces and data structures needed for fitting, instead of leaving some of them to be algorithm- or context-dependent.
- We need to support algorithms that simultaneously fit multiple objects.
- We need to run multiple algorithms in a single processing run, without duplicate I/O.
- We need to allow an external workflow system to control the highest levels of parallelization and data flow.

[1] None of this is working code yet, and its home may change in the future.

It is also a requirement that it be possible to implement at least the current MEDS *fitting* workflow on top of the new interface. We may or may not be able to implement the current MEDS *generation* workflow exactly using the new interface (but we hope to make it possible to implement something no worse, even if the logic isn't quite the same).

The design centers around the following classes:

- `ObjectData` and BlendData contain the revelent information from one or more (respectively) astronomical objects, independent of any observations they appear in: their location on the sky, and the results of any measurement algorithms that do not generate per-exposure output quantities.
- `Model` abstracts the per-object results of running a fitter or other measurement algorithm, including algorithm-independent serialization of those results and rendering them to images for diagnostic purposes and deblending (by subtracting neighbors).
- `ObsData` holds all information needed to measure a single object on a single exposure, and `ObsRef` provides an interface for backend-implemented lightweight handles that can load this information on-demand. `BlendObsData` and `BlendObsRef` generalize these for algorithms that wish to fit objects simultaneously. These classes also hold `Model` objects that represent per-exposure results, such as forced photometry.
- `ObsDataStack`/`ObsRefStack` and `BlendObsDataStack`/`BlendObsRefStack` are containers of `ObsData`/`ObsRef`, indexed by different combinations of object and exposure IDs.

These in turn rely on a number of primitive classes for representing key image processing concepts:

- images: conceptually a 2-d array with a potentially nonzero origin
- masks: integer images where each bit is assigned a different meaning
- *local* coordinate system transformations (which we approximate as locally affine over the scale of an object)
- *local* point-spread functions (which we approximate as spatially constant over the scale of an object, but potentially a function of wavelength)
- *local* photometric transmission functions (which we also approximate as spatially constant over the scale of an object, and a function of wavelength).

The local approximations are critical here: they let us avoid including the complex spatial variations of these objects in this interface. These local approximations must be generated from their complete spatially-varying forms by any `umemofi`backend, but those more complex objects remain completely hidden behind the backend.

Jim: `Algorithm` and harness interfaces (the end-game for all of this) still need some fleshing out; do that in Python before trying to describe it here.

We are also working to make this an active collaboration between DES and LSST, and in particular make sure there are no blockers on eventual DES adoption of `umemofi` as the primary interface for DES multi-epoch measurement (though we do not expect that to happen soon). While we expect both projects will always utilize different backends, reflecting different pipelines for earlier stages of processing, this should allow them to share the most important algorithmic code. This is a huge win for LSST (and DES members who are also LSST members), as it would allow DES-developed state-of-the-art model-fitting codes to be run on LSST DM stack outputs. It will hopefully also be advantageous for DES, as the new interface should make it easier to reconfigure and test algorithms (especially for dealing with blending), and it should consolidate much of the bookkeeping logic currently scattered across algorithms into a common harness. To encourage that cross-project collaboration, we intend to minimize the dependencies of the `umemofi` library itself, even if that involves duplicating in it some primitives LSST has already implemented in its own codebase. But even this can be seen as an opportunity for LSST: a way to get a small amount of important, high-use, stable LSST DM code (largely from `afw.geom`) into a separate library that will be more Pythonic and easier to install. Given that these geometry primives would require work (largely to integrate them with LSST's own `sphgeom` package, as has long been planned) to meet `umemofi`'s needs, doing this work in a "spin-off" package should involve little or no additional effot.

