Making and using camera models with mrcal: the practical details

Dima Kogan

June 13, 2023

What is mrcal for?

mrcal is a generic toolkit, applicable to anything that makes or uses camera models.

Common applications:

- Calibration
- Triangulation
- Photogrammetry
- SFM

Why does mrcal exist?

I couldn't find a set of tools precise-enough to make my visual ranging work possible, so I wrote my own

By necessity, mrcal is a complete re-design and re-implementation of camera modeling tooling from the ground-up:

- Richer camera model available to precisely model lens behavior
- Uncertainty propagation and cross-validation techniques available to validate a calibration
- No implicit pinhole assumption anywhere (no homogeneous coordinates, essential, fundamental matrices, etc)
- Fisheye-friendly stereo rectification

Computing details

- mrcal is a library (C and Python)
- Many commandline tools available, so no coding required for common tasks
- *Thoroughly* documented
- Open-source, available in stock Debian

Contributions welcome!

Where is it?

Detailed documentation is available at
http://mrcal.secretsauce.net/

- This talk is a repackaging of the documentation on that page
- See the docs for more detail, and for links to all the data and commands that produced the presented results

Demo calibration

Let's start by looking at the "tour of mrcal":
http://mrcal.secretsauce.net/tour.html

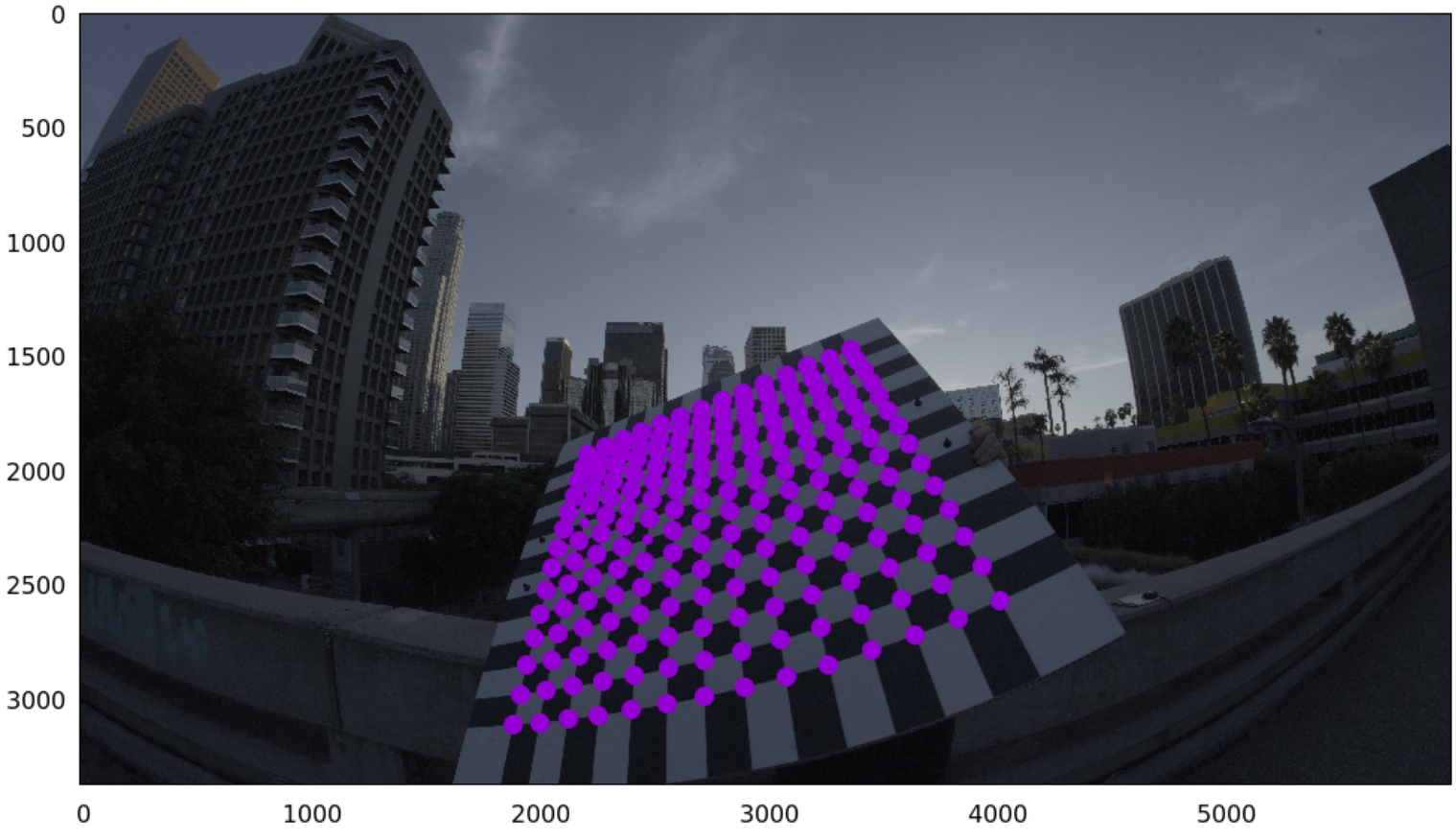
We follow a real-world data flow, starting with chessboard observations. Images captured using

- Sony Alpha 7 III full-frame SLR. 6000x3376 imager
- *Very* wide lens: Samyang 12mm F2.8 fisheye. 180deg field of view corner-corner
- Just one camera (in this demo; mrcal supports multiple cameras)
- Outdoor images captured in downtown Los Angeles

Sample image of the scene

Gathering and detecting chessboard corners

We capture images, and use mrgingham to detect chessboard corners. For an arbitrary image:



Let's run a calibration!

This is a wide lens, so we need a lens model that can handle it.
Let's use the 8-parameter OpenCV model: `LENSMODEL_OPENCV8`

```
$ mrcal-calibrate-cameras --lensmodel LENSMODEL_OPENCV8 ..
...

RMS reprojection error: 0.4 pixels
Worst residual (by measurement): 1.8 pixels
Noutliers: 564 out of 16464 total points: 3.4% of the data
calobject_warp = [-0.00012726 -0.00014325]
```

Why is the reprojection error > 0?

We have two sources of error:

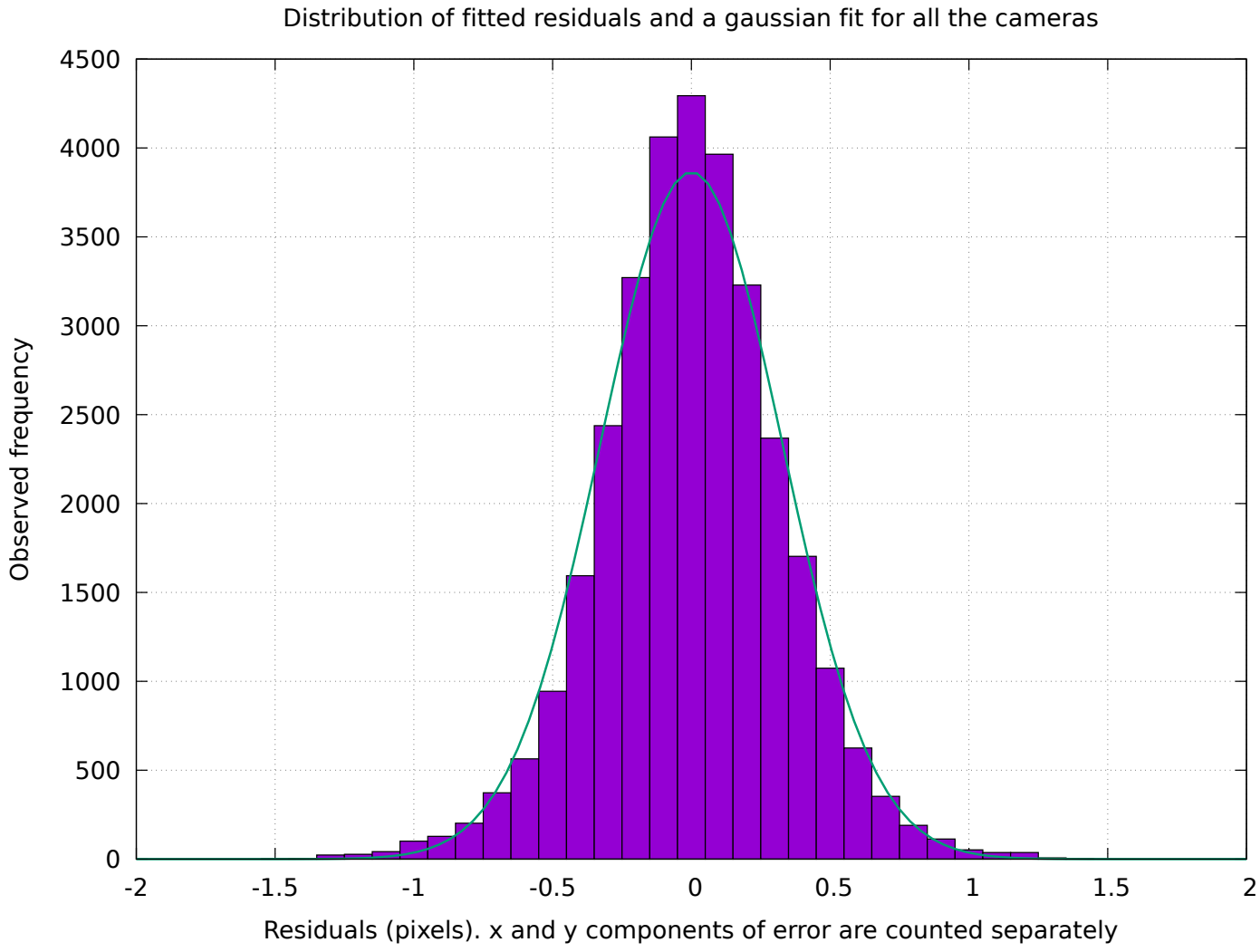
- Sampling error: chessboard observations are noisy. We can study this, but we cannot reduce it
- Model error: our model of the lens behavior, chessboard shape and everything else isn't perfect. We can and must suppress this as much as possible

We want the model error to be negligible. If it is and if the sampling error is normal and i.i.d., then we get a bias-free maximum-likelihood calibration result

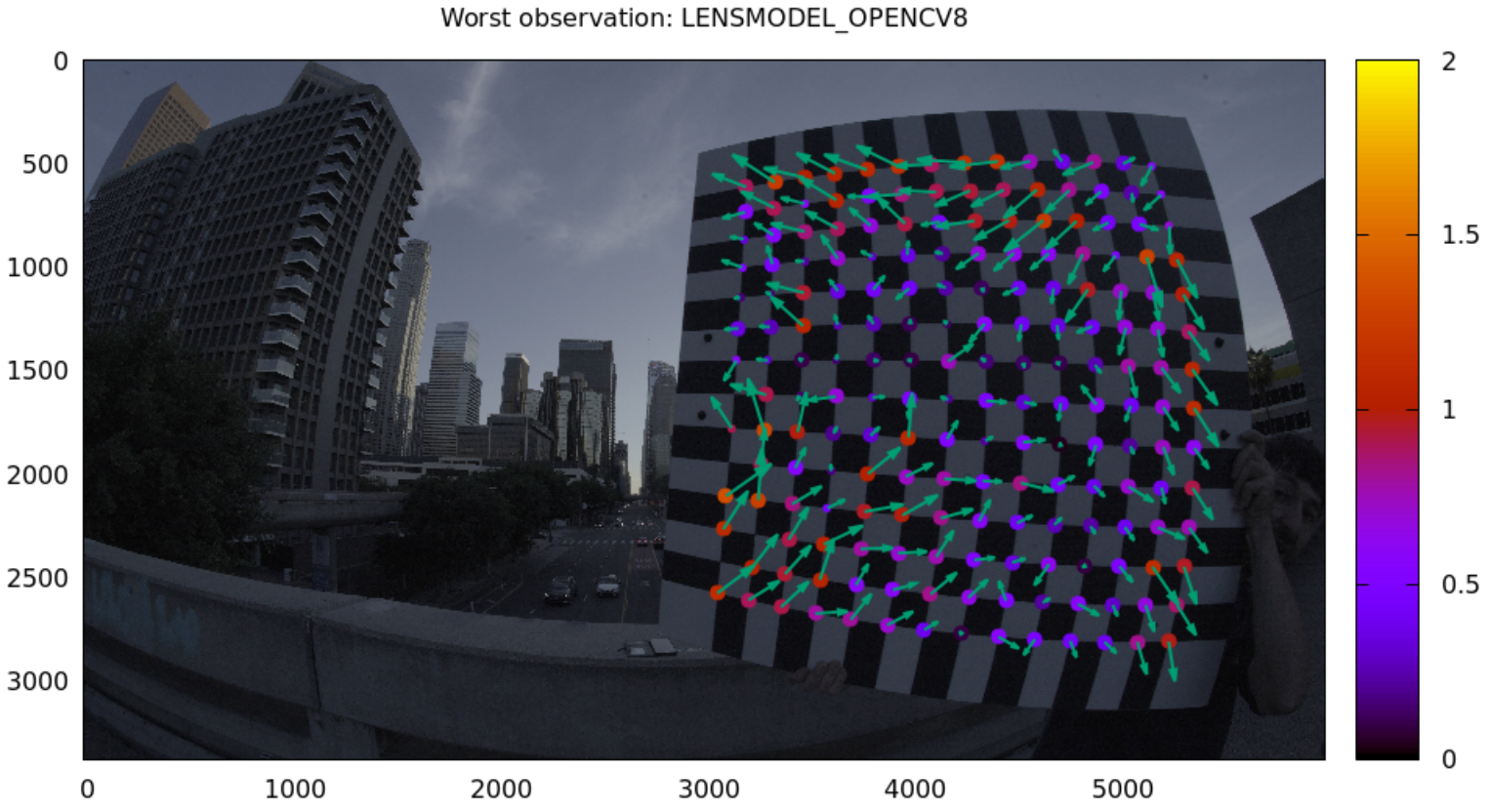
Patterns in the residuals indicate the presence of model errors

LENSMODEL_OPENCV8 residuals histogram

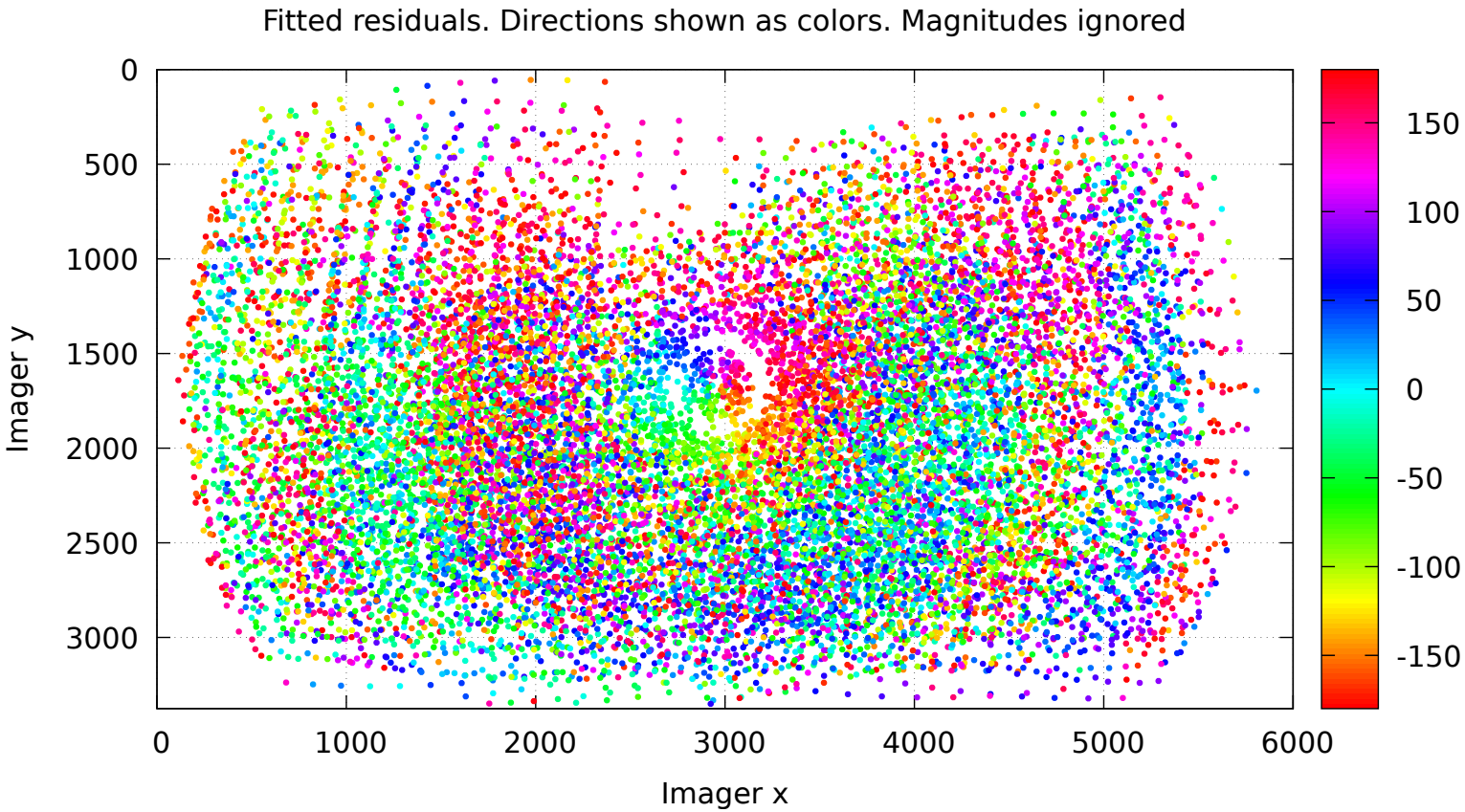
What does the error distribution look like?



LENSMODEL_OPENCV8: the worst image



LENSMODEL_OPENCV8: residual directions



LENSMODEL_OPENCV8: conclusions

We see clear patterns in the residuals, so

- LENSMODEL_OPENCV8 does not fit our data

Let's fix it.

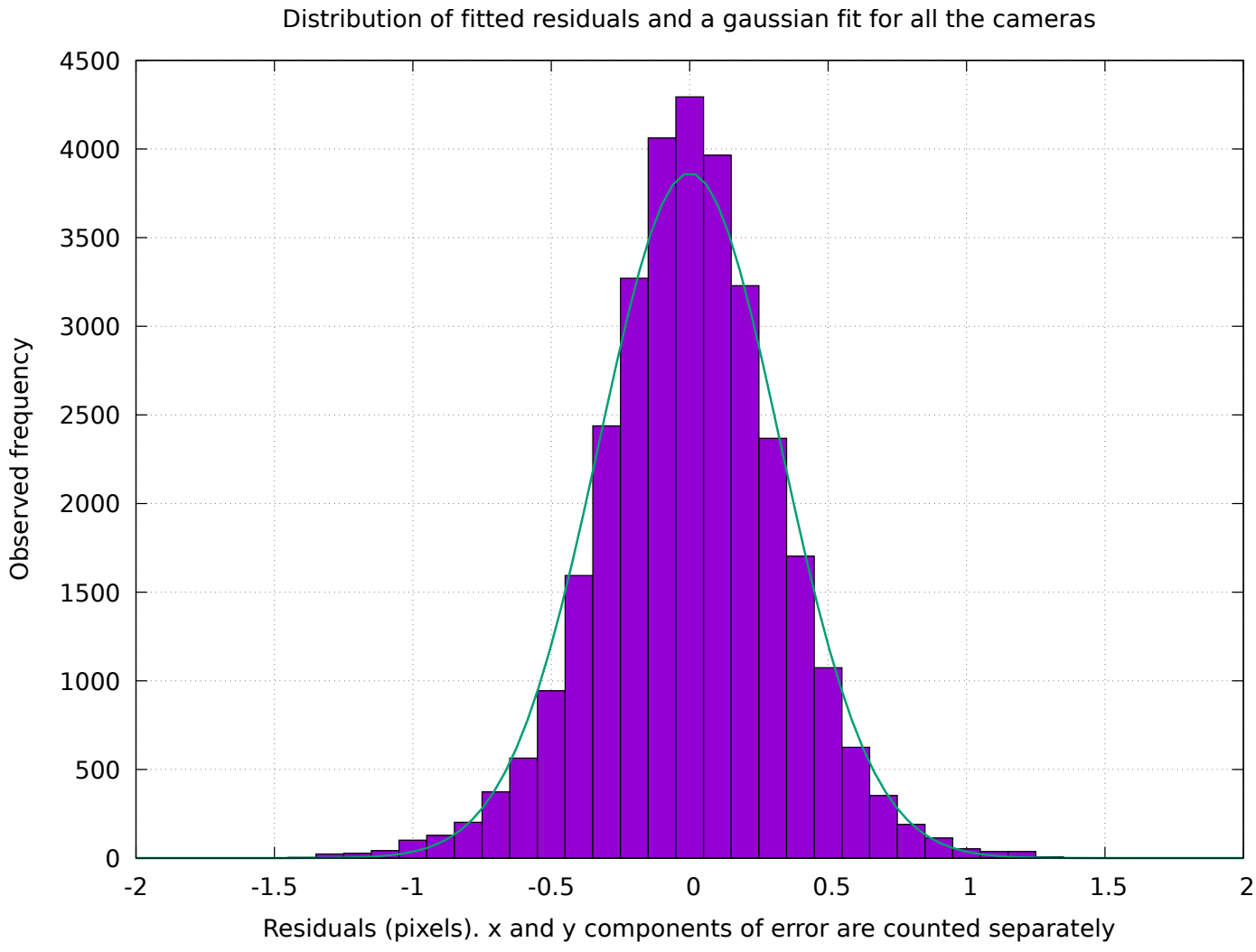
Doing it again with LENSMODEL_SPLINED_STEREOGRAPHIC

Let's re-process the same calibration data using the splined model. We run the same command as before, but using the `LENSMODEL_SPLINED_STEREOGRAPHIC_ ...` `order=3_Nx=30_Ny=18_fov_x_deg=150` model. This is one long string.

