Homework Assignment 6

15-463/663/862, Computational Photography, Fall 2021
Carnegie Mellon University

Due Sunday, Dec. 5, at 11:59pm

The purpose of this assignment is to use structured light as a means for reconstructing the 3D shape of a surface. Rather than having to use a projector, you will rely on shadows to create a structured illumination, a technique often referred to as "weakly structured light".

In particular, you will implement the "desktop scanner" of Bouguet and Perona [1]. As shown in Figure 1, this system is composed of five primary items: a camera, a point-like light source (e.g., desk lamp), a stick, two planar surfaces, and a calibration checkerboard. By waving the stick in front of the light source, you can cast line shadows into the scene. As Bouguet and Perona demonstrate, we can recover the depth at each pixel using simple geometric reasoning.

In the course of completing this homework, you will need to develop a good understanding of camera calibration, Euclidean coordinate transformations, and manipulation of lines and planes. Rather than just "encouraged" to read it, to solve this assignment you will need to carefully go through Bouguet and Perona [1]. So you should read it carefully before starting the assignment. Note that, for simplicity, the calibration and reconstruction procedures (Parts 1.2 and 1.3) we use in the assignment are significantly different from those the paper describes.

1 Implementing structured-light triangulation (100 points)

For the first part of the homework, you will be using two image sequences contained in the `./data` directory of the homework ZIP archive. One is a `calib` sequence you will use to estimate the intrinsic and extrinsic calibration parameters of the camera, and consists of ten images of a checkerboard at various poses. The second is a sequence captured for the `frog` object shown in Figure 1. For each sequence we have provided both a high-resolution 1024×768 sequence, and a low-resolution 512×384 sequence you can use during development. You should convert these color images to grayscale, e.g., using `rgb2gray`.

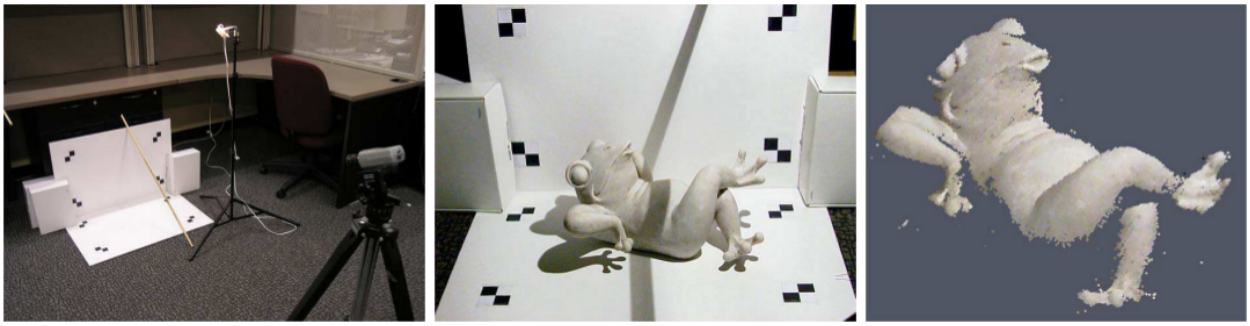


Figure 1: 3D Photography using Planar Shadows. From left to right: the capture setup, a single image from the scanning sequence, and a reconstructed object (rendered as a colored point cloud).

1.1 Video processing (25 points)

Your first task is to estimate two fundamental quantities from an input video sequence: (1) the position of the shadow edge within each frame t; and (2) the time t that a shadow enters a pixel. The following sections outline the basic procedures for performing these tasks. You will need to consult Section 2.4 in [1] for additional information.

Spatial shadow edge localization. In terms of Figure 2 in [1], you need to estimate the shadow lines $\lambda_h(t)$ and $\lambda_v(t)$, for the horizontal and vertical planar regions, respectively. You will later use these lines to recover the position and orientation of each shadow plane as a function of time.

In order to perform this and subsequent processing, you will utilize a spatio-temporal approach. We begin by defining the maximum and minimum intensity observed at each pixel (x, y),

$$I_{\max}(x, y) = \max_t I(x, y, t), \tag{1}$$

$$I_{\min}(x, y) = \min_t I(x, y, t). \tag{2}$$

In order to detect the shadow boundaries, we choose a per-pixel detection threshold which is the midpoint of the dynamic range observed in each pixel. Then, at each time t, the shadow edge can be localized using the spatial *zero-crossings* of each line of the difference image

$$\Delta I(x, y, t) = I(x, y, t) - I_{\text{shadow}}(x, y), \tag{3}$$

where the shadow threshold image is defined to be

$$I_{\text{shadow}}(x, y) = \frac{I_{\max}(x, y) + I_{\min}(x, y)}{2}. \tag{4}$$

To detect the horizontal and vertical lines at each frame, for each planar region, you will need to work with the portion of the frame where the plane is unobstructed by the object you are scanning.