Our first near-term goals is to finish implementing the utility classes that form the core of the interface, at least in Python. LSST will ultimately want many of these classes to be available in C++, and it may make sense to implement these in C++ initially (especially if this largely transferring code from `afw` and adapting it slightly). We expect that DES will only use the Python interfaces for the forseeable future – while DES codes will of course use C or C++ in inner loops, we are confident the `umemofi` interfaces will never need to be used in those inner loops (the LSST desire for these utility objects in C++ is about design philosophy, not performance requirements). At the same time, we hope to start working on a MEDS backend that allows `Observation` objects to be loaded from DES-generated MEDS files using the `umemofi` interface. From there it should be easy to adapt any MEDS-based fitter into an `umemofi Algorithm`, allowing those codes to be run via `umemofi` on MEDS files in more flexible ways. A minimal LSST backend for `umemofi` with only simple intra-node parallelization and naive I/O should also be relatively easy to develop, which will enable algorithm development work to proceed while a more scaleable production backend is developed.

3. PIFF: PSFs in the Full FOV

write an introduction of why we need this: astrometric distortions -¿ WCS, coherent patterns over full FOV, Zernickes, better interpolation schemes

3.1. Gaussian Process Interpolation

Josh, Gary, Mike, Niall, Pierre-Francois, Ami: describe

3.2. Errors on Adaptive Moments

The wavefront-based PSF performs a χ^2 fit to the adaptive moments of stars. However, up to now we have not included an error on the adaptive moments in the fit. We calculated the errors on the adaptive moments, using the following expressions from simple error propagation:

$$\sigma^2(e_0) = \sum \left\{ \left[(u-u_0)^2 + (v-v_0)^2\right] K(u,v) \right\}^2 \sigma_I^2(u,v)$$

$$\sigma^2(e_1) = \sum \left\{ \left[(u-u_0)^2 - (v-v_0)^2\right] K(u,v) \right\}^2 \sigma_I^2(u,v)$$

$$\sigma^2(e_2) = \sum \{[(u-u_0)(v-v_0)]\, K(u,v)\}^2 \sigma_I^2(u,v)$$

where $\sigma_I(u,v)$ is the shot noise on an individual pixel and $K(u,v)$ is the kernel used by the HSM algorithm. Note that we have not yet included a contribution from the error on the centroids, nor from the (omitted) normalization term.

To test this calculation, we used an ensemble of simulated stars, constructed with arbitrary aberration, and with shot noise appropriate for different choices of star S/N. For a stellar flux of 10^5 photo-electrons, we created 5000 noisy stars, calculated their adaptive moments and errors, and plot the pull distribution for e_0, e_1 and e_2 (note that our e_1 and e_2 are unnormalized) in Figure 1. The pull distributions have RMS somewhat larger than 1., and we find very similar RMS values independent of flux for values from 10^6 down to 4×10^3. We suspect that the errors are underestimated because we omit the contribution to the error from the uncertainty in the centroids and the normalization.

3.3. Combining PSF Models

A goal with PIFF is to model the PSF as a combination of the optical and atmospheric PSFs. Because these two PSFs have significant differences in implementation, they need to be combined together within PIFF. Chris wrote a new `PSF` class, `CompoundPSF`, while at the lighthouse. This class can take an arbitrary number of PSFs, and fits each piece iteratively. The final PSF is a convolution of the PSFs.