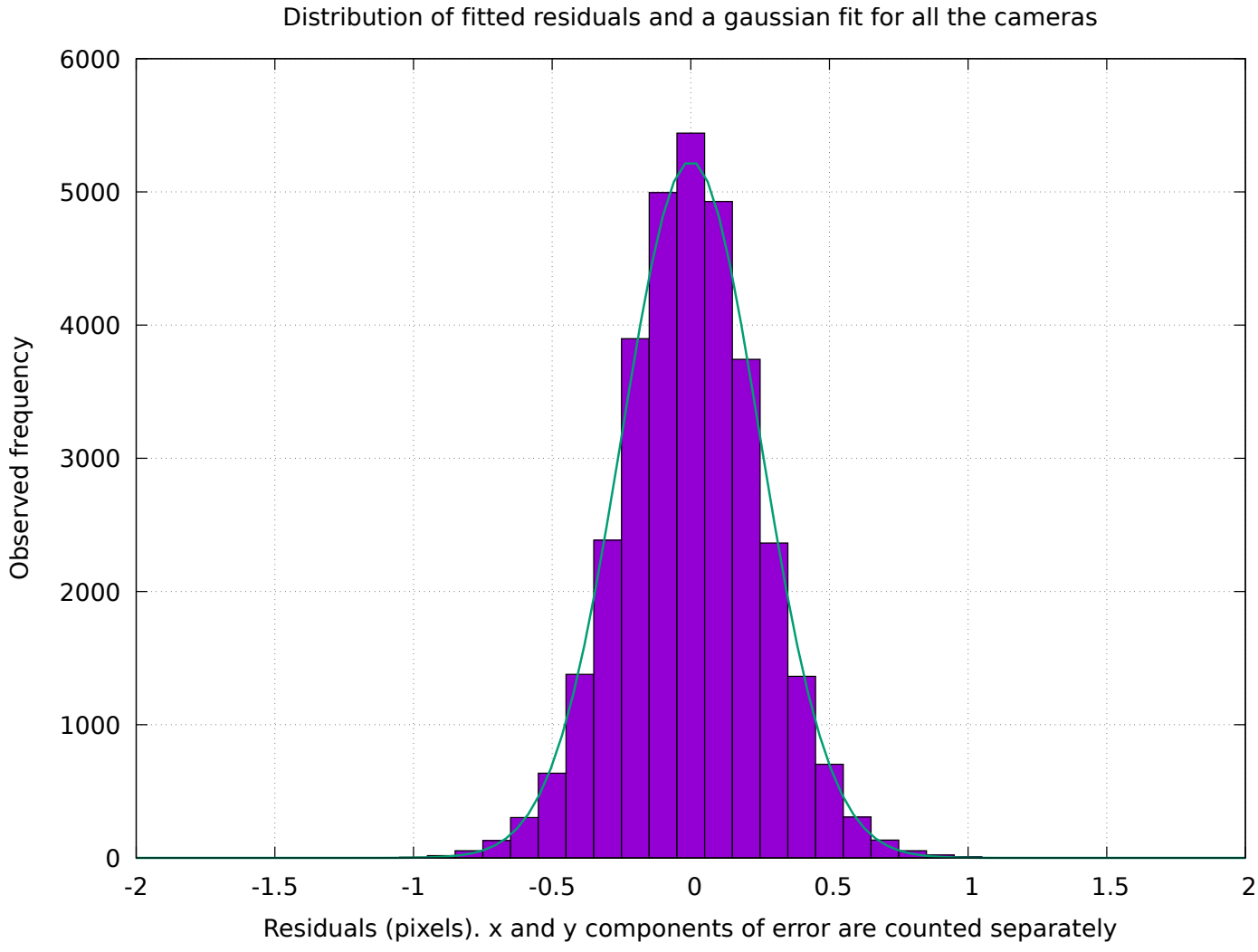
This model has 1084 parameters.

```
$ mrcal-calibrate-cameras
    --lensmodel LENSMODEL_SPLINED_STEREOGRAPHIC_ ...
    ... order=3_Nx=30_Ny=18_fov_x_deg=150 ...
...
RMS reprojection error: 0.2 pixels
Worst residual (by measurement): 1.3 pixels
Noutliers: 28 out of 16464 total points: 0.2% of the data
calobject_warp = [-1.26851438e-04 -8.03269701e-05]
```

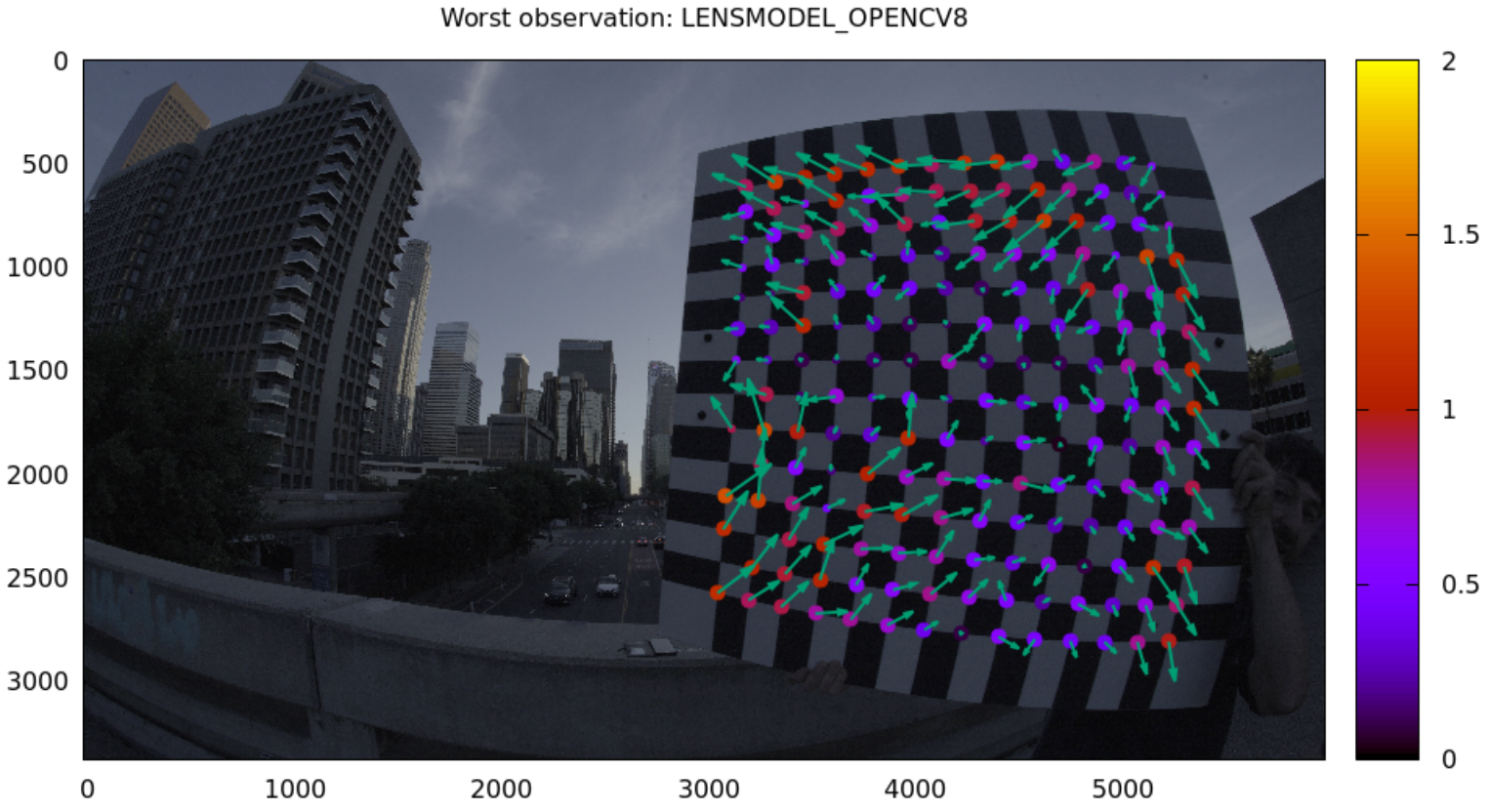
LENSMODEL_OPENCV8 residuals histogram



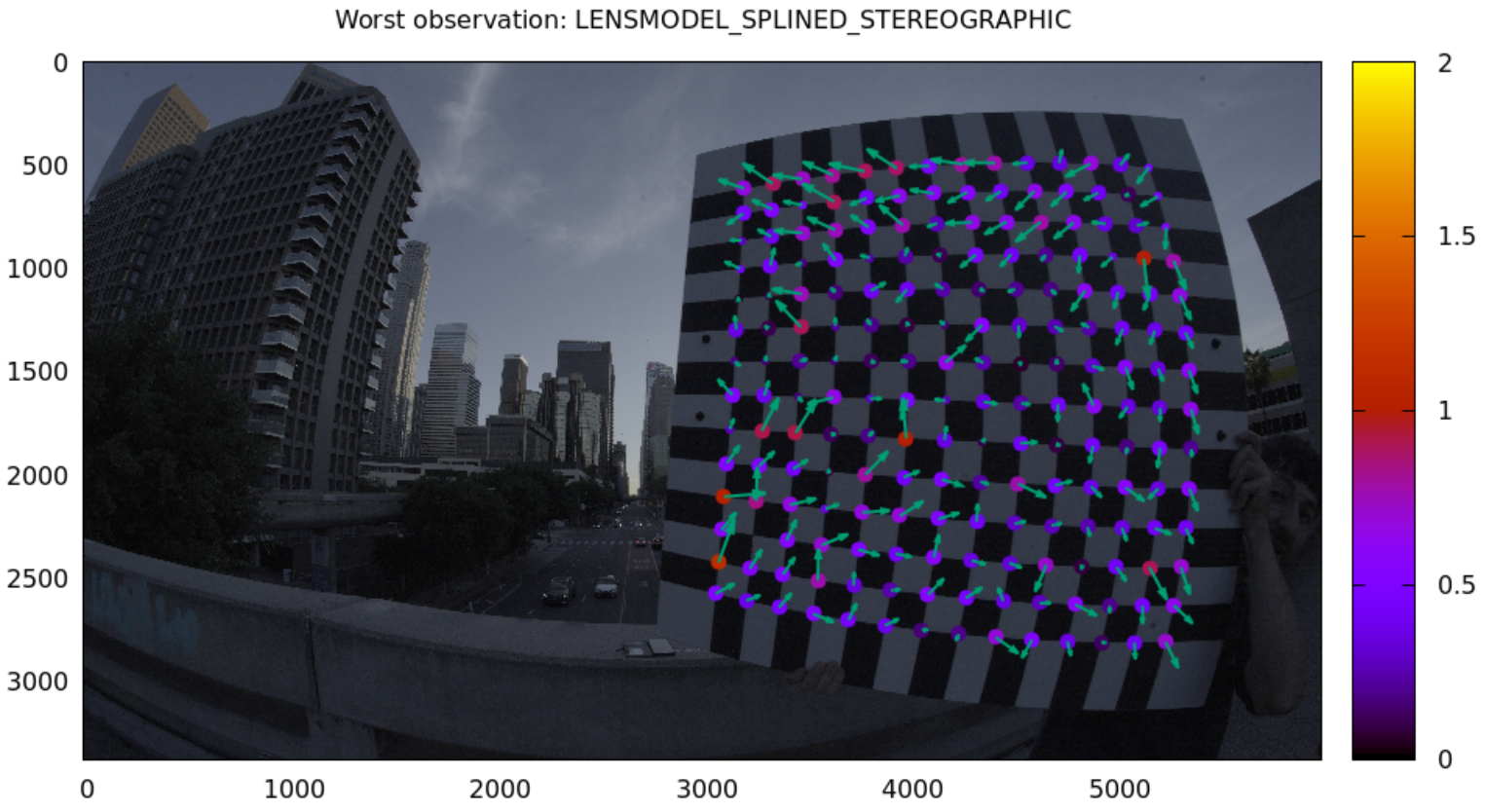
LENSMODEL_SPLINED_... residuals histogram