For each row of these regions, by interpolating the positions of the zero-crossings of $\Delta I(x, y, t)$, you can compute sub-pixel shadow edge locations. To produce the final estimate of the shadow edges $\lambda_h(t)$ and $\lambda_v(t)$, you should find the best-fit line (in the least-squares sense) to the set of shadow edge samples. Figure 2(a) shows the desired output of this step for one of the frames, where the best-fit lines are overlaid on the original image.

Temporal shadow edge localization. As described in Section 1.3, in order to reconstruct the object, you also need to know when each pixel entered the shadowed region. This task can be accomplished in a way similar to spatial localization. Instead of estimating zero-crossing along each row for a fixed frame, you can assign the per-pixel shadow time using the zero crossings of the difference image $\Delta I(x, y, t)$ for each pixel (x, y) as a function of time t. Figure 2(b) shows the desired output of this step, where the shadow crossing times are quantized to 32 values (with blue indicating earlier times and red indicated later ones). Note that you may want to include some additional heuristics to reduce false detections. For instance, dark regions cannot be reliably assigned a shadow time—thus, you should eliminate pixels with very low contrast $I_{\max}(x, y) - I_{\min}(x, y)$.

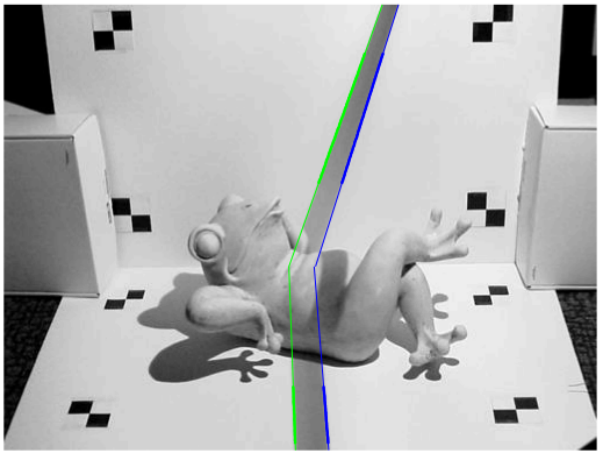
In your submission, show a few examples of spatial and temporal edge localizations, analogous to those shown in Figure 2.

1.2 Intrinsic and extrinsic calibration (50 points)

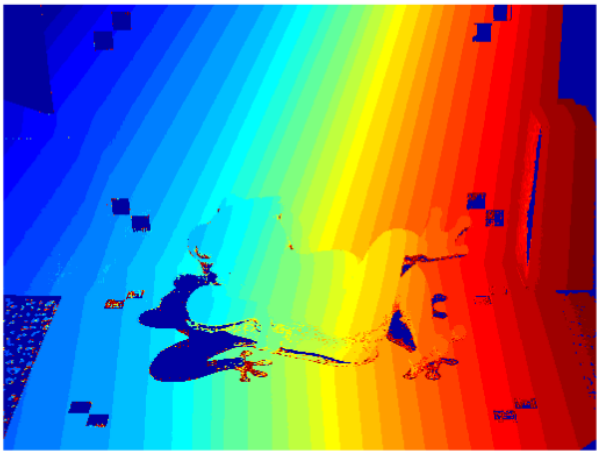
You will need the intrinsic and extrinsic calibration of the camera and scene in order to transfer image measurements into the world coordinate system. We've provided a modified version of the calibration pipeline used in the *Camera Calibration Toolbox* for Matlab, also created by Jean-Yves Bouguet. We provide the functions in in cp_hw6.py, and a demo of the process on the low-resolution frog example in `calibrationDemo.py`. The intrinsic and extrinsic parameters are estimated by capturing several images of a checkerboard at various poses.

Intrinsic calibration. The function `computeIntrinsic()` takes in a stack of at least ten images of a calibration checkerboard at various poses and the dimensions of board. It will then visualize the extracted corners for each captured image. Note that only inner corners are detected, so the provided checkerboard dimensions should *not* be the number of squares but the number of inner corners (6 by 8 for the provided example). In the case where the detected corners are noticeably misaligned from the checkerboard, you will need to adjust the size of the search window used to refine the detected corner locations.

Include the resulting `.npz` file with the intrinsic calibration parameters in your submission.



(a) spatial