4. Shape measurement

quick intro of lessons learned from DES Y1

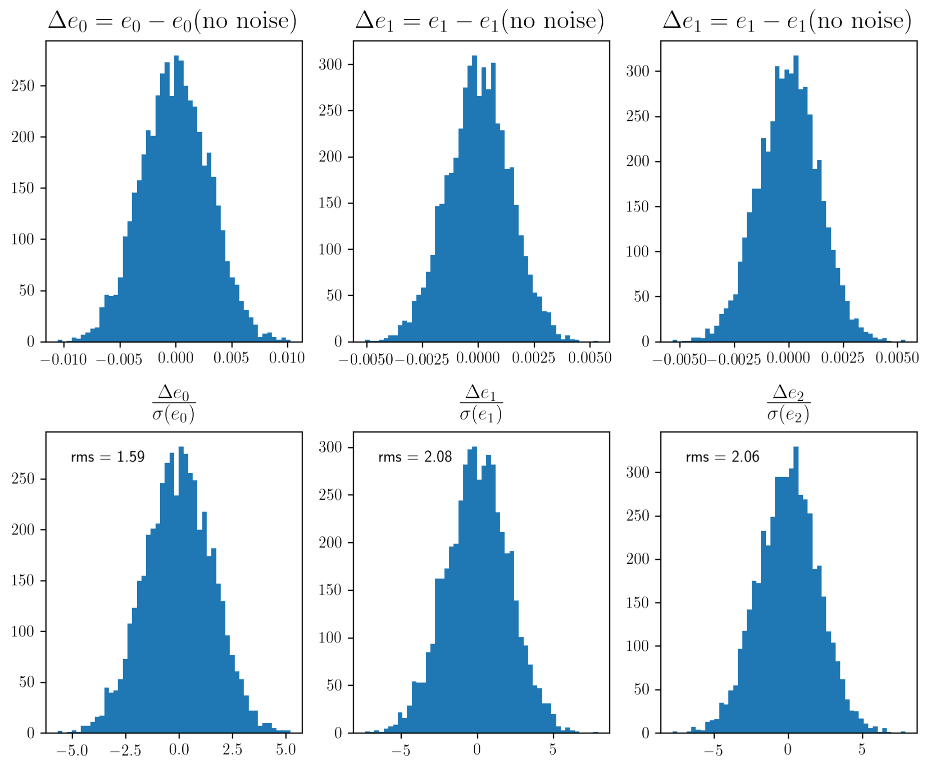

Fig. 1.— The pull distributions for e_0, e_1 and e_2 from a sample of simulated stars with $S/N = 316$. The errors are underestimated by a factor which is independent of S/N.

4.1. BFD on real data

The BFD approach to galaxy shape measurement, described in Bernstein & Armstrong (2014) and implemented in Bernstein et al. (2016), uses a collection of moment templates derived from deep data to generate a posterior for galaxy shapes in shallow data.

We have made progress connecting the BFD code to the MEDS format used in DES and LSST.

The two components missing for this were

- a variant of `simpleImage` (see `momentcalc.py` in the BFD repository) that can take multiple postage stamps of the same galaxies with their respective WCS registrations (in the form of the position of a centroid estimated in WCS and transformed to the postage stamp pixel system) and Jacobians, PSF models, and an estimate of the overall centroid in WCS **(Katie)**
- a function that can get these inputs to the new variant of `simpleImage` from a MEDS file

(using the python `meds` or a derived class) **(Daniel)**

A git repository was created at `https://github.com/danielgruen/bfdmeds` containing code to use the MEDS library to provide the input PSF information needed by BFD, and to wrap the existing MEDS objects with the new astrometric information provided by the improved WCS measurement process and stored in YAML files. The library was used with DES Y3 meds files to produce the PSF image below.

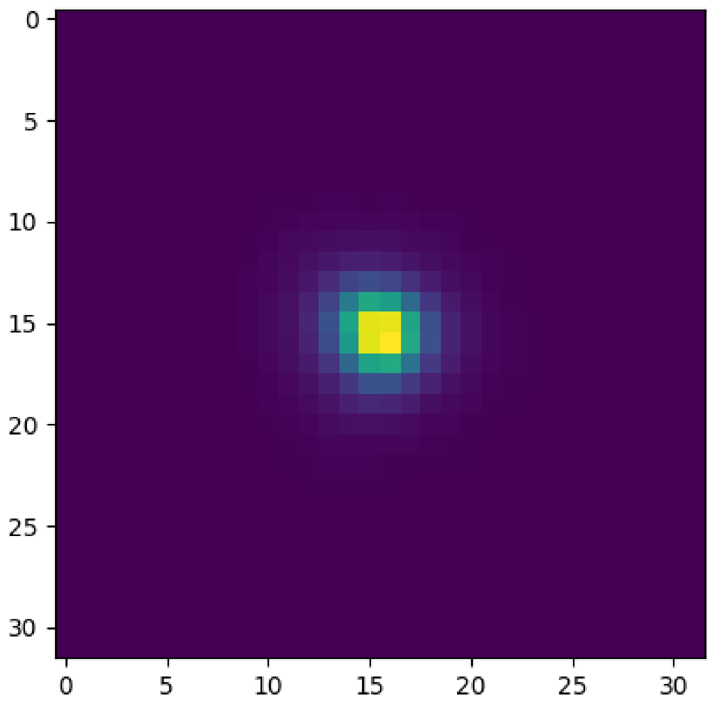

Fig. 2.— A PSF image generated from DES Y3 PSFEx models from the new bfdmeds repository.