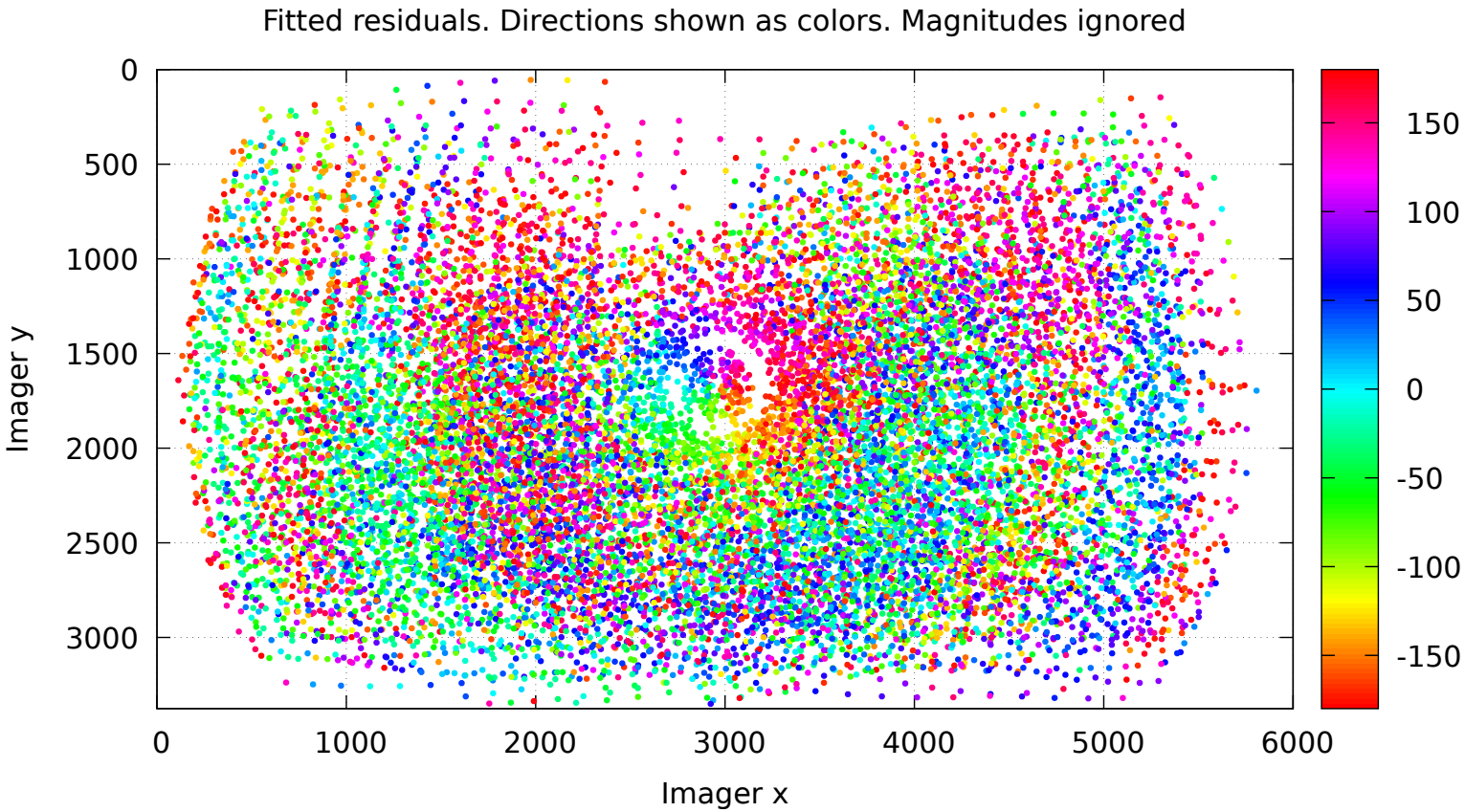
LENSMODEL_OPENCV8: the worst image



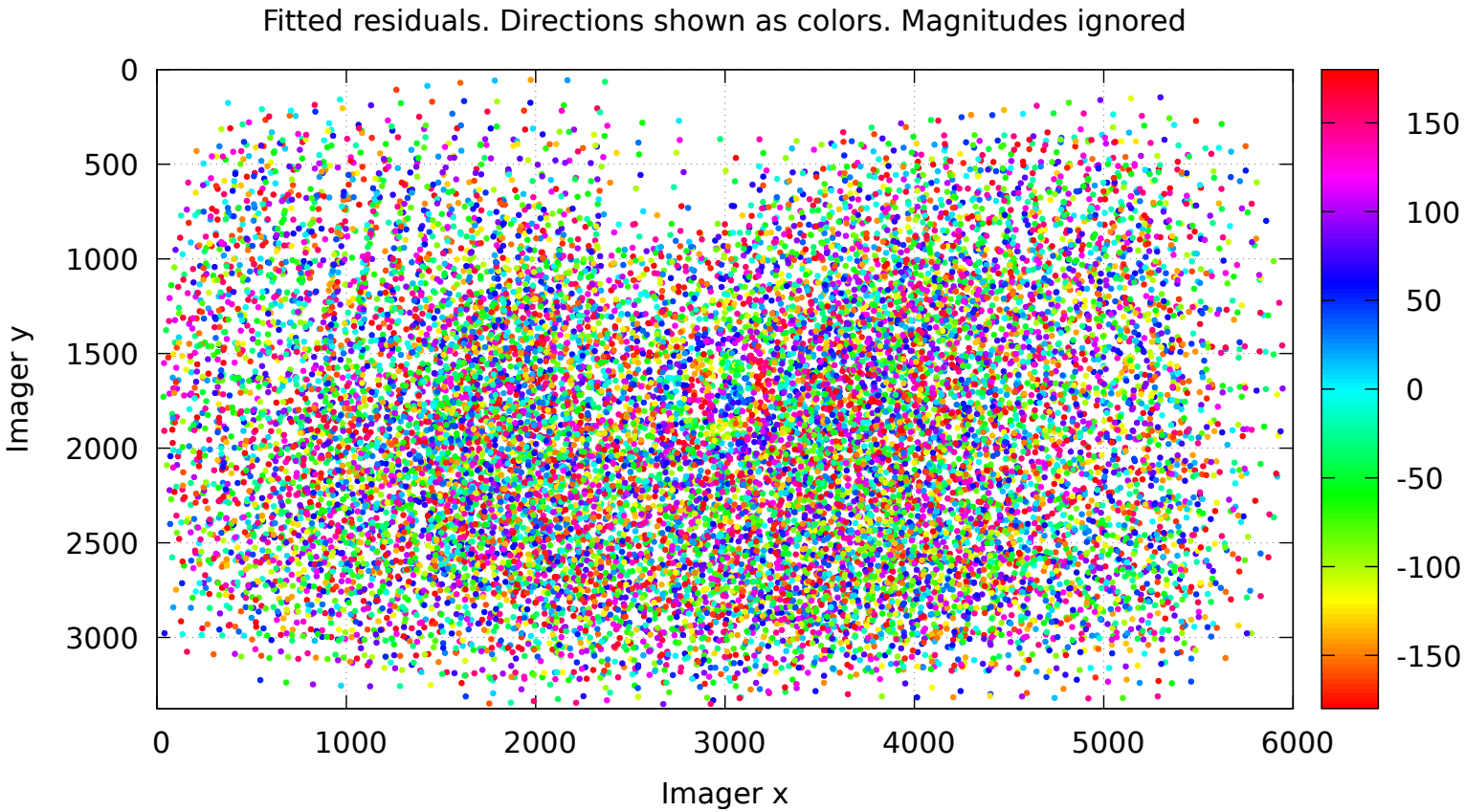
LENSMODEL_SPLINED_...: the worst image



LENSMODEL_OPENCV8: residual directions



LENSMODEL_SPLINED_...: residual directions



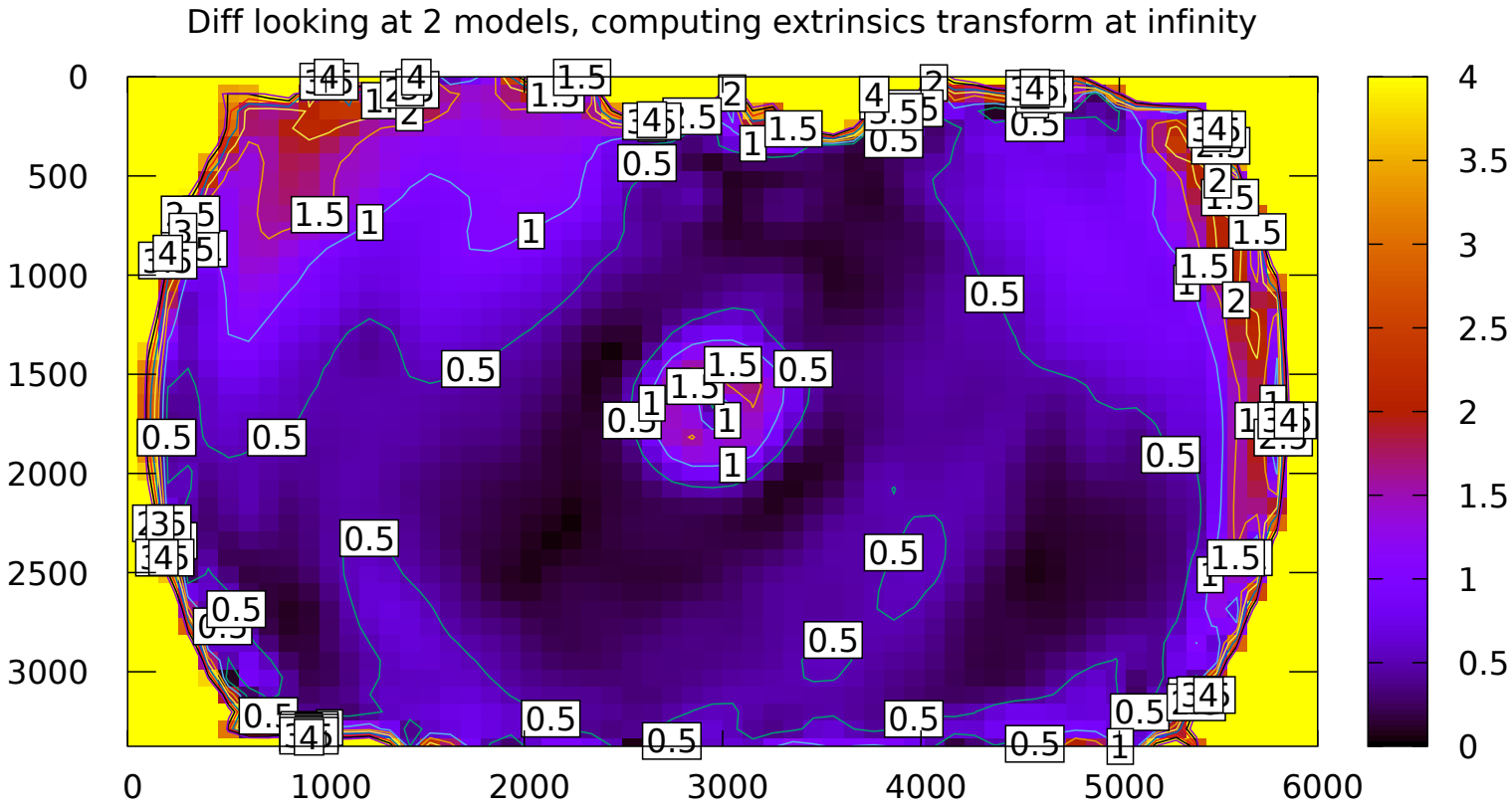
Conclusion

We have good evidence that `LENSMODEL_SPLINED_STEREOGRAPHIC` fits this lens much better than `LENSMODEL_OPENCV8`

Differencing

We computed the calibration two different ways. How different are the two models?

Differencing



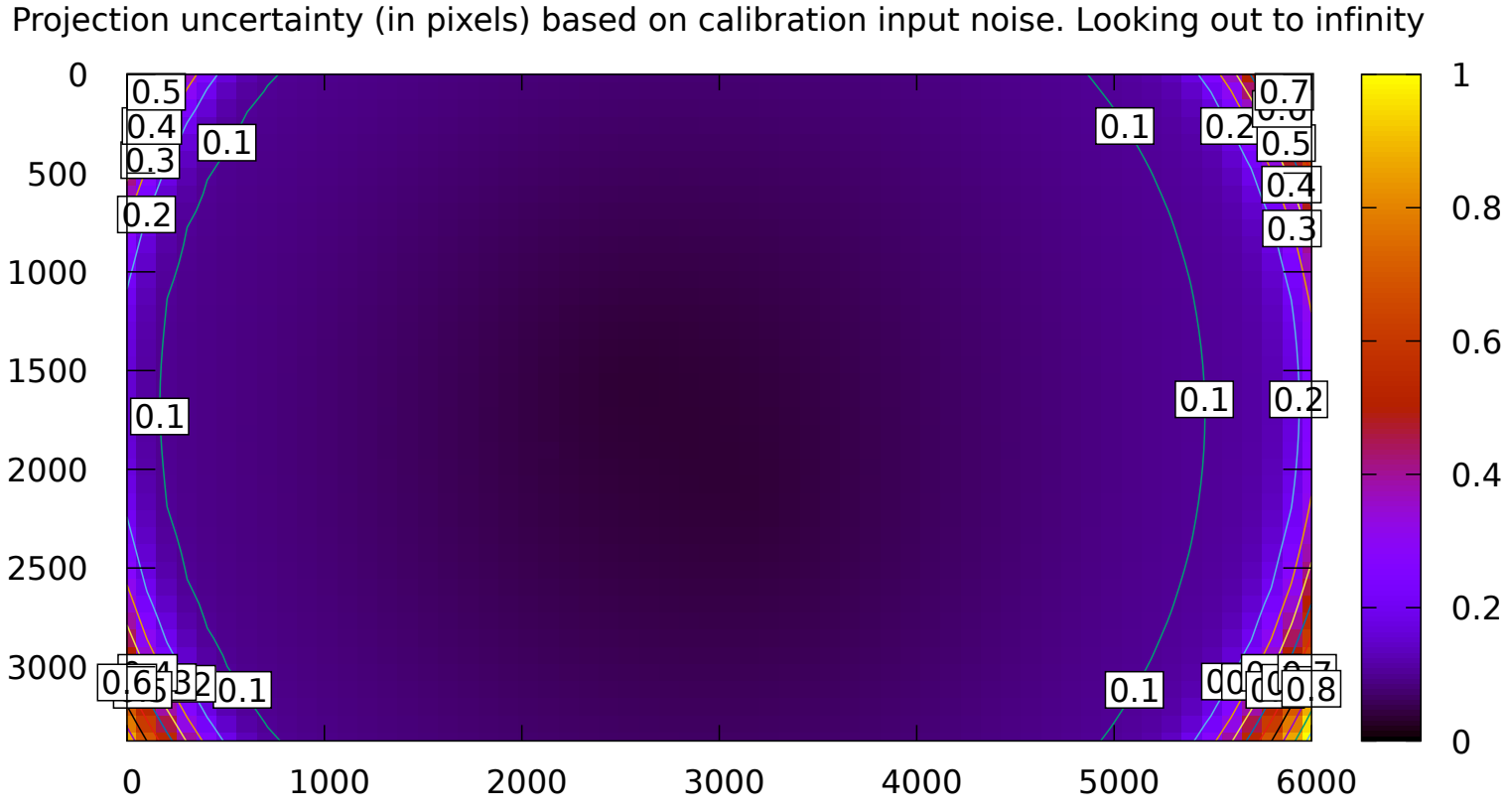
Uncertainty

- All calibrations are based on observations of the calibration object (chessboard corners)
- These observations *always* contain some noise (sampling error)
- A calibration result is trustworthy *only* if it is insensitive to this noise

We quantify this sensitivity by computing a projection uncertainty

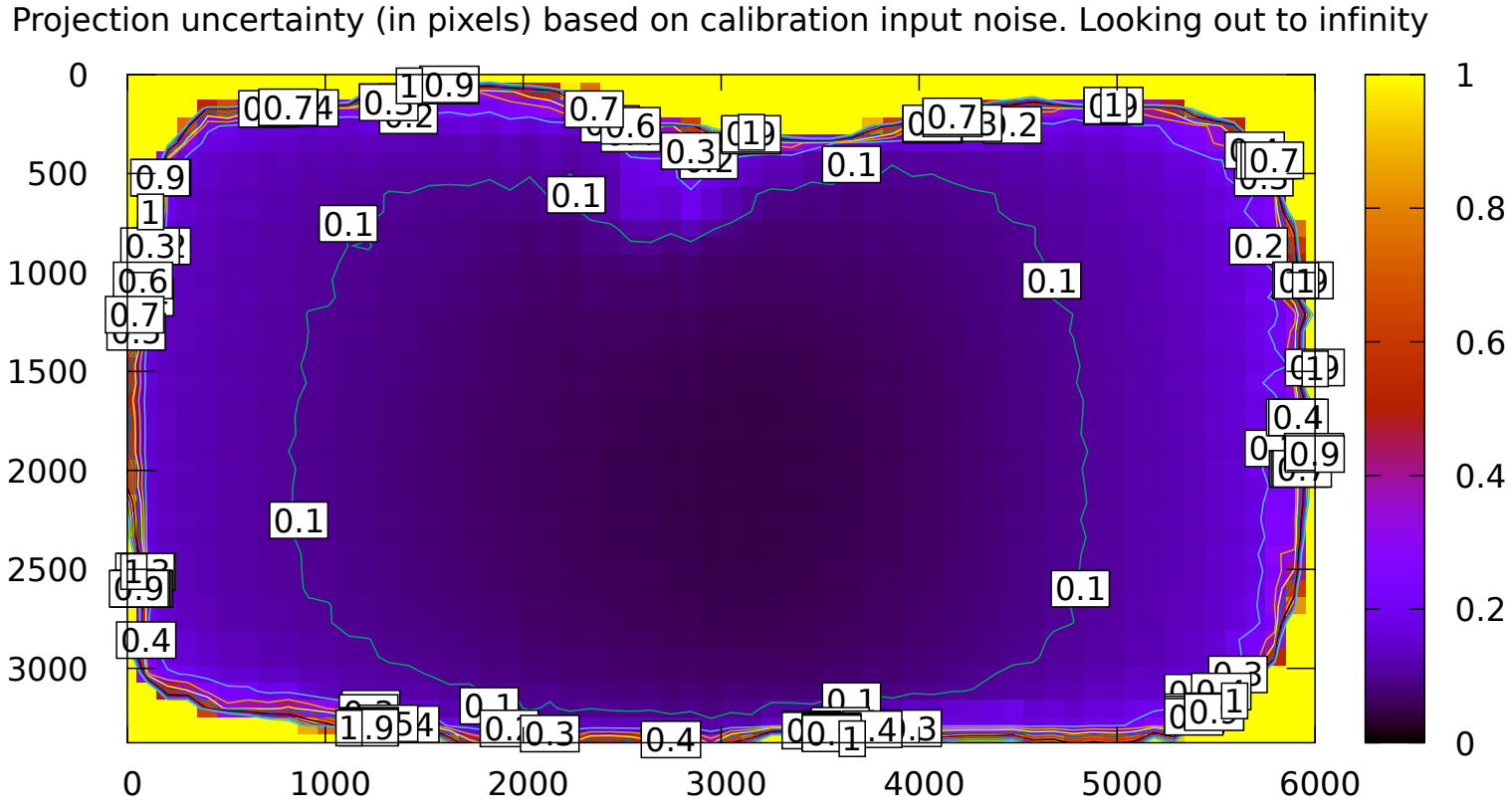
Uncertainty from the DTLA data

Computing the uncertainty map from the earlier `LENSMODEL_OPENCV8` calibration:



Uncertainty from the DTLA data

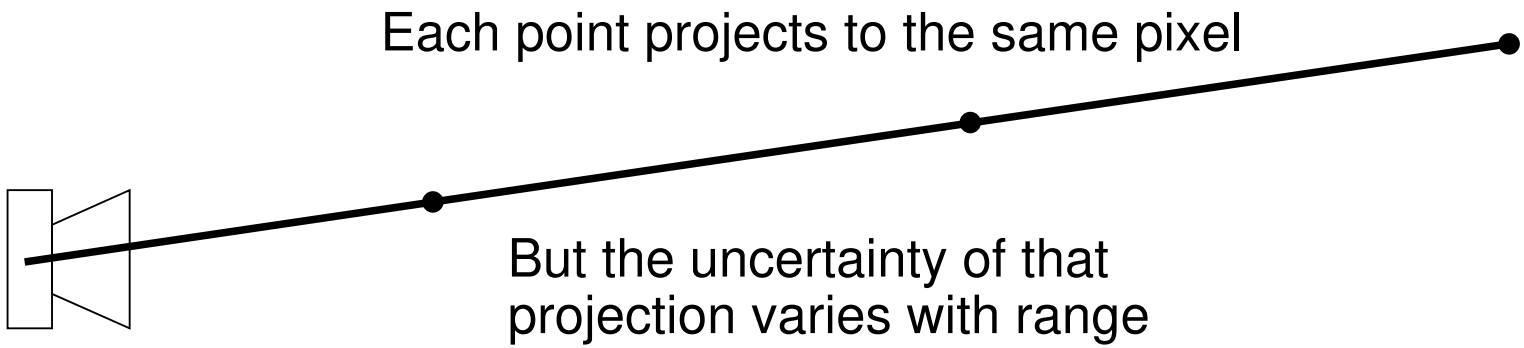
And from the `LENSMODEL_SPLINED_STEREOGRAPHIC_...` calibration:



Ranging note

Let's revisit an important detail I glossed-over when talking about differencing and uncertainties. Both computations begin with $\vec{p} = \mathrm{unproject}\,(\vec{q})$

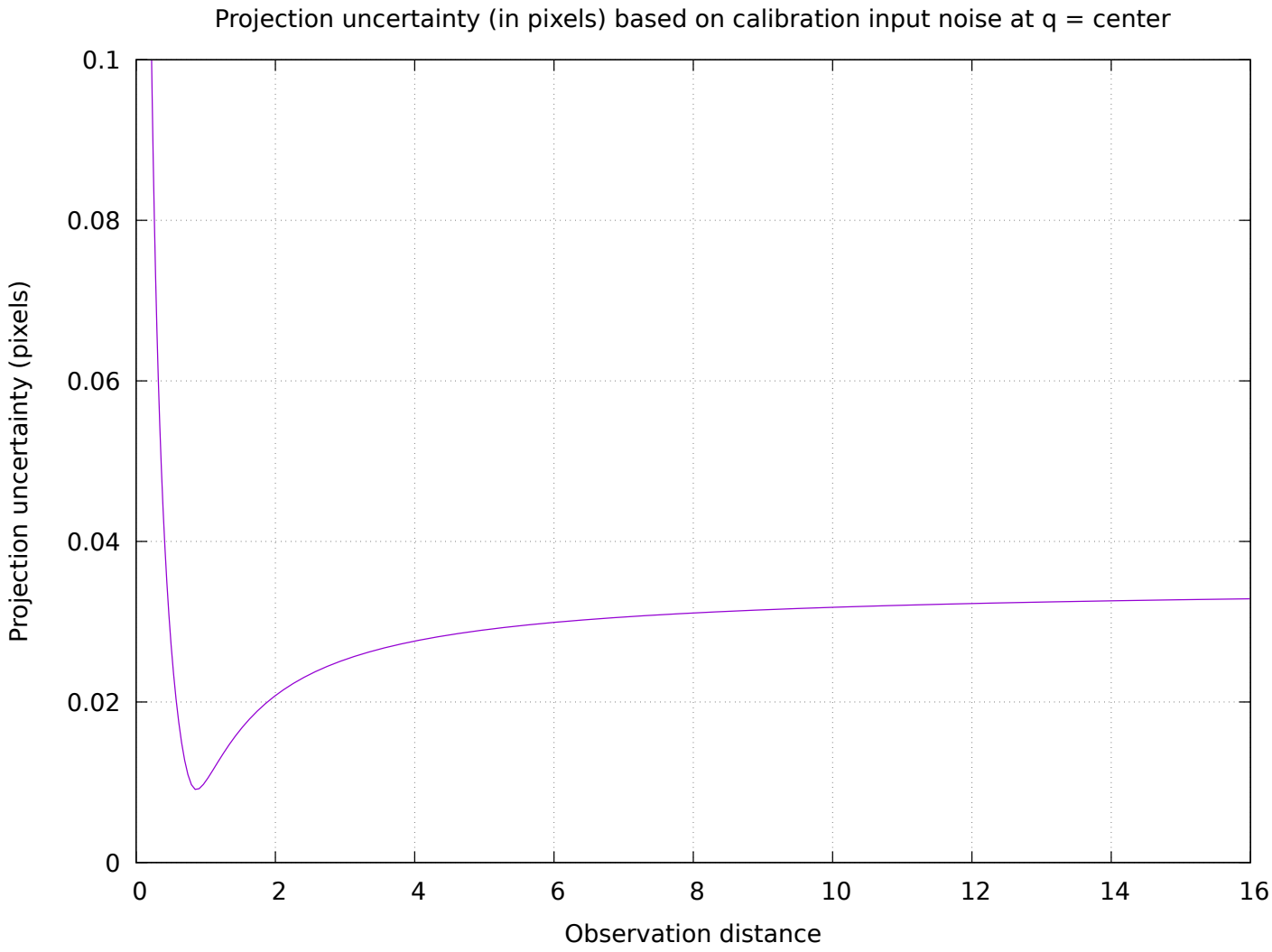
But an unprojection is ambiguous in range, so diffs and uncertainties are defined as a function of range



All the uncertainties reported so far were at ∞

The uncertainty figure

The uncertainty of our `LENSMODEL_OPENCV8` calibration at the center as a function of range:



Let's apply these techniques

We described several analysis techniques:

- Visualizing the solve residuals
- Computing projection differences between two models
- Evaluating projection uncertainty

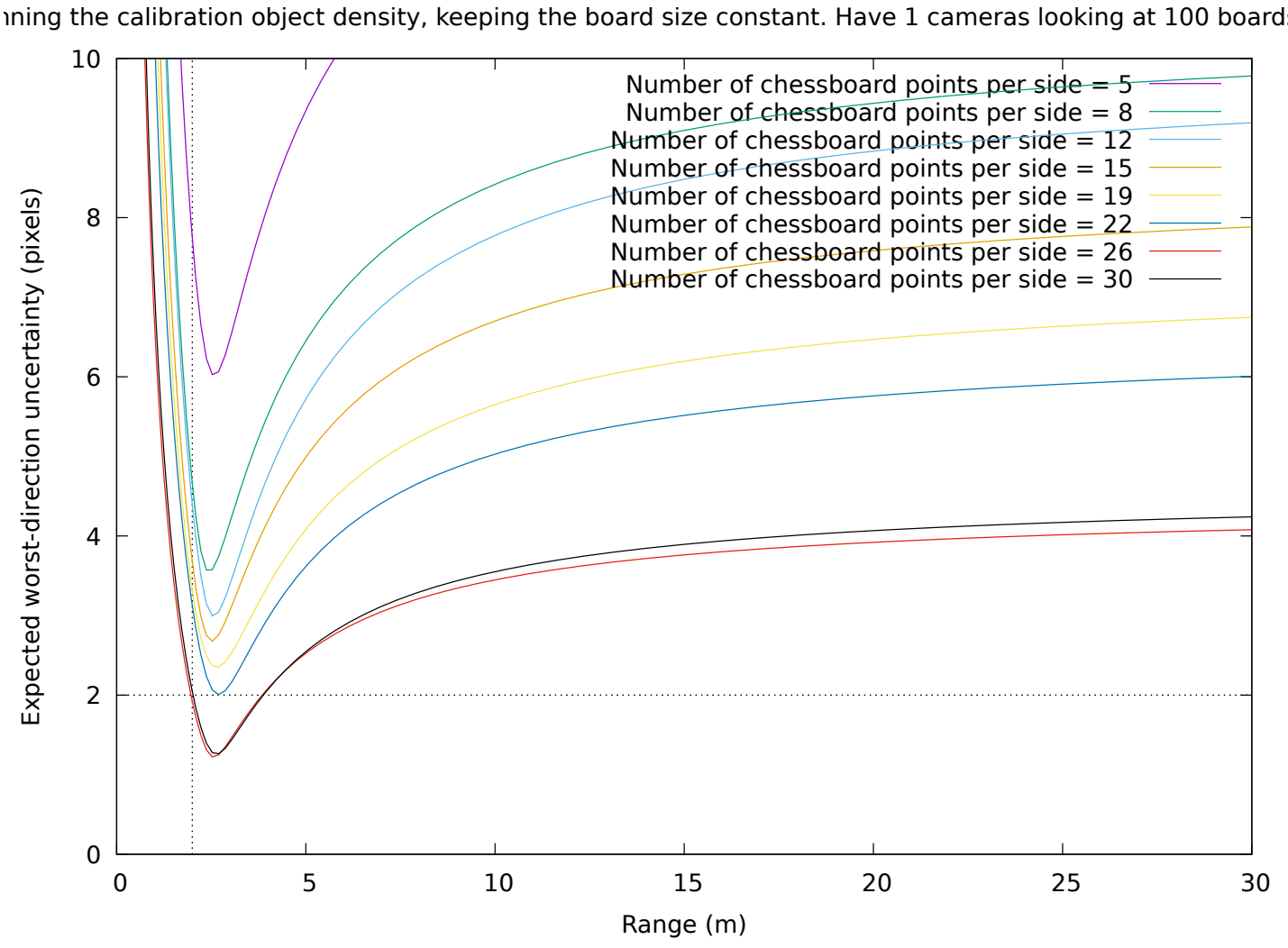
Let's use these to answer practical questions