5. An image simulation pipeline for PSF and shape measurement validation

Joe, Mike, Erin, Troxel, Gary, Daniel, Niall, Ami, Katie: describe

5.1. Goals

Goals: Simulation engines primarily for validation. Win some Gin from Catherine Heymans.

The subsections below describe stages of this process and the options or considerations for it.

5.2. (Statistical) Requirements

We need to define cosmology-driven requirements for the simulation, as these will set the simulation volume requirements and thus the computing requirements etc.. Particularly important to do this if we e.g. need to find/apply for more computing resources.

One simple approach to setting requirements is to assume we only need to estimate a mean multiplicative bias, $\langle m_i \rangle$ and additive bias, $\langle c_i \rangle$ per redshift bin i. Then from a simulation with input shear $\gamma_{\rm true}$,

$$\langle m_i \rangle = \left\langle \frac{\gamma_{\rm obs} - \gamma_{\rm true}}{\gamma_{\rm true}} \right\rangle , \tag{1}$$

where the (possibly weighed) averaging is over all galaxies in redshift bin i. It follows that

$$\sigma_m = \frac{\sigma_e}{\gamma_{\rm true}\sqrt{N_{{\rm gal},i}}} \tag{2}$$

i.e. for a given σ_m one can estimate the required number of simulated galaxies per redshift bin, $N_{{\rm gal},i}$. For postage-stamp style image simulations which provide effective tests of effects like noise bias, galaxy bias and PSF deconvolution, this kind of approach is probably sufficient, and can be used to set the number of postage-stamps required. We believe these types of simulation are still useful as a first test of shear estimation methods, see the *tide-pool* simulation mode below.

When it comes to testing our ability to estimate unbiased shear statistics on real data, there are many more effects to consider. Even if a shape measurement method can succeed on postage-stamp style simulations selection effects, PSF modelling errors and crowding/blending will produce scale dependent biases on shear statistics. For example, excluding blended galaxies produces a bias in the small-scale cosmic shear signal (blended galaxies are more likely to be in high projected galaxy number density, high convergence regions (Hartlap et al. 2011; MacCrann et al. 2017)). Meanwhile for tangential shear statistics such as cluster-lensing, source galaxies which are close to the lens centres are likely effected by different noise properties and worse contamination from blending.

A more complete way of setting the requirements on the image simulations is simply to aim to be able to test the recovery of any shear statistic we estimate on the the real survey data, to some fraction of the statistical uncertainty with which we can estimate that statistic on the real survey data. Assuming shot/shape-noise is the dominant source of statistical error, any shear statistic (that is reliably reproduced in the simulation) can be tested at a precision which is a fraction f of the statistical error by producing $N_{\rm sim} 1/f^2$ simulated copies of the real survey data.

We envisage two main modes of the image simulations. See Figure 3 for schematics.

1. *Tide-pool* Fast, straight to postage-stamps simulations where different marine organisms (e.g.

complex morphology, PSF model errors) are turned on and off. Galaxies written straight to MEDS files, or are generated on-the-fly. No neighbours.

2. *Pacific* Full single-epoch survey images are rendered. Coadd images are produced for detection/deblending. PSF estimation is performed on the SE images. MEDS files produced using SE images and coadd information. Some shortcuts probably needed!