Optimal choreography overview

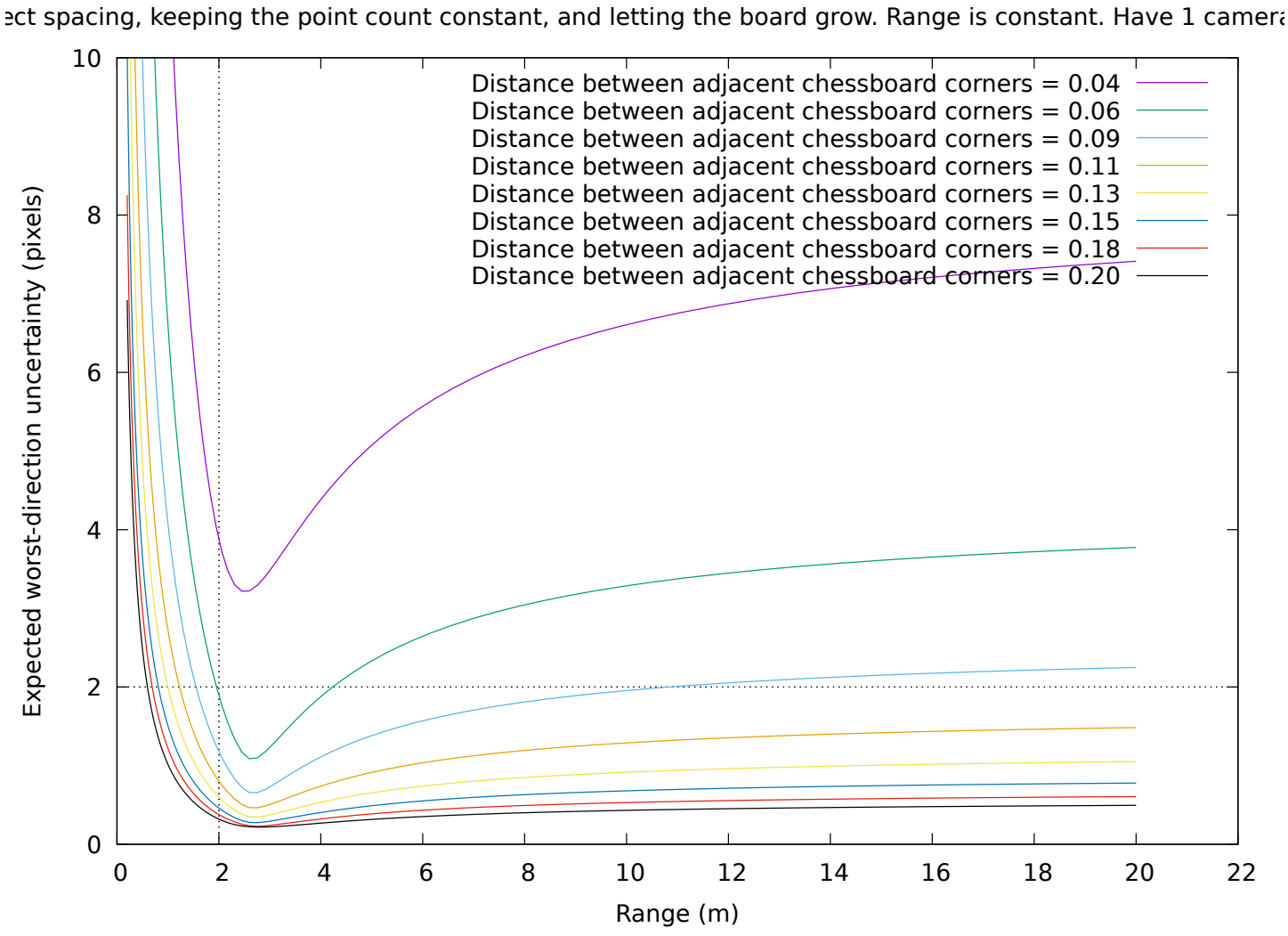
For many of the following analyses we study the effects of sampling error. We

- Set up a simulated world with some baseline geometry
- Scan some parameter
- Calibrate
- Look at the uncertainty-vs-range plots as a function of that parameter

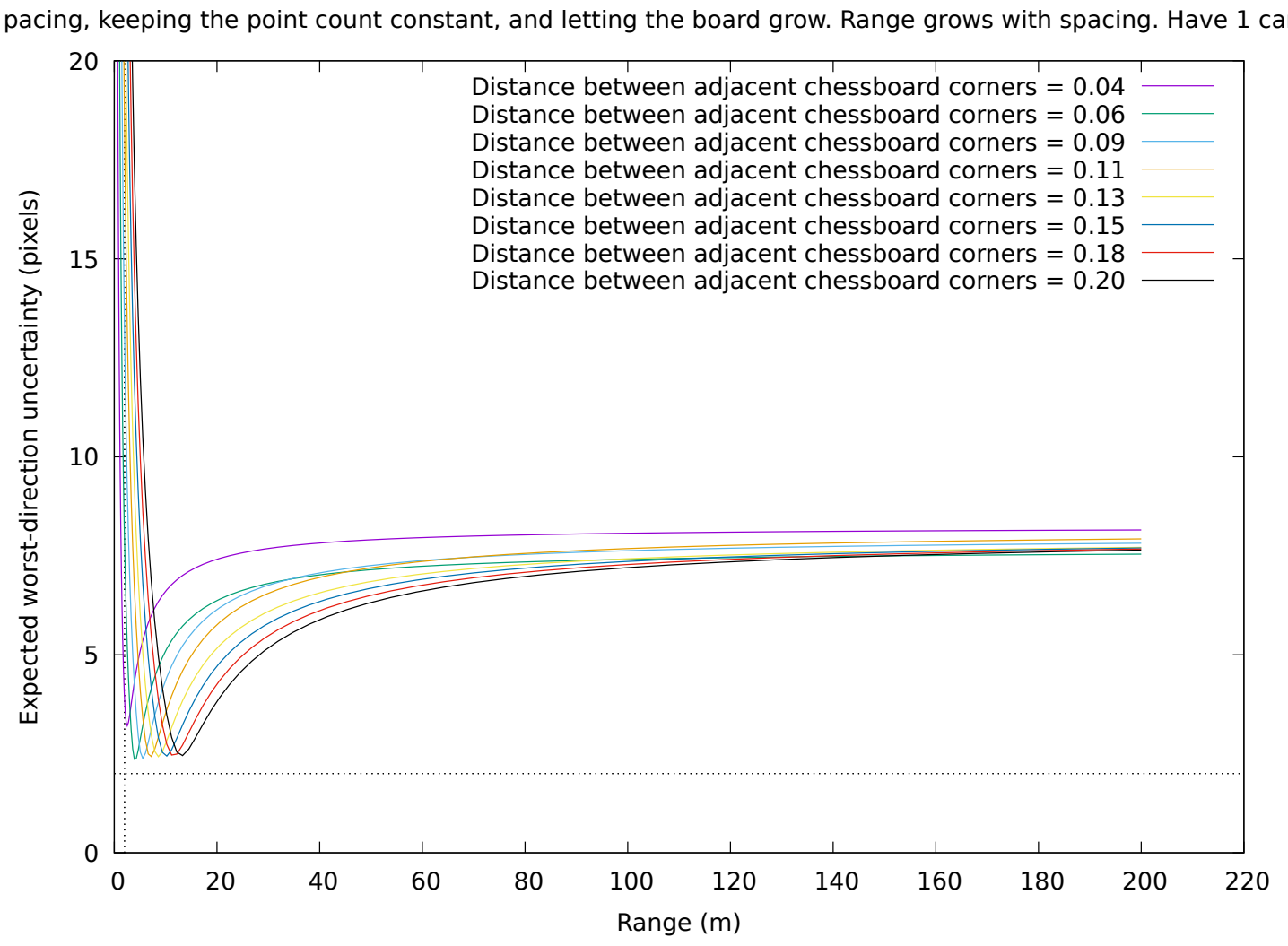
How dense should our chessboard be?



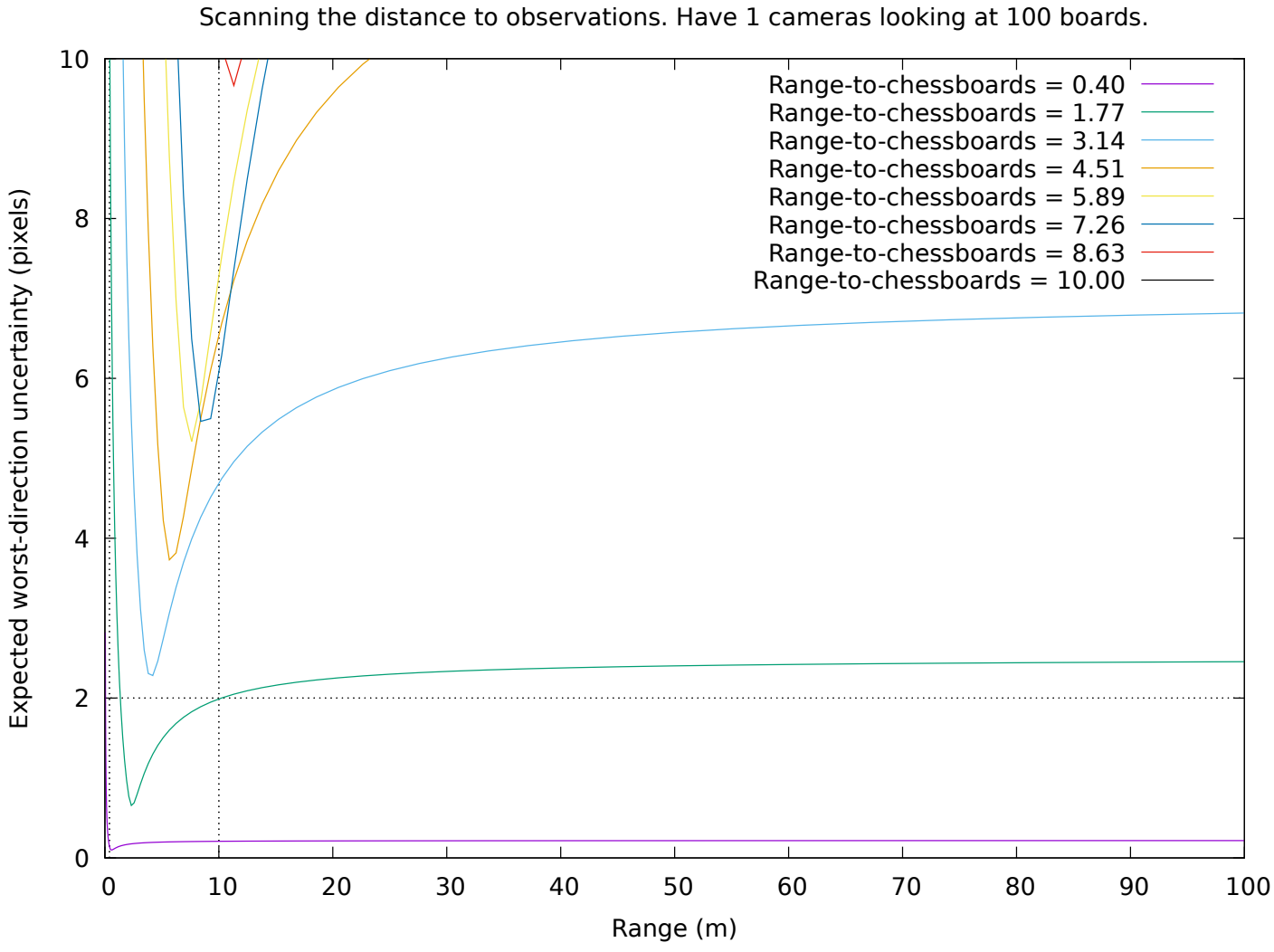
What should the chessboard corner spacing be?



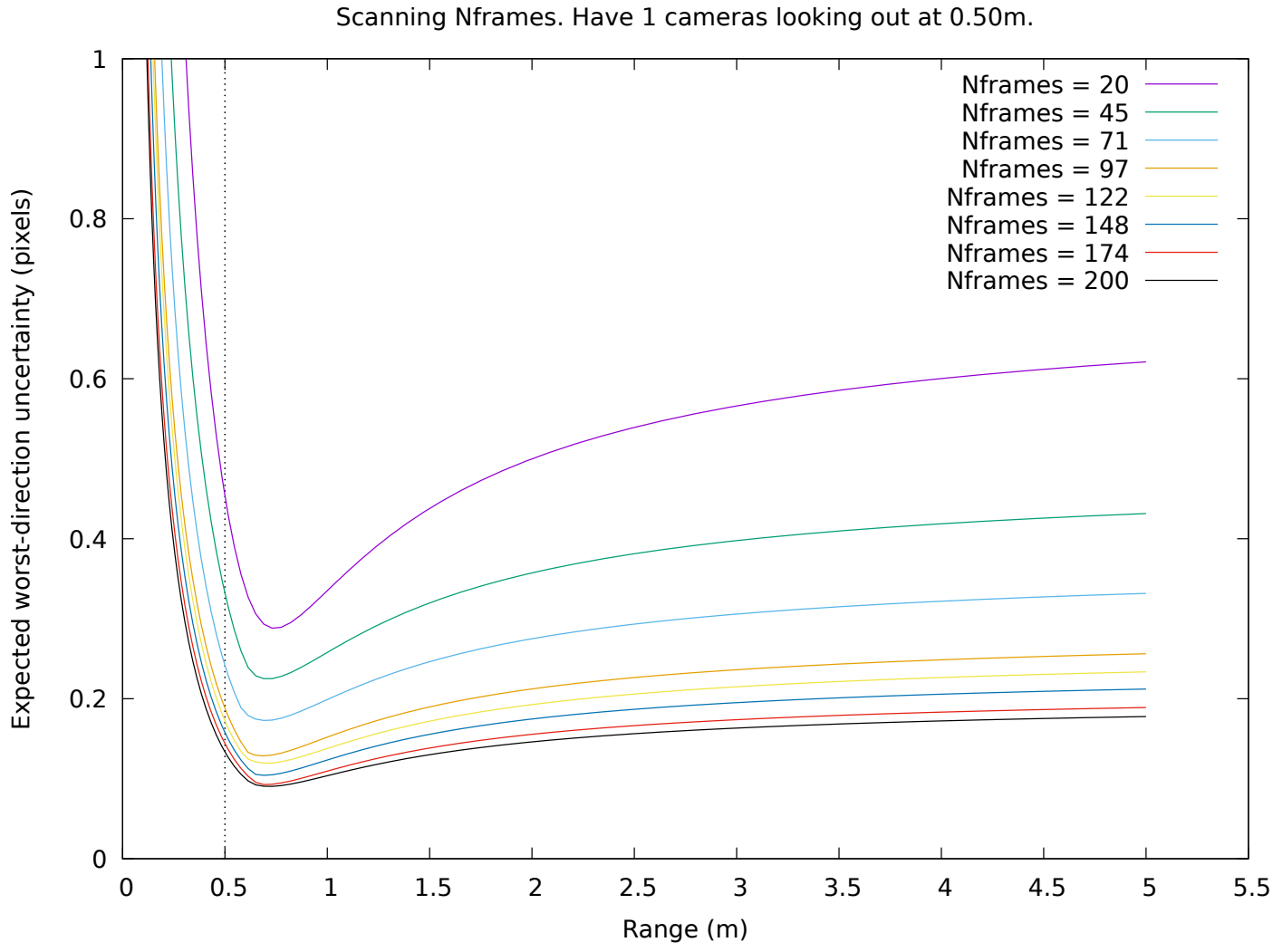
Do we want tiny boards nearby or giant boards faraway?



How far should the chessboards be placed?



How many chessboard observations should we get?



What kind of calibration object do we want? Guidelines

- More data is good
  - More chessboard corners
  - More chessboard observations
- The chessboard should fill the imager
  - Close-ups
  - Big chessboards

Questions:

- So what kind of calibration object do we want? Are *chessboards* the right choice?
- Should we place the chessboard immediately in front of the lens? Should we use a *giant* chessboard?

Chessboards? Circles? AprilTags? Charuco?

mrcal doesn't care!

- Grids of circles (and possibly AprilTags) don't directly observe the point, so they could be biased. mrcal has a visual validation tool: `mrcal-reproject-to-chessboard` that produces a validation sequence
- Anything with AprilTags needs a high-resolution-enough image to resolve the AprilTag. This resolution could instead be used to cram extra chessboard squares into the image

I use chessboards with the mrgingham detector

The downsides of extreme closeups

Corners out of focus

- If the blur is unbiased and gaussian: this will increase the noise, but we can compensate by gathering more data
- It looks like the blur mostly *is* unbiased and gaussian, but don't push it

Noncentral effects become significant

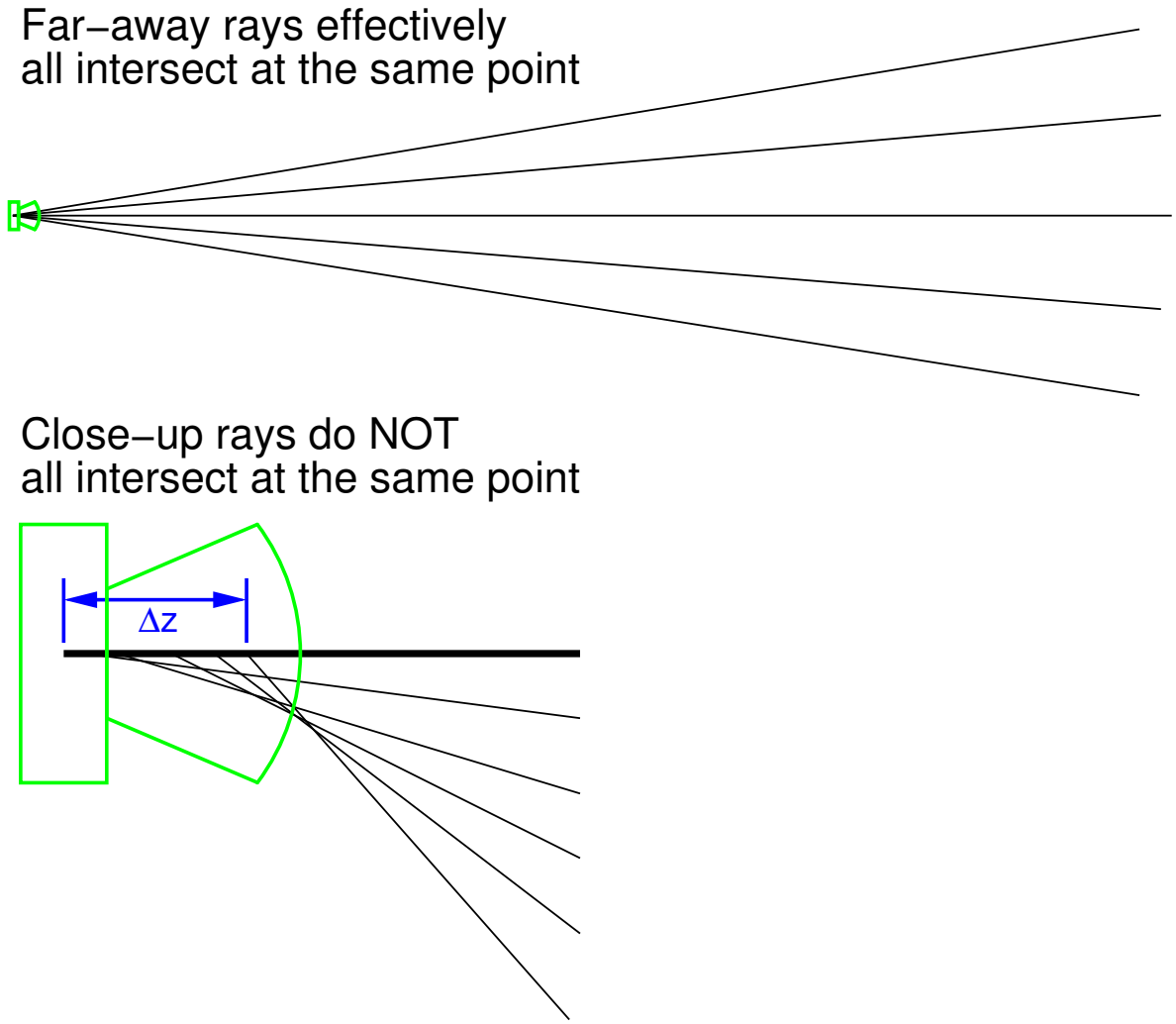
Core assumption of almost all camera modeling and processing:

- All rays of light intersect at a single point

This is not a valid assumption near the lens

Noncentrality

The size of the glass in the lens becomes non-negligible as we observe nearby objects



Noncentrality

- Most triangulation and stereo routines assume a central projection. This is true for non-closeups
- If necessary, noncentral behavior *can* be modeled:
  - mrcal has partial support, which was critically important for some projects
  - CAHVORE is noncentral with most people throwing away the noncentrality when they use it
- We should try to calibrate and use the cameras beyond where noncentral effects are significant. mrcal cross-validation will tell you if you're too close.

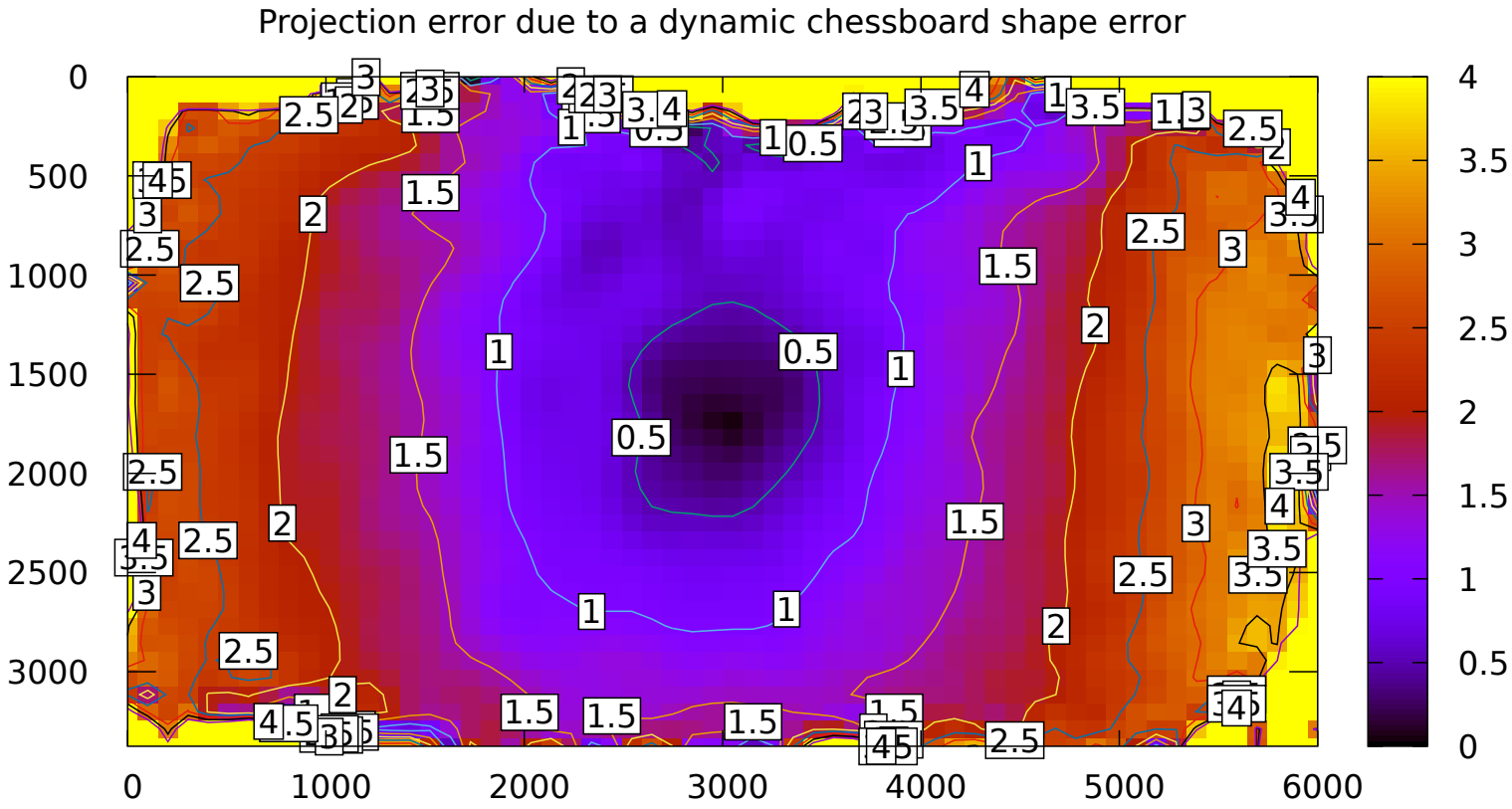
The downsides of huge chessboards

- Difficult to manufacture
- Expensive
- Unstable

mrcal has a simple *static* deformation model: a parabolic deformation in x and in y. Usually this isn't enough to accurately represent foam boards

The downsides of huge chessboards

Because intrinsics are sensitive to chessboard shape errors.
Simulated intrinsics calibration error due to a board shape error of 1mm in the center in one direction, and 0.5mm in the center in the other direction. No other noise present.



The downsides of huge chessboards

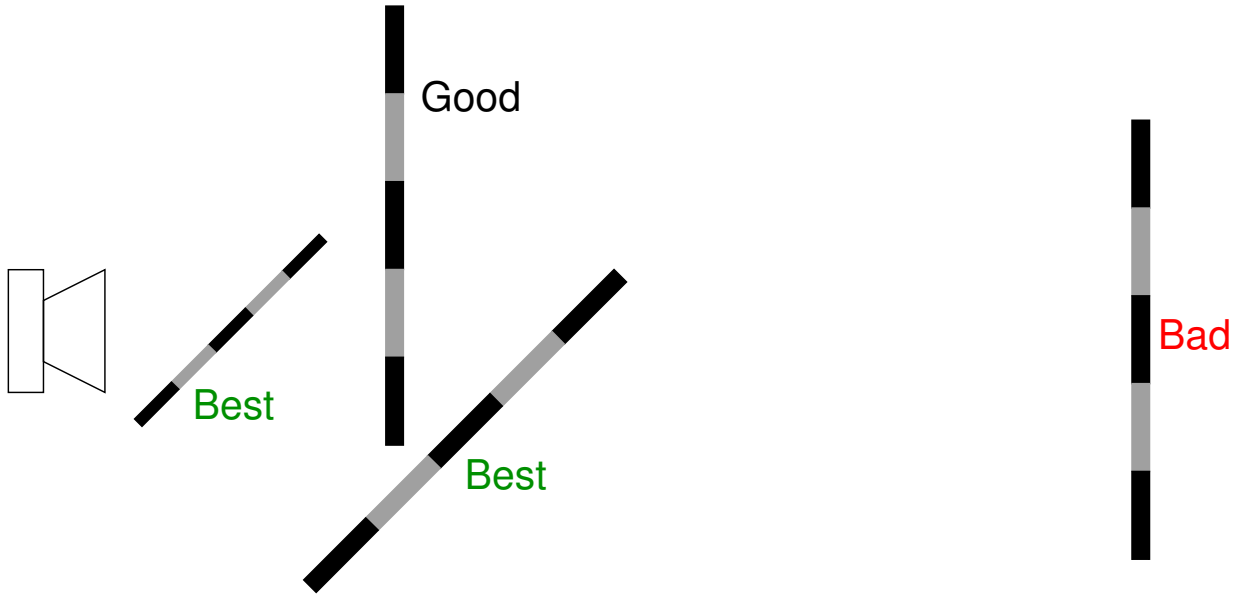
I usually use an Aluminum-honeycomb-backed 1m x 1m square board. This works well.

What kind of calibration object do we want? Conclusions

- Chessboard as large as possible
- Placed as close to the camera as possible
- With as dense a chessboard grid as possible

Using the mrcal tools to verify that we didn't go too far

How should we dance? Conclusions



Use mrcal tools to validate

Which model should we use for the lenses?

Today mrcal supports

- OpenCV models with 4,5,8,12 parameters
- CAHVOR, CAHVORE
- `LENSMODEL_SPLINED_STEREOGRAPHIC`: the rich, splined model

Unless you really need compatibility with a legacy system or you have low accuracy requirements,
`LENSMODEL_SPLINED_STEREOGRAPHIC` is strongly recommended.

Interpreting the calibration results

Once we have a calibration, we should see how well we did:

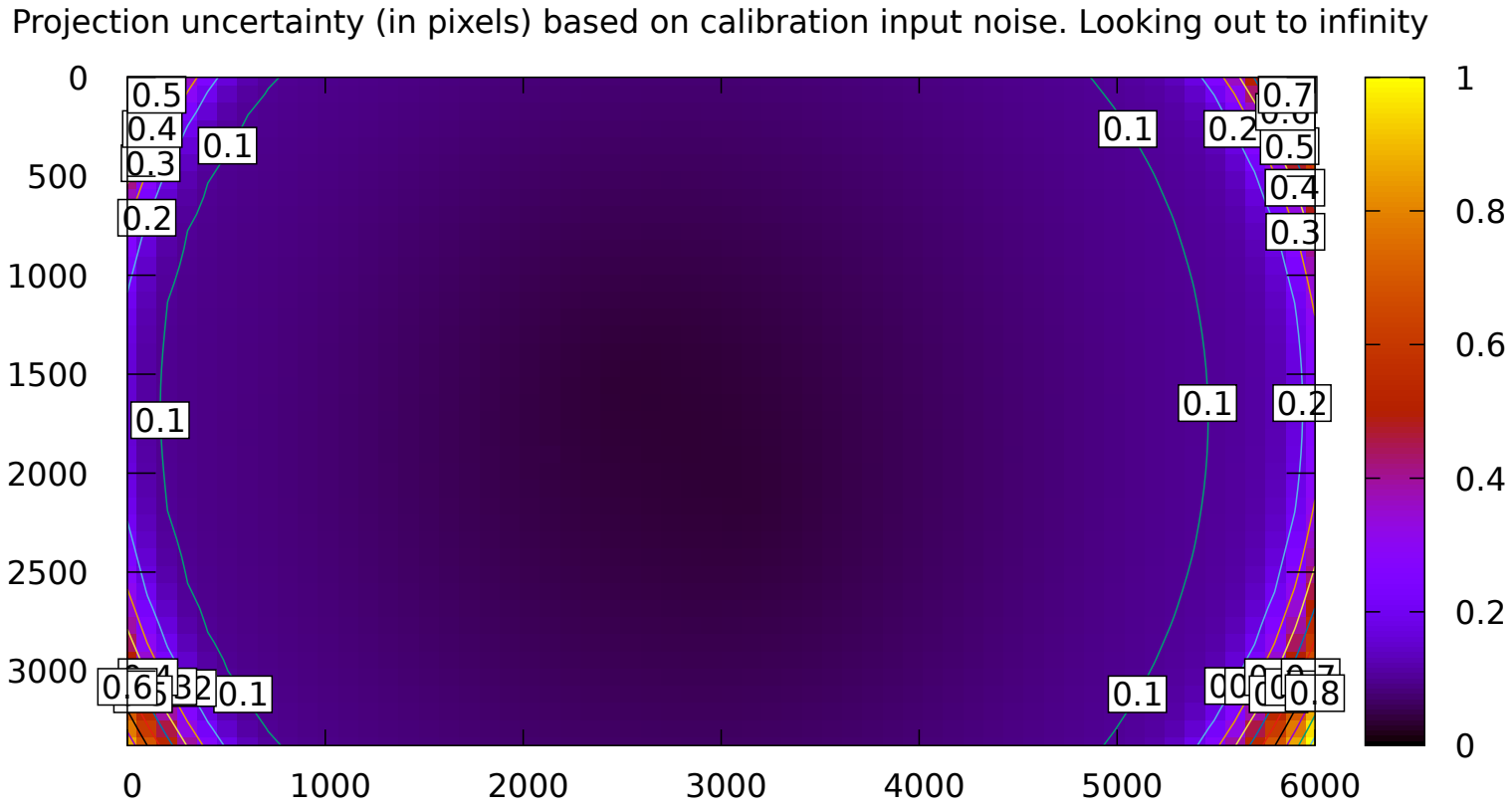
- We examine the projection uncertainty to make sure we have enough good data in the right places
- We examine the cross-validation diffs to confirm that the model fits
- If these diffs are too high, we examine the residuals to find the cause of our model errors

Projection uncertainty

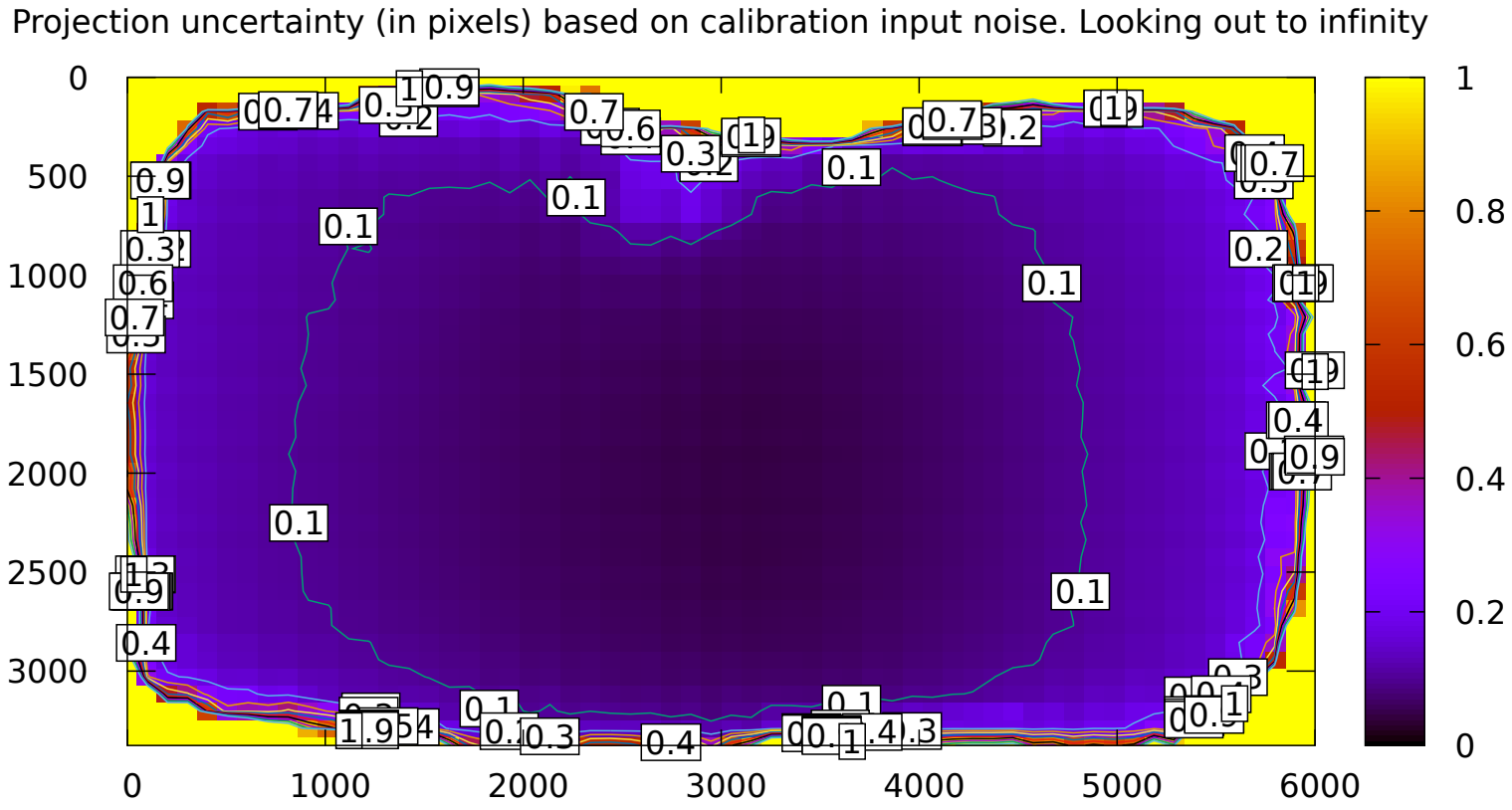
- Projection uncertainty gauges the effect of sampling error
- This is directly affected by the quality of the data we gathered. Problems with the chessboard dance will show up here
- Lean lens models (anothing other than `LENSMODEL_SPLINED_STEREOGRAPHIC`) will produce an overly-optimistic uncertainty report
- A low projection uncertainty is a necessary, but not sufficient condition for a good calibration: uncertainty reporting samples the input pixel noise, but not the model noise

If the uncertainty is unacceptable, stop there, and fix that first.

DTLA projection uncertainty: OPENCV8



DTLA projection uncertainty: splined model



Cross-validation diffs

Now we look for model errors

- We split our input dataset, and process the subsets independently: this samples the model error
- We use the differencing method to compare the projection behaviors
- Unlike the uncertainty reporting, interpreting these requires some thought

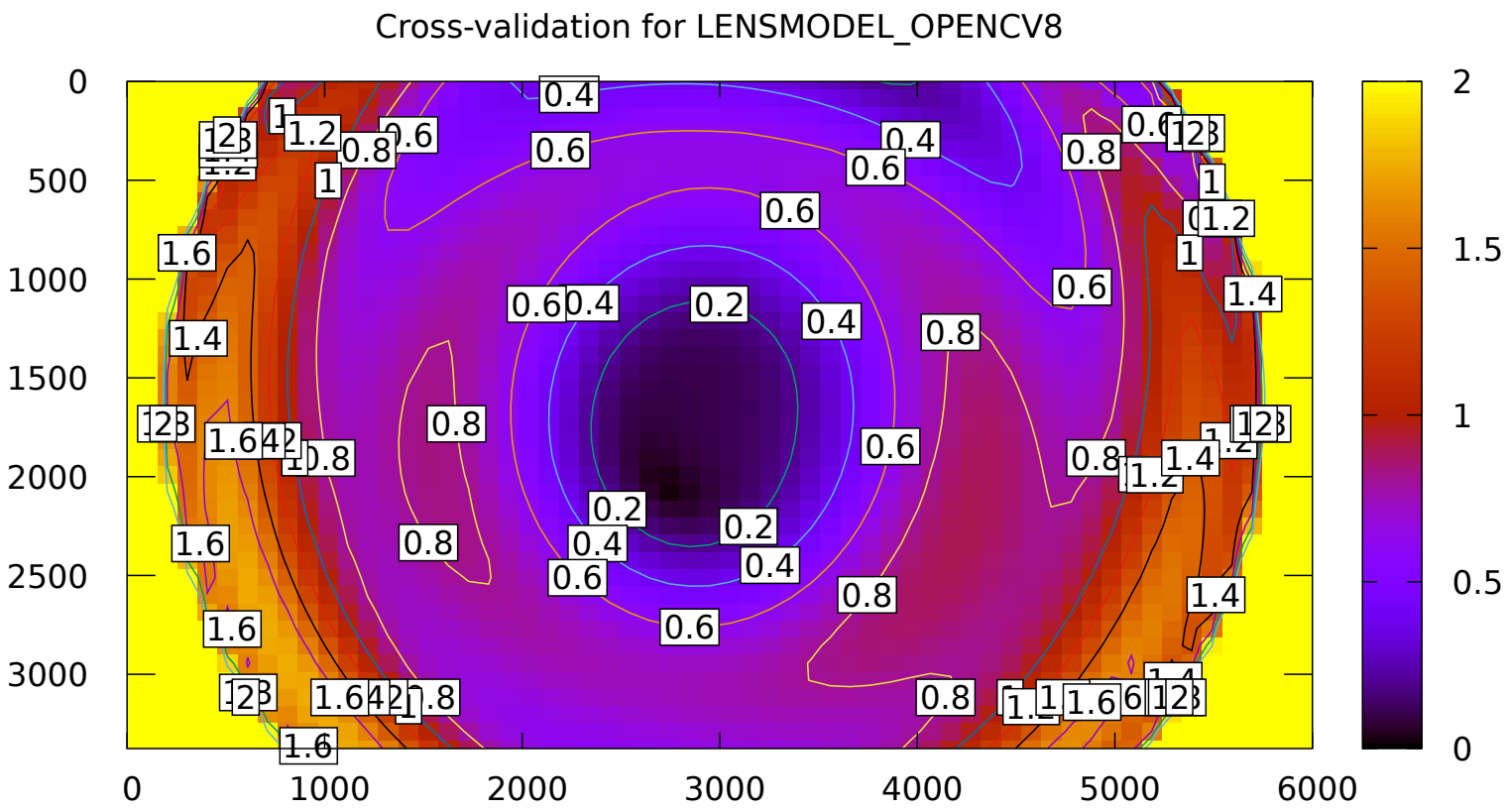
Cross-validation diffs: detecting model errors