Discussion points:

- What should $N_{\rm sim}$ be? Note that most of the statistics we use are cosmic variance limited on large scales, so beating down the shape noise may not be so relevant. We will know the 'cosmic variance' (or rather we'll know the particular realisation of the cosmological signal) in the simulation. Having said that, if they effects we are most worried about (blending etc.) are worst at small scales, then shape noise would be the dominant source of statistical uncertainty.
- How can we reduce $N_{\rm sim}$? One option would be to boost the shear signal - the stronger the simulated signal, the better fractional precision can be tested. However, concerns exist about higher order shear biases, and breaking the relation between galaxy number density and shear. For postage-stamp styles simulations, presumably we can use ring-test type tricks to reduce shape noise. Is there something similar we can use for the detection simulation e.g. rotate shape noise between realisations?
- What kind of shear fields should we use? If we use something realistic (e.g. from ray-tracing), how do we test that we're recovering it correctly, given that a given method will only use a subset of the galaxies?

Several main initial tasks:

1. Finish writing up this plan. Produce (NM+)
2. Design simulation software framework (JZ+)
3. Design and implement input galaxy catalog $->$ galaxy appearance modules (can start v. simple, and produce multiple of these!)
4. Estimate timings + memory usage for simulation steps. Could split this into several tasks e.g. image rendering timing (i.e. mostly GalSim stuff) (Name?), Coadding/Detection (i.e. SWARP + SExtractor) timing (Name?).
5. Estimate computing requirements for various simulation modes (this should be straightforward once the previous step is completed).

5.3. Required Ingredients

5.3.1. Input Catalogues

Requirement: Galaxy sample with physically sensible clustering, redshift evolution, morphologies, colours, and correlations between the above. Sufficiently complex morphologies to test model bias, and sensible variation in morphology between filters.

- Starting from an observed catalog is problematic: measured quantities get noisy at the faint end, we won't get sub-detection objects.
- Proposal: Start from N-body + galaxy simulation e.g. BCC.
- Map simulation outputs (e.g. color, size) to image properties in each filter (e.g. bulge/disc, size, amount of star formation knots). Look into Lanusse method.
- Needs to go to high enough redshift and depth for deep fields (and for sufficient sub-detection objects in wide-field).

5.3.2. Shear Field

The following options, probably we'll want to do one of the first two for the Pacific simulation. Other options fine for tide-pool simulation.

- From saved N-body results (see above)
- Spatially varying but without evolution within a redshift bin
- Spatially varying but without redshift evolution
- Constant

5.3.3. Survey Details

- Real data pointings
- Exposure times - from real survey
- Noise levels - from real survey
- Sky background - from real survey
- Deep data - Same process but with more exposures

5.3.4. True Astrometry

- Flat WCS
- Real image WCS
- Gary's WCS
- FITS header WCS
- One of the above + some error distribution

5.3.5. True PSF

- Estimated real ones from Piff
- Additional complexity, adding variation on smaller scales than
- Fixed values
- Colour-dependence of PSF. Full implementation 100 times slower. Could make it a function of a single colour parameter, linearly interpolated. Effective PSF from real SED? Function of (g-i). Intra-band resolution of PSF. Do this at n wavelengths. Like using a chunky SED.

5.3.6. Star Catalogue

- Gaia?
- Randomly, function of latitude.
- SEDs - some in galsim already, need more?

5.3.7. Artifacts

- Tape bumps
- CTI
- Brighter-fatter
- Non-linearity
- Non-convolutional things
- Image artifacts

5.3.8. Masking

- Real
- Real + artifacts

5.3.9. Single-epoch Rendering

- GalSim

5.3.10. Coadding

- Run full pipeline - i.e. run swarp
- Shortcut option: draw coadd directly. What does this miss? What is required to test this?

5.3.11. Detection & Segmenting

- Needed for lists of detected objects and segmentation masks
- Run full pipeline - SExtractor on coadd
- Generate object list and seg map from truth catalogs

5.3.12. Estimated PSF

- Find starting stars - on coadd using SExtractor? Can we do this without spread-model?
- Run PIFF
- Use truth

5.3.13. Estimated astrometry

- Cannot re-do the actual astrometry process
- Use the original FITS ones
- Use truth + some error term

5.3.14. Photometric Calibration

- Could place a small error on the truth - scale objects by some factor.

5.3.15. MEDS

- Make a MEDS file and run on it!

6. `Metacalibration` Response for Stars

Testing to see if we can cause positive responses for stars in a simulation. In some scenarios we see $< R >\sim 0.25$ in real data.