I want to see

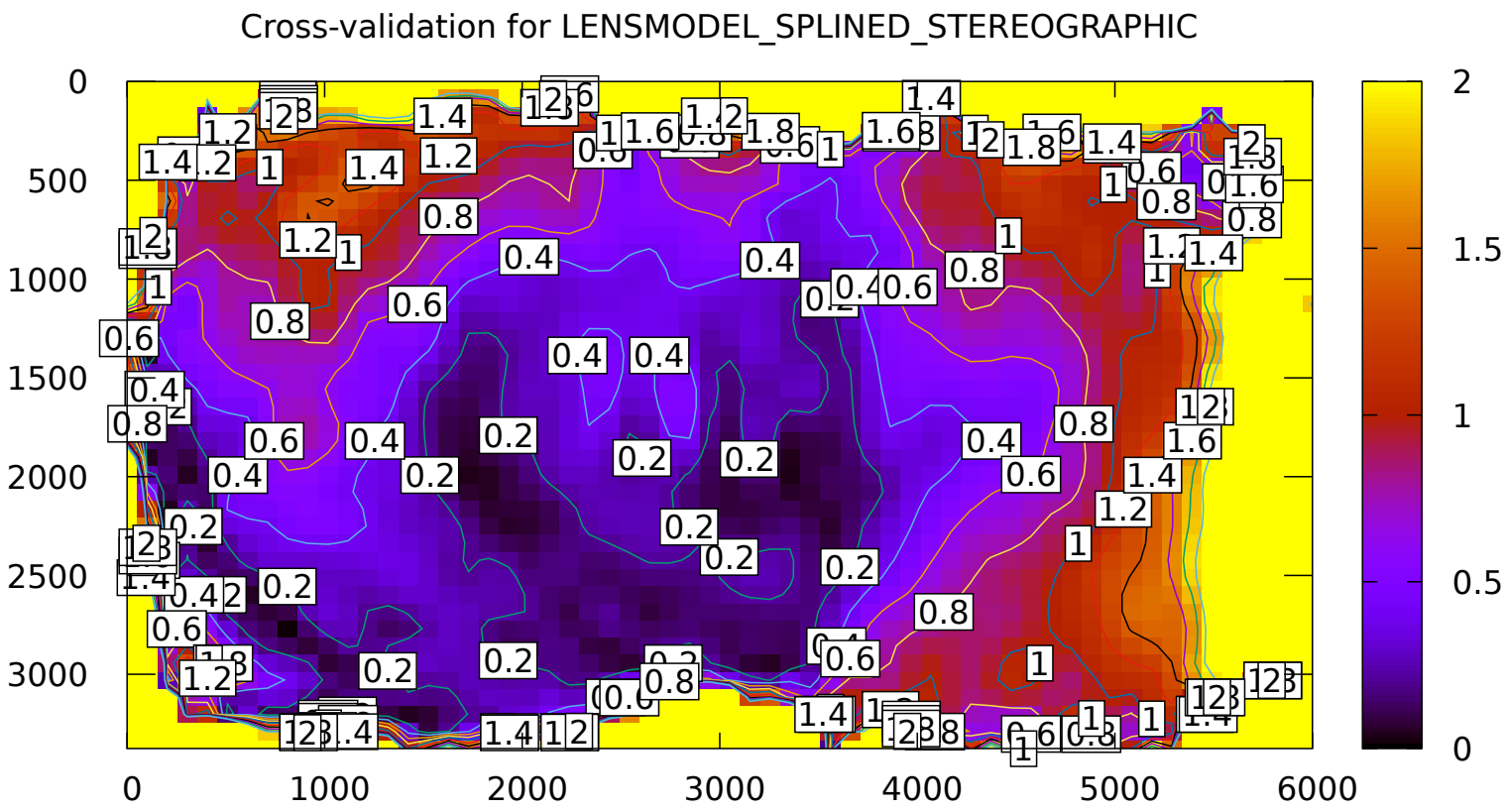
$$E_{\text{uncertainty}_0} + E_{\text{uncertainty}_1} \approx E_{\text{crossvalidation}}$$

Let's look at the downtown LA data. We want to see a cross-validation diff of ~ 0.2 pixels.

DTLA cross-validation diffs: OPENCV8



DTLA cross-validation diffs: splined model

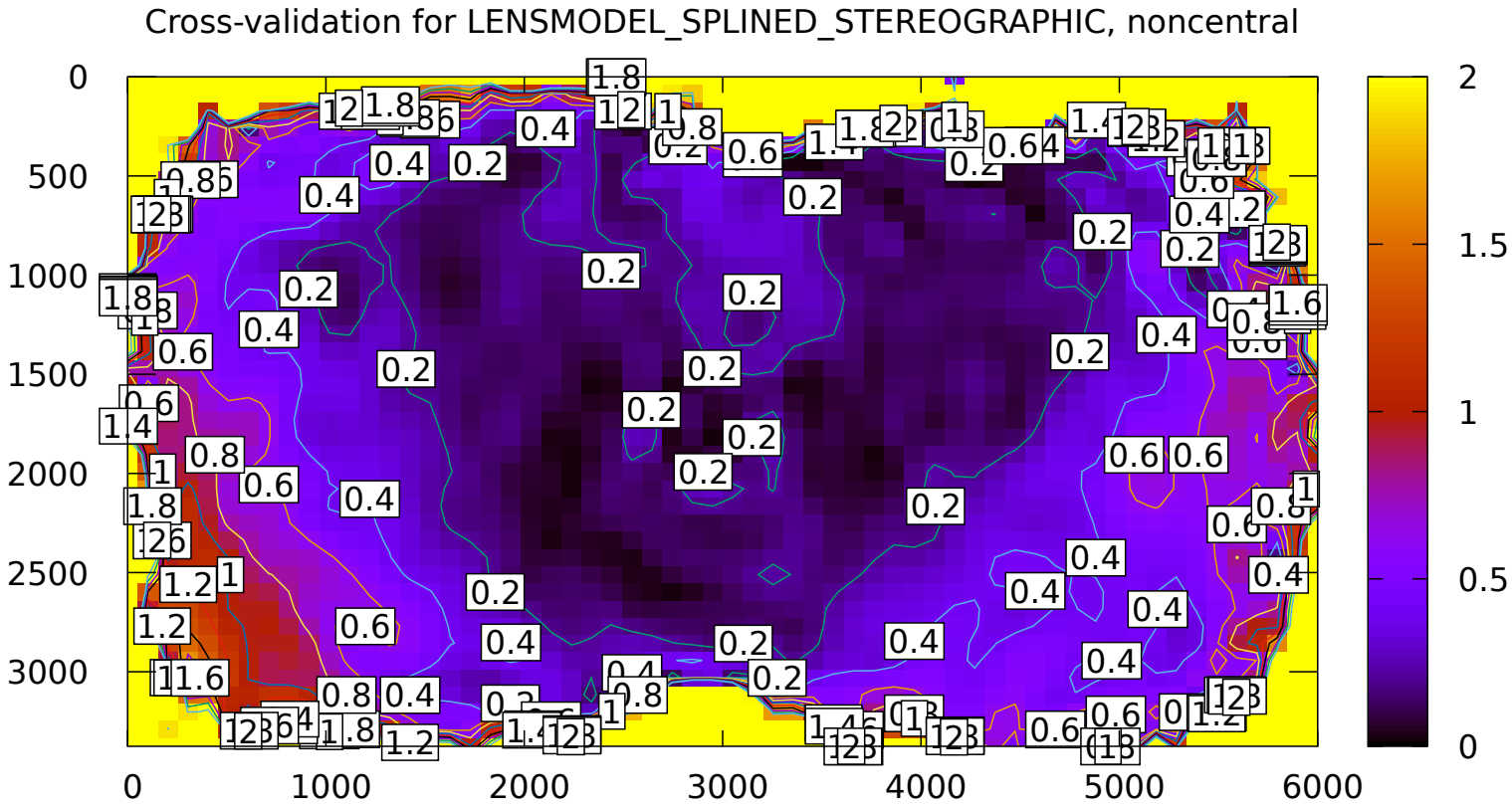


DTLA cross-validation diffs

- Clearly the `LENSMODEL_OPENCV8` result has issues
- But the `LENSMODEL_SPLINED_STEREOGRAPHIC` result has too-high errors too

Because I captured images from too close to the lens, and we're seeing non-negligible noncentral behavior. Asking mrcal to model that behavior produces:

DTLA cross-validation diffs: splined model, noncentral



DTLA cross-validation diffs

- If this calibration was important, I would get a different dataset from further out

DTLA cross-validation diffs

- Here the cross-validation diffs alerted us to the presense of a problem. They are *very* good at that
- Finding the cause of the problem requires some intuition and experimentation

Residuals

- One technique is available to help diagnose problems: examining the solve residuals

Residuals

We usually have a *lot* of images and a *lot* of residuals. I look at the few worst-fitting images. Usually I only look at the residuals if

- I'm calibrating an unfamiliar system
- I don't trust something about the way the data was collected
- Something unknown is causing issues (we're seeing too-high cross-validation diffs), and we need to debug

Model errors are indicated with noise that is correlated or heteroscedastic, so we look for patterns in the residuals.

Let's examine the residuals we get from common problems

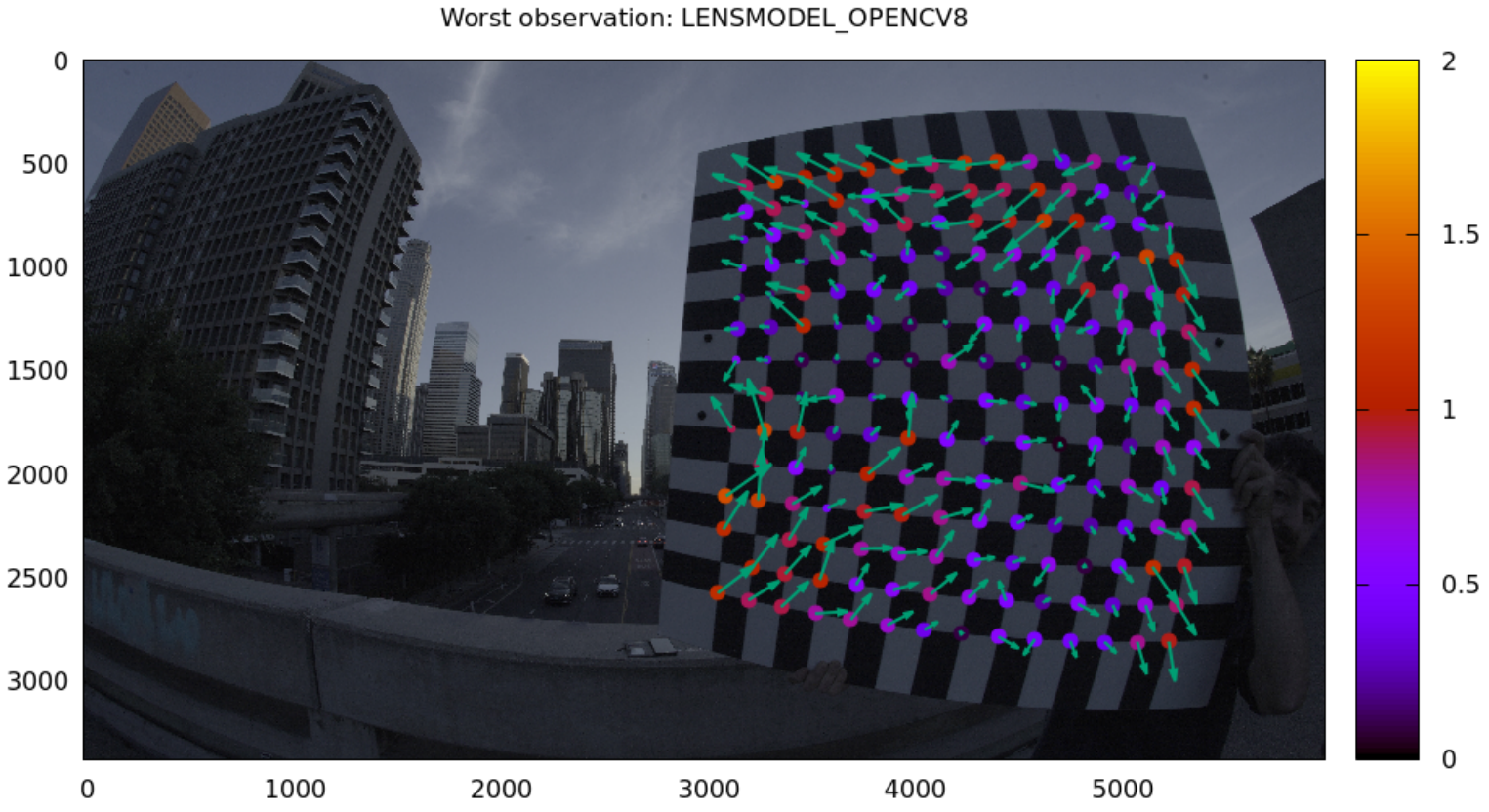
Residuals: poorly-fitting lens model

We saw this in the downtown Los Angeles data

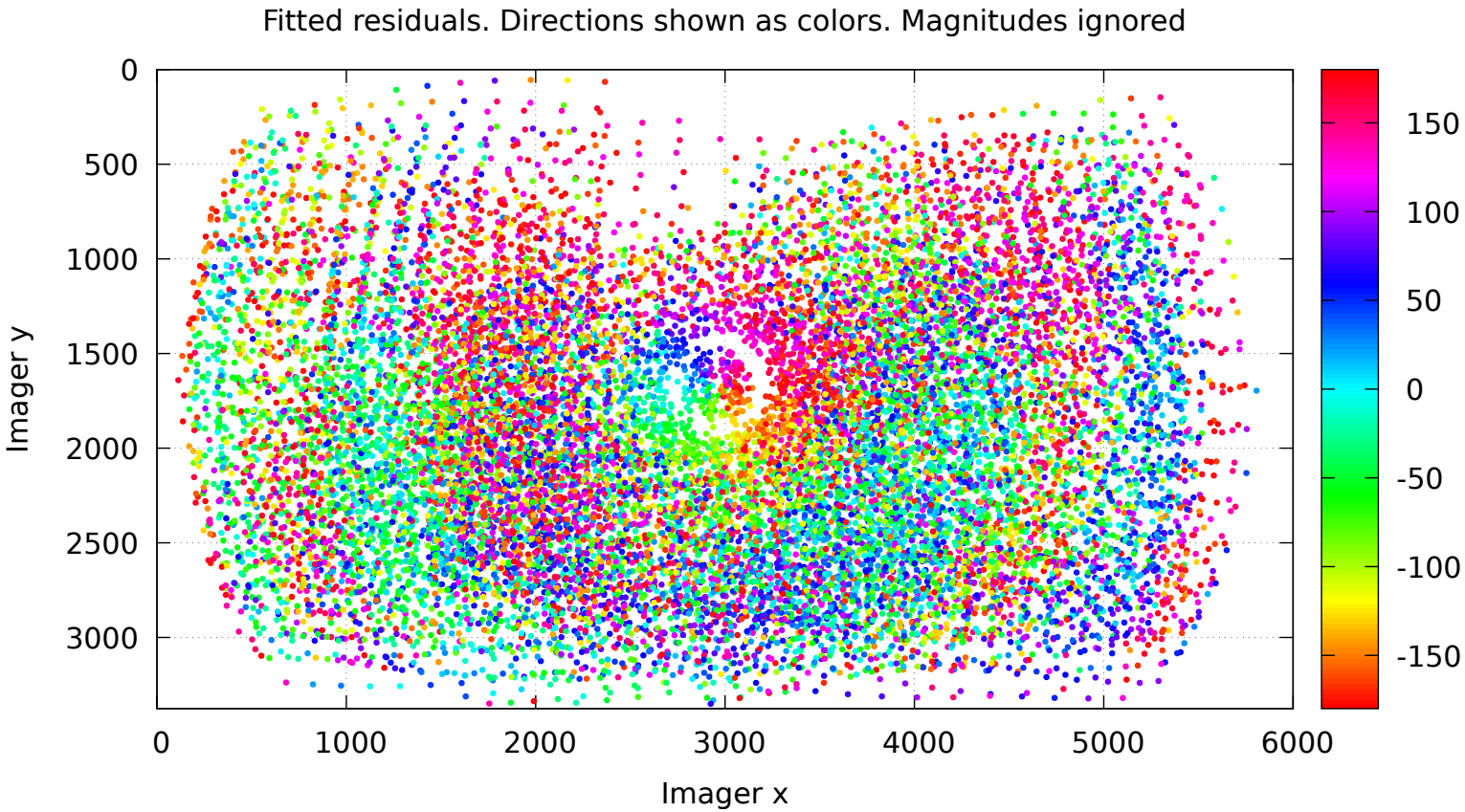
- We looked at both the `LENSMODEL_OPENCV8` and `LENSMODEL_SPLINED_STEREOGRAPHIC` residuals
- The latter was much better, but still showed patterns

Earlier residual plots follow below

Residuals: LENSMODEL_OPENCV8: the worst image



Residuals: LENSMODEL_OPENCV8: residual directions

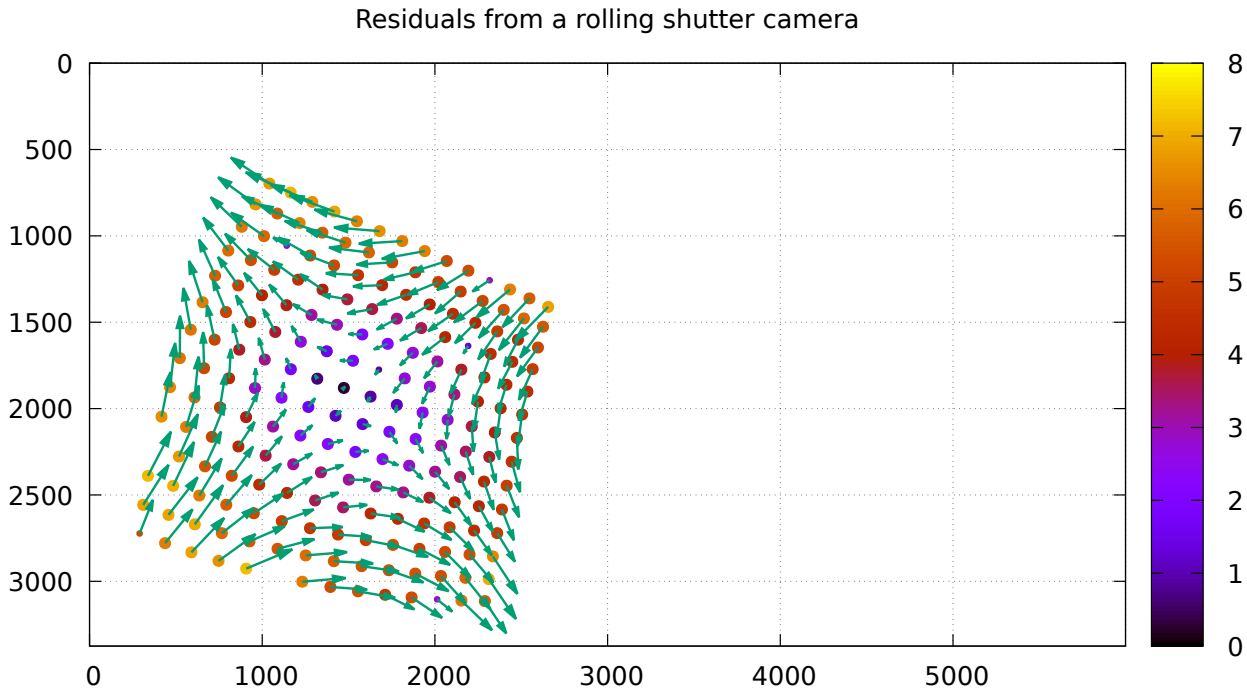


Residuals: rolling shutter

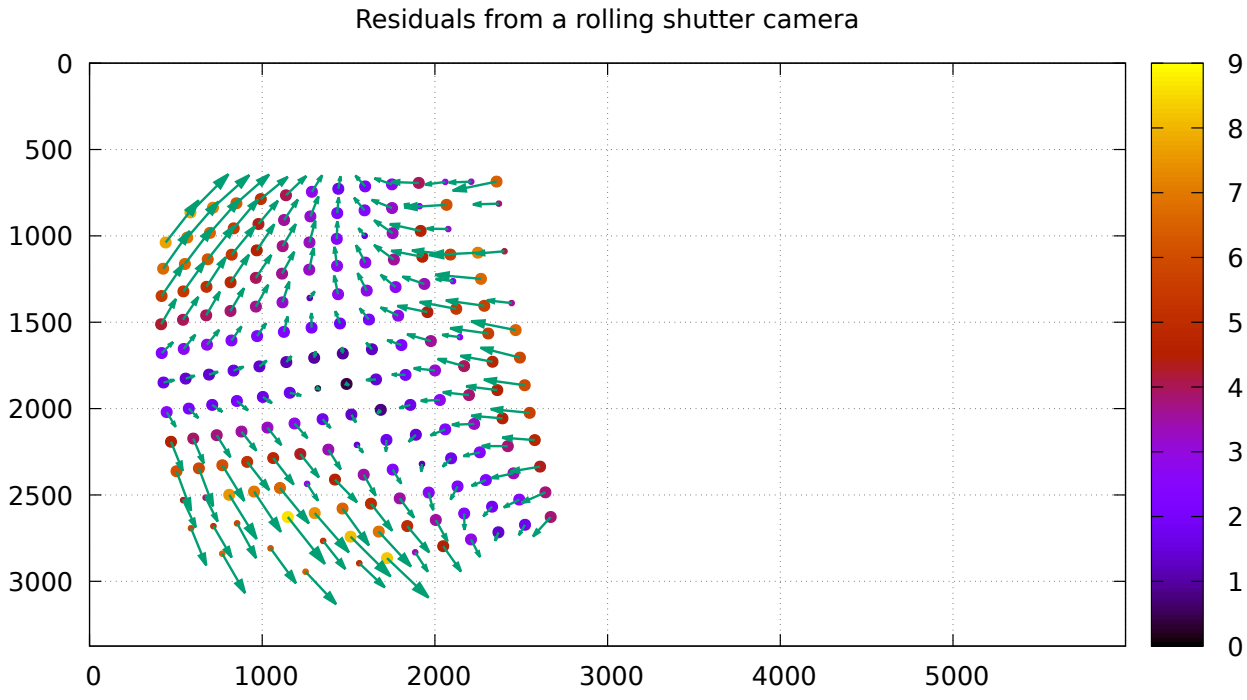
Some cameras save money on memory by sending pixel data as it is captured. The result: rolling shutter cameras capture different parts of the image at different times.

This produces funky residuals

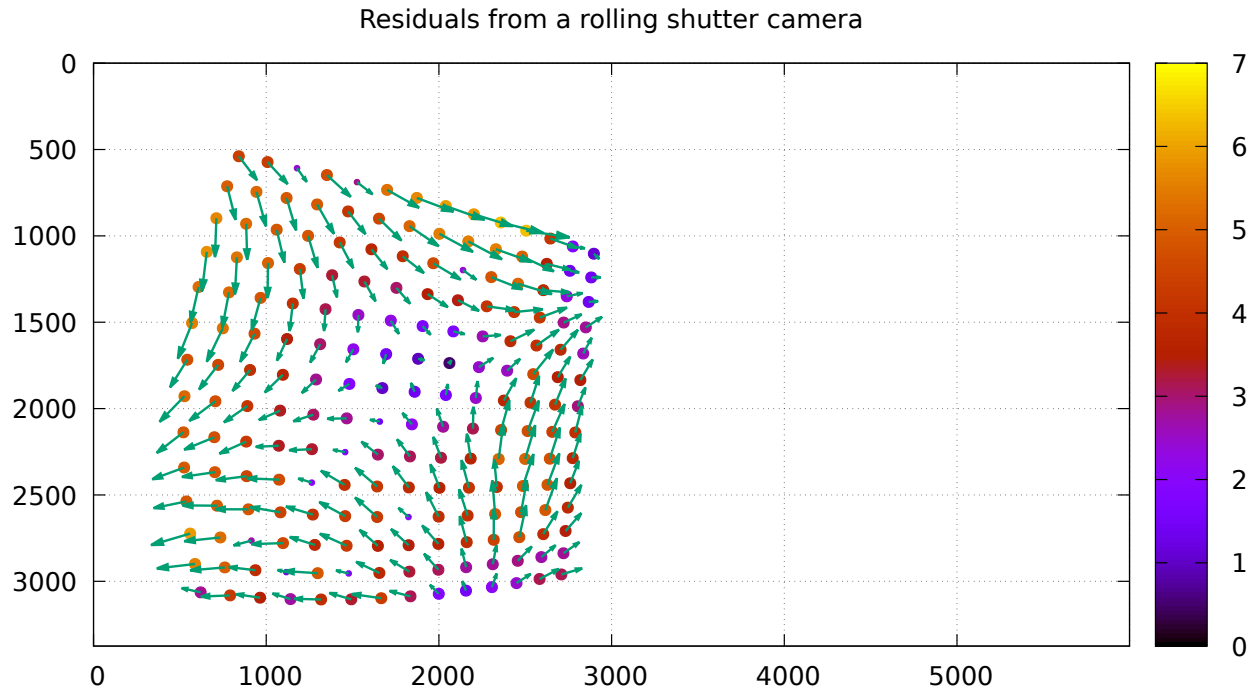
Residuals: rolling shutter



Residuals: rolling shutter



Residuals: rolling shutter

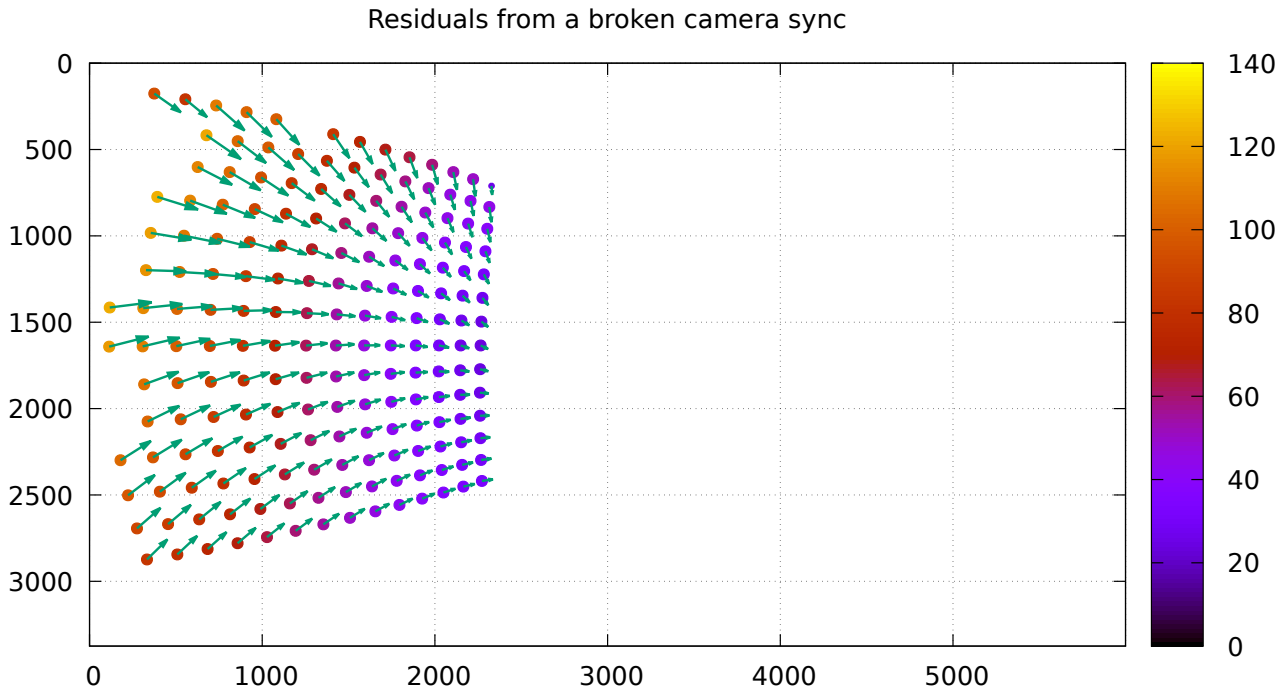


Residuals: syncronization errors

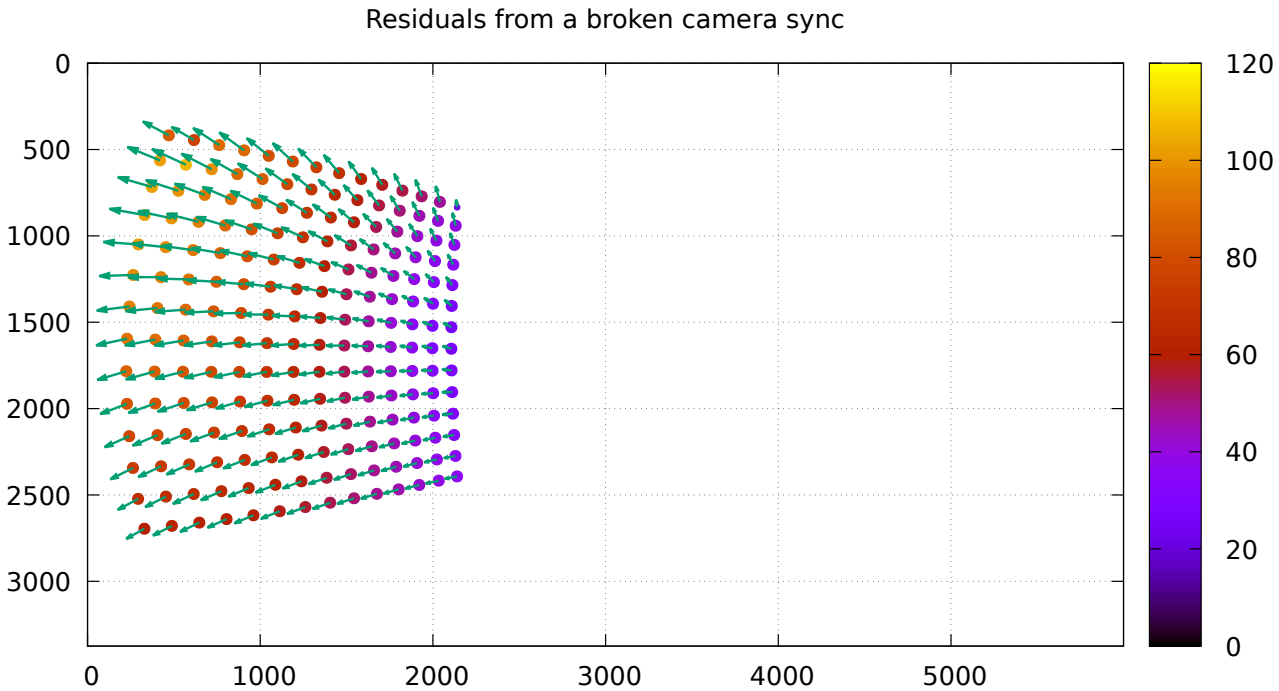
- In a multi-camera calibration we assume that sets of images were captured at the same instant in time
- This requires a shared physical wire that each camera uses to initiate image capture

If this doesn't work right we get the tell-tale residuals, and we can examine the solution to find the smoking-gun images that prove the breakage

Residuals: syncronization errors



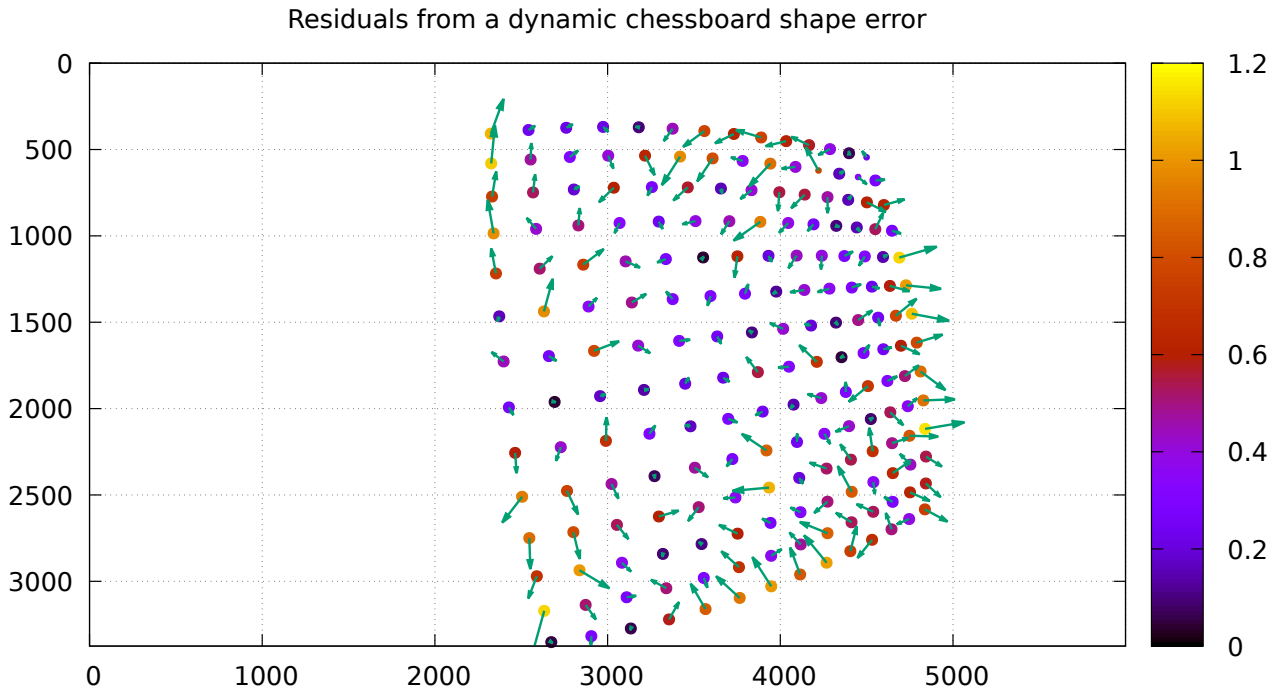
Residuals: syncronization errors



Residuals: chessboard shape errors

- Errors in chessboard shape are difficult to disentangle from errors in intrinsics
- We can have static and dynamic shape errors:
  1. The chessboard is non-flat, but in a way not modeled by the solver
  2. The chessboard shape changes over the course of the chessboard dance

Residuals: errors due to unstable chessboard shape



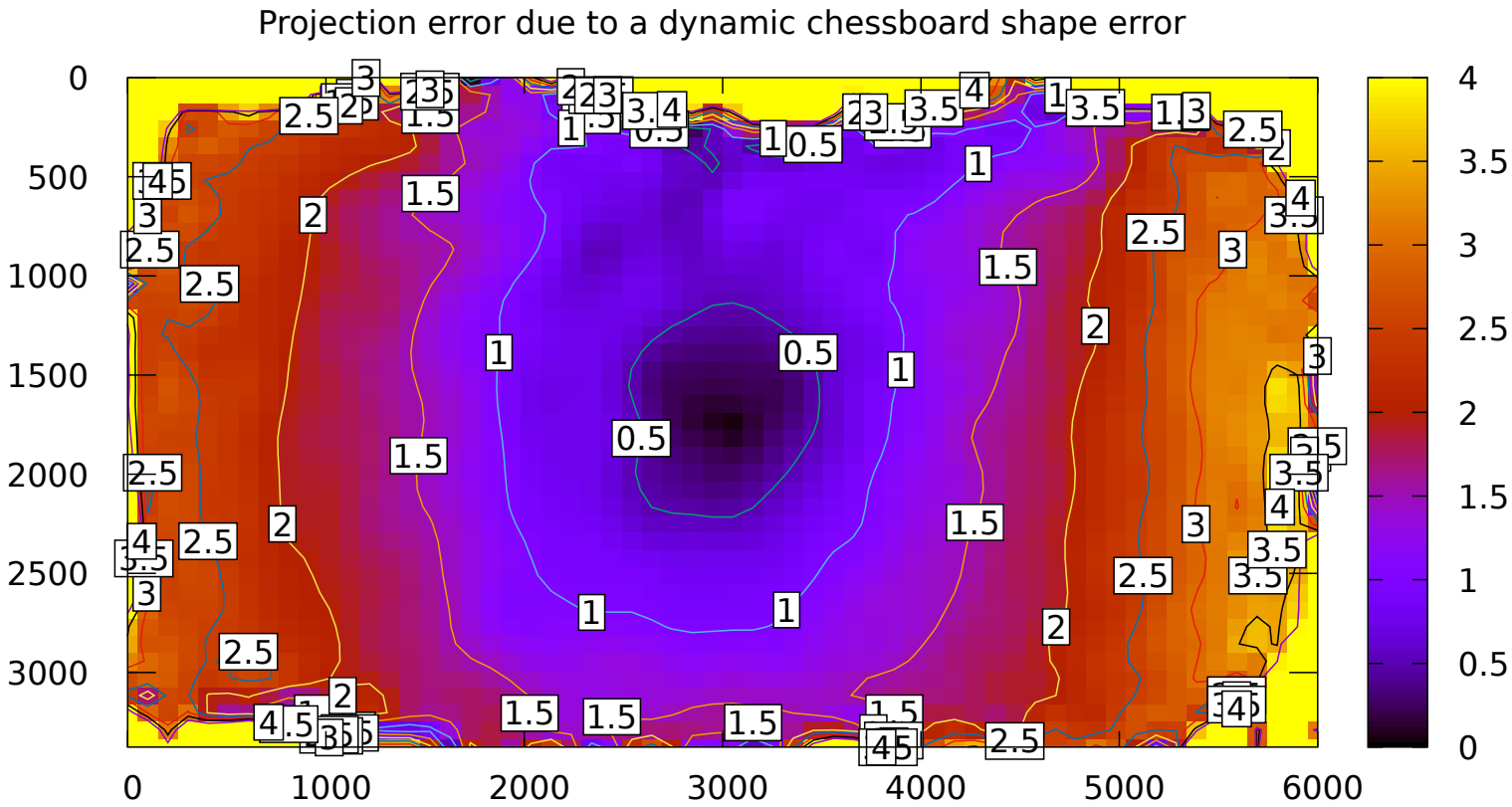
Residuals: chessboard shape errors. Conclusions

- These are hard to conclusively pick out from residual plots
- It's helpful to look at more than just 1 or 2 worst-case images
- The most tilted chessboard observations usually show very consistent residual vectors along the far edge of the chessboard

Perfectly-corrupted solves

mrcal can report the errors from a solve containing *only one kind* of hypothetical error. This measures the effect of problems we think may exist

A board shape error of 1mm in the center in one direction, and 0.5mm in the center in the other direction does this:



Camera stability

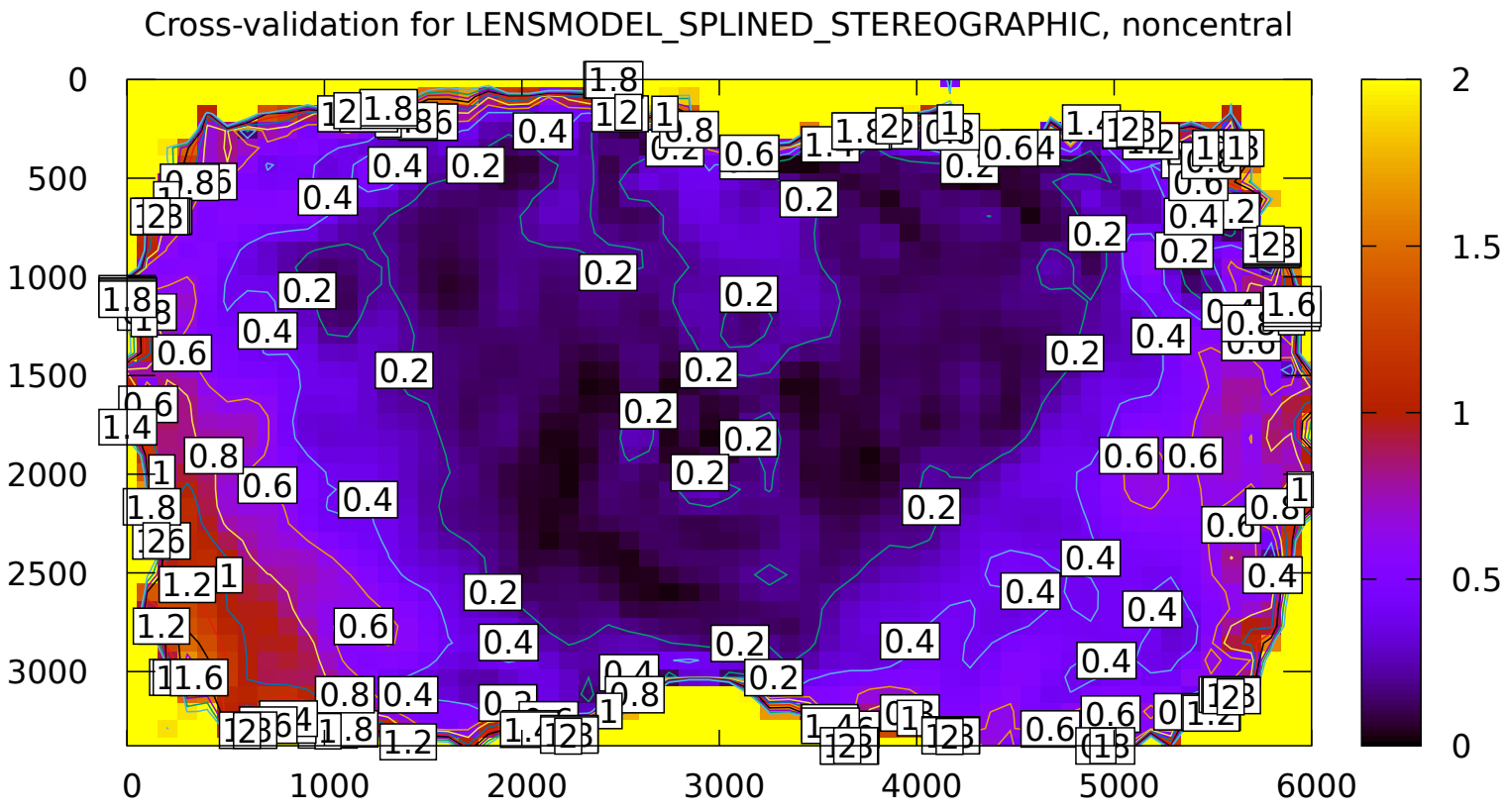
Let's switch gears, and look at some applications

Let's use the differencing method to gauge stability of intrinsics:

- If we stress a camera system (mechanically, thermally, etc), does its behavior change?