6.1. Variations in PSF

Consider the case where the PSF model used is accurate in the mean but for an individual galaxy the truth varies significantly. The response for stars at high S/N(> 100), and using forward modeling, is shown in figure 4. The same for adaptive moments, without PSF correction. The response has mean about 0.1 for forward modeling, but nearly zero for adaptive moments. In both cases the distribution is nearly gaussian, whereas in real data it tends to be highly asymmetric, with mode at $R \sim 0.5$.

Acknowledgments

We acknowledge financial and organizational support for our workshop from LSSTC and KIPAC and the hospitality of Hostelling International.

This is the text imported from `acknowledgments.tex`, and will be replaced by some standard LSST DESC boilerplate at some point.

REFERENCES

Bernstein, G. M., & Armstrong, R. 2014, MNRAS, 438, 1880

Bernstein, G. M., Armstrong, R., Krawiec, C., & March, M. C. 2016, MNRAS, 459, 4467

Hartlap, J., Hilbert, S., Schneider, P., & Hildebrandt, H. 2011, A&A, 528, A51

MacCrann, N., Aleksić, J., Amara, A., et al. 2017, MNRAS, 465, 2567

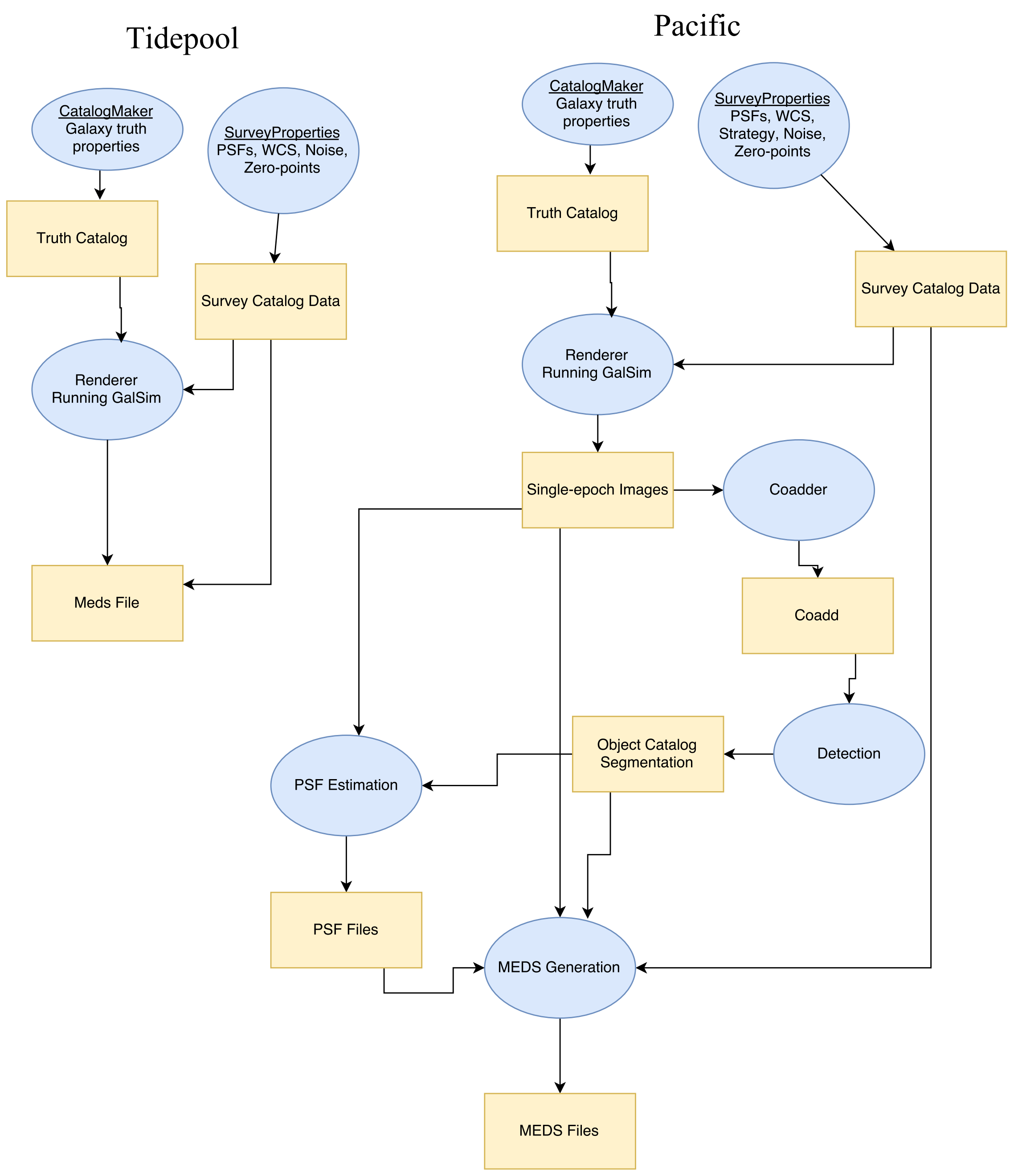

Fig. 3.— Flowcharts of the *Tide-pool* and *Pacific* scenarios for the image simulations.

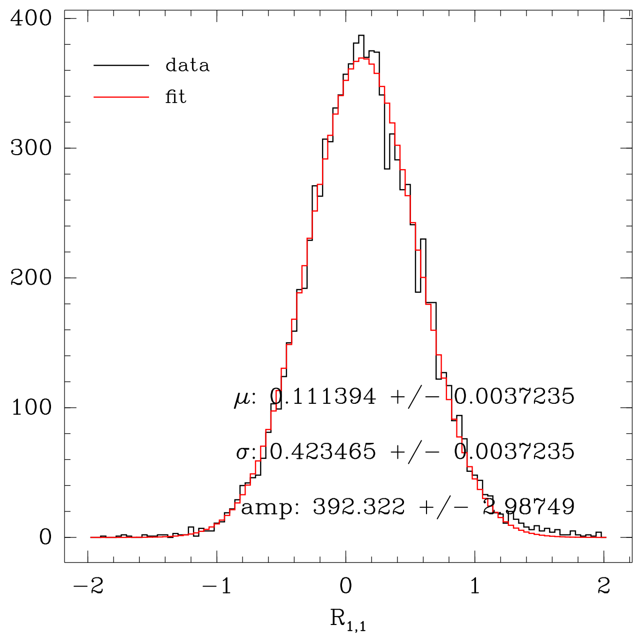

Fig. 4.— Response for high S/N stars when varying the PSF from object to object, but using the mean model when performing `metacalibration` operatioins, using the forward modeling estimator. Parameters of the best fit gaussian are shown.

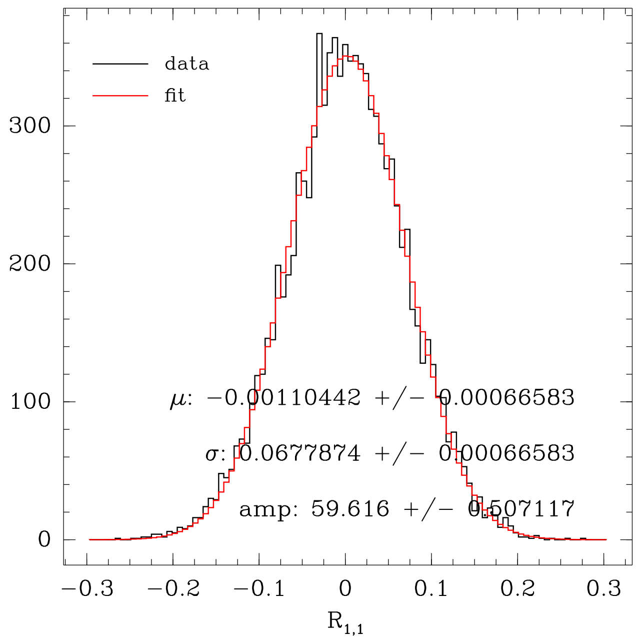

Fig. 5.— Same as figure 4, but adaptive moments estimator with no PSF correction