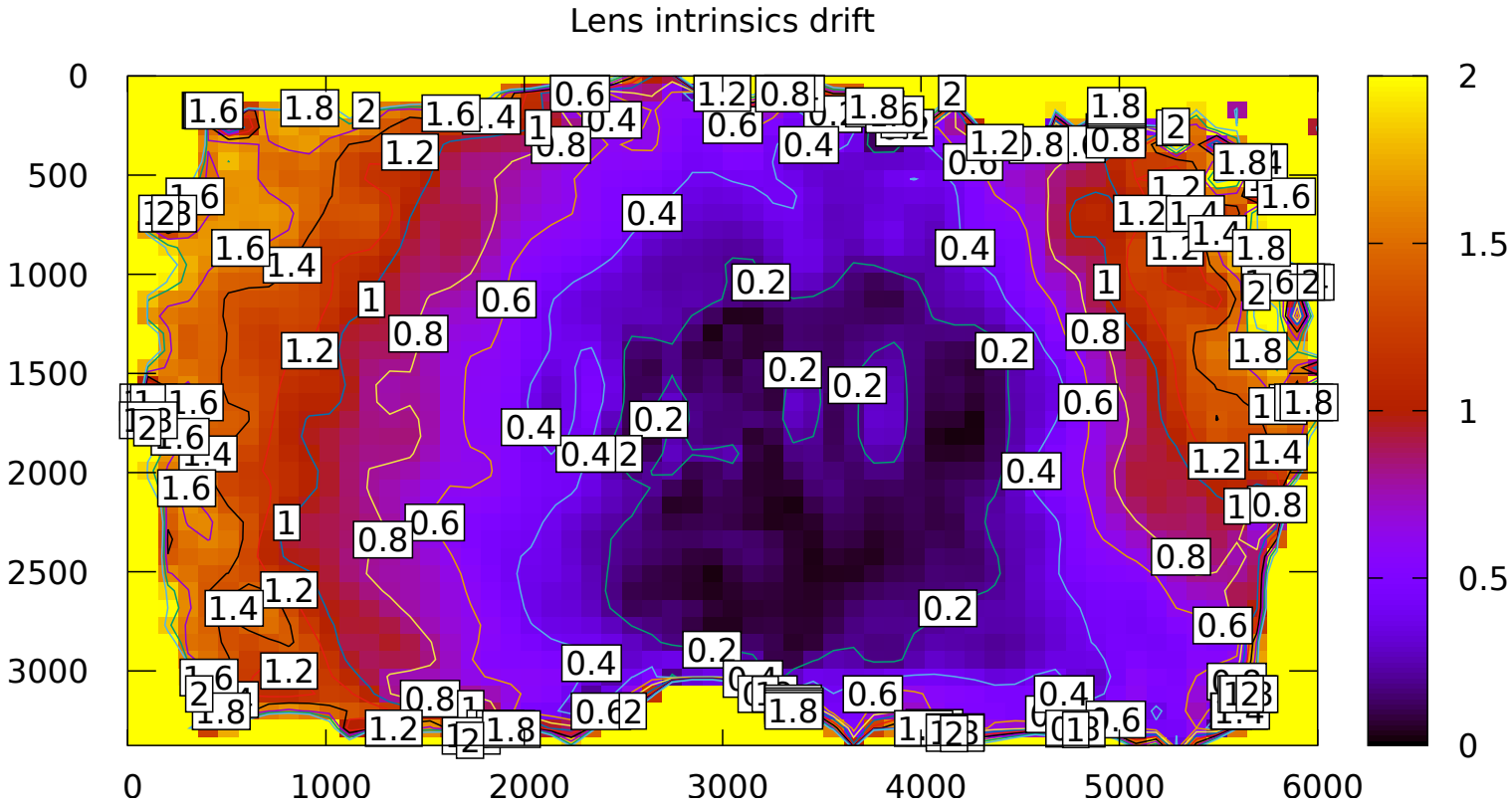
Lens stability

As a baseline, once again here's the cross-validation diff from the downtown Los Angeles dataset. This is the *difference between two subsequent solves without touching anything*



Lens stability

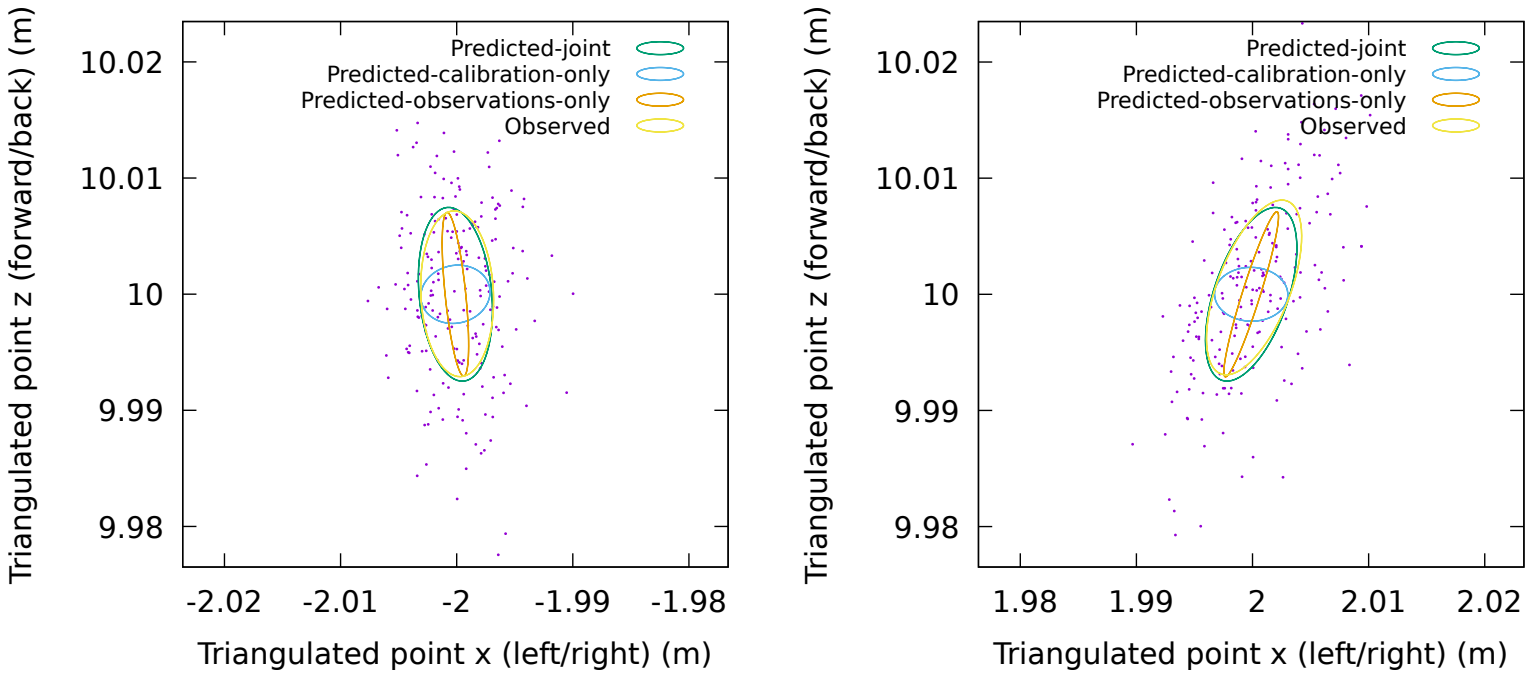
- Then I moved the camera and tripod over by 2m or so, and gathered more chessboard images. Comparison from before:



Estimating ranging errors caused by calibration errors

- Projection errors aren't what we ultimately care about
- mrcal allows us to propagate these to what we care about

Propagating errors to triangulation:



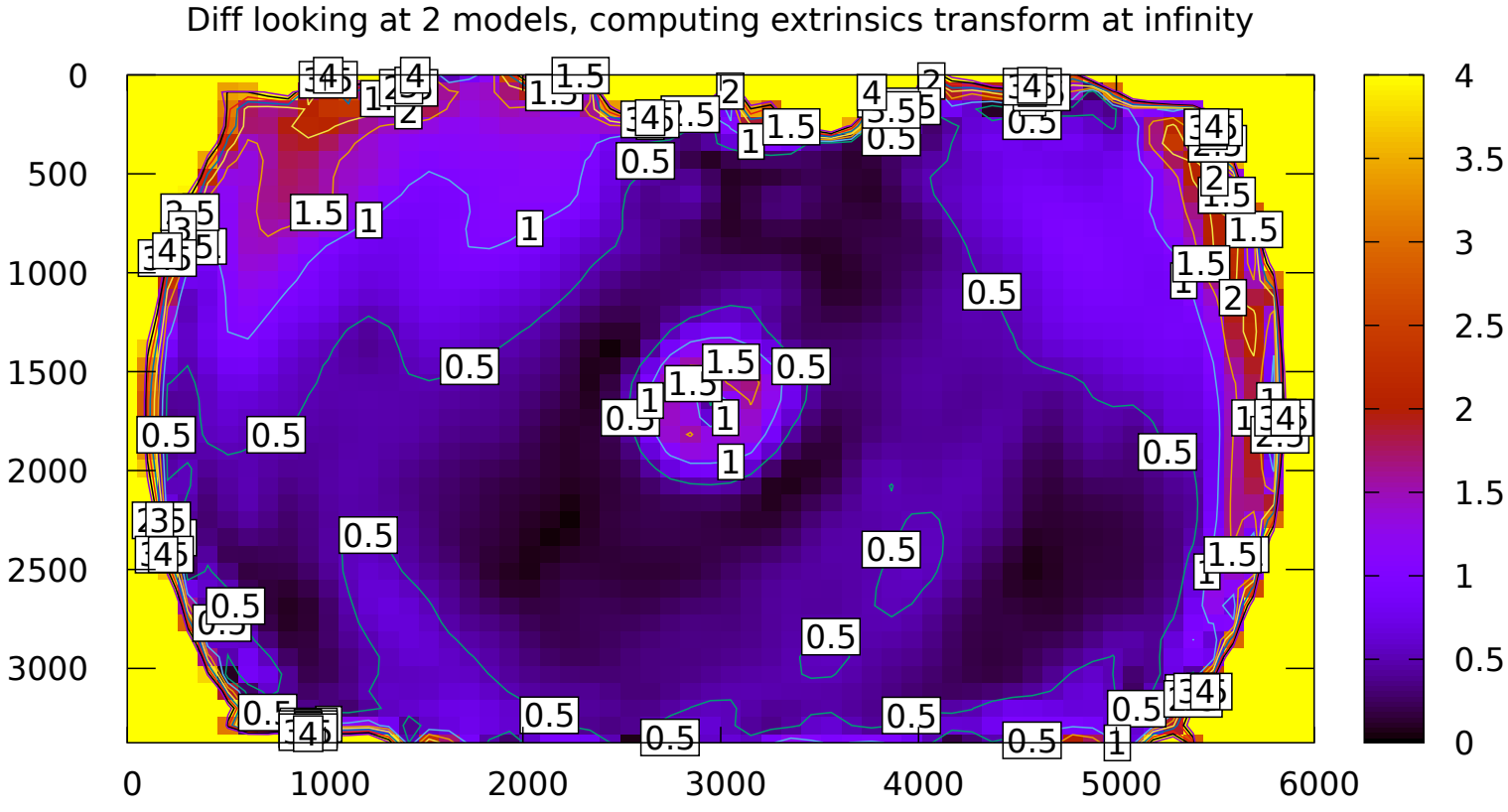
Scene-aware error propagation

If the rough geometry of an observed scene is known beforehand, we can make a rough expected-error map:

1. Compute $\frac{\partial \text{range}}{\partial \text{azimuth}}$ from the triangulation expression
2. Estimate $\Delta\text{azimuth}$ by combining expected sampling error and calibration error
3. $\Delta\text{range} \approx \frac{\partial \text{range}}{\partial \text{azimuth}} \Delta\text{azimuth}$

Scene-aware error propagation

In the downtown Los Angeles scene we observed this calibration error using `LENSMODEL_OPENCV8`:



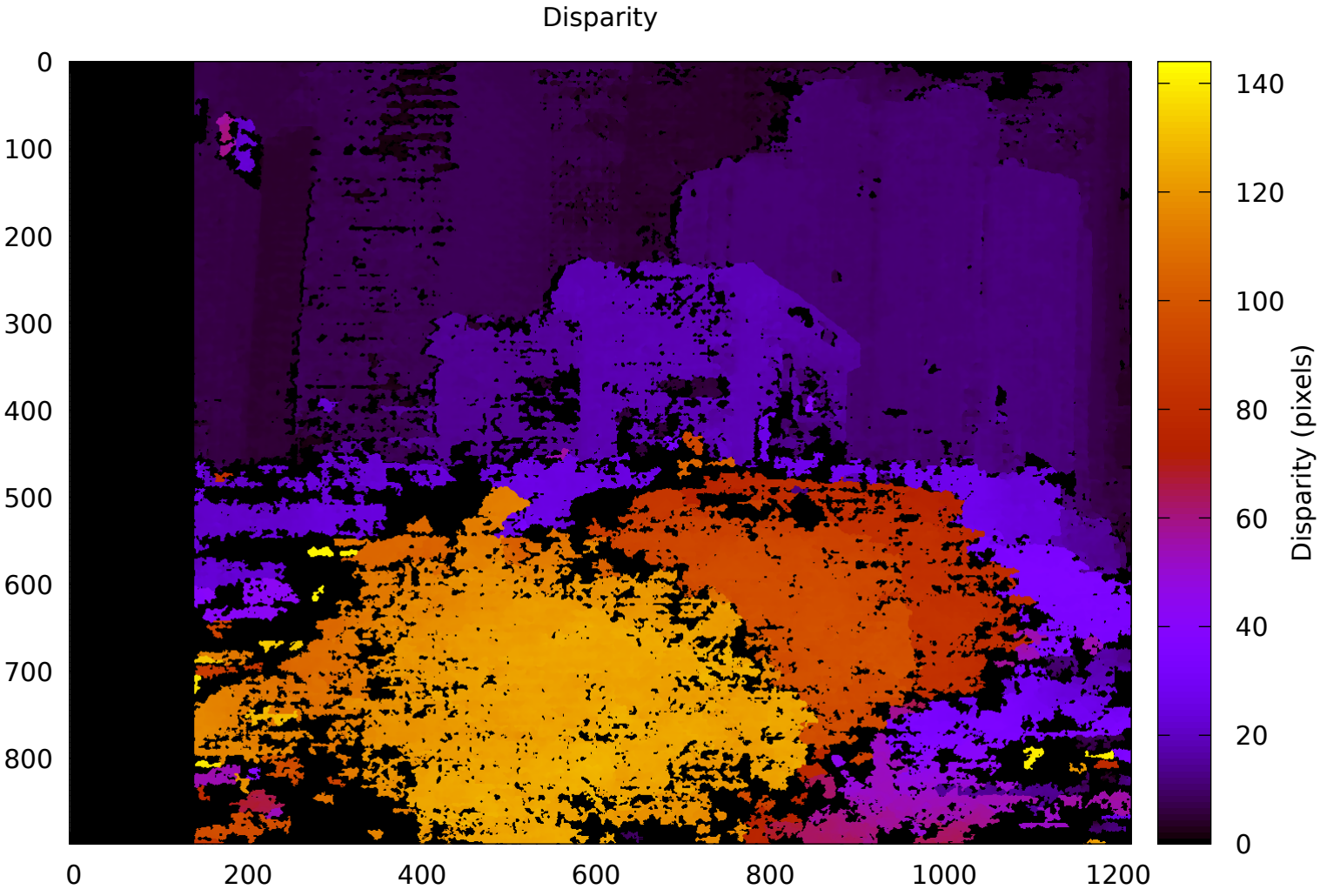
Scene-aware error propagation

- Rough median of calibration error: 0.5 pixels per camera
- At worst: 1.0 pixels of calibration error
- Noise in stereo matching is ~ 0.3 pixels

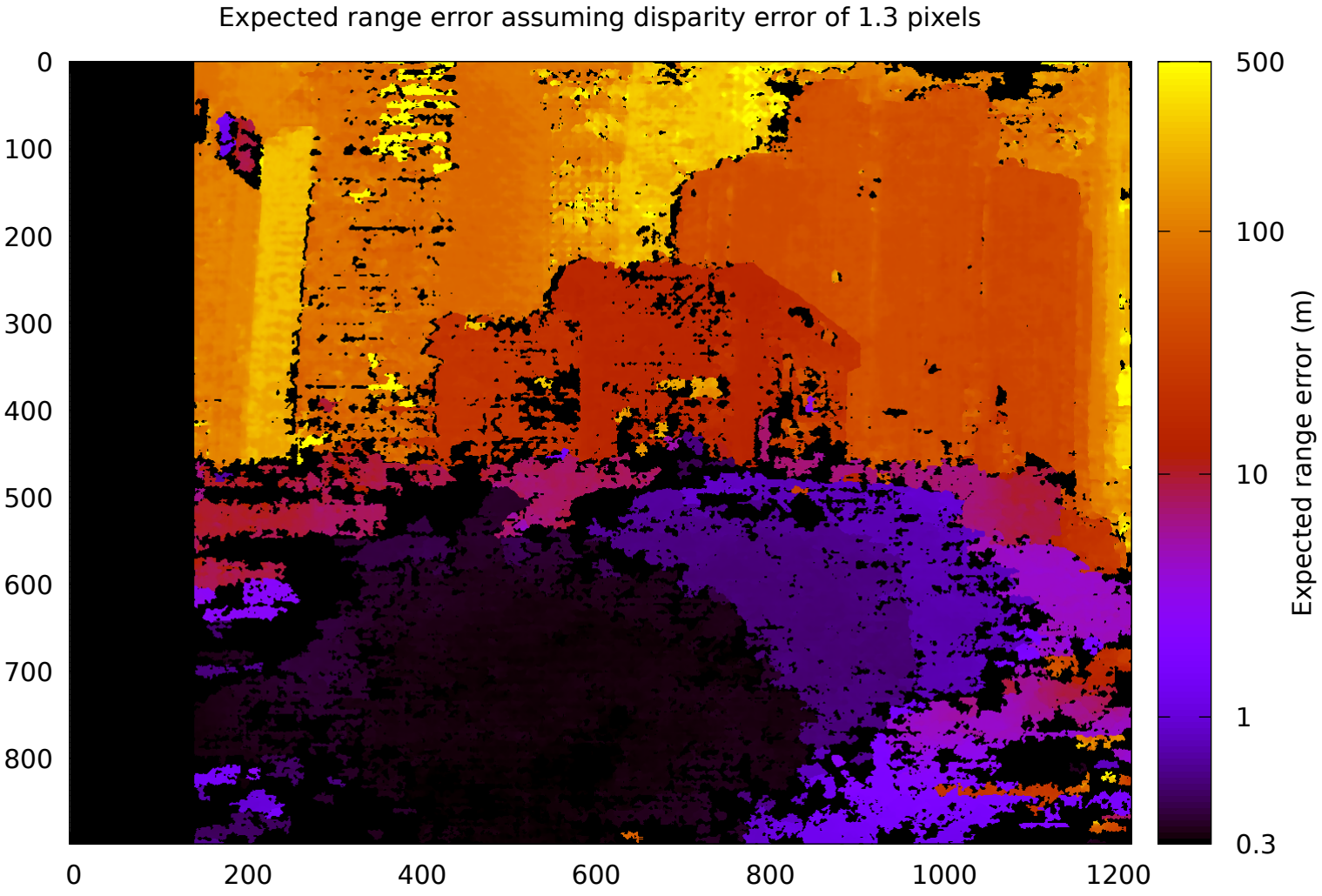
So we assume a disparity error of $1.0 + 0.3 = 1.3$ pixels

Scene-aware error propagation: left-rectified image

Scene-aware error propagation: disparity



Scene-aware error propagation: propagated range error



Scene-aware error propagation

These errors are correlated and will *not* average out. They should be minimized.

Conclusion

- mrcal solves many pervasive issues in traditional camera-modeling toolkits
- Allows many practical questions to be addressed directly
- Many improvements and extensions and applications planned and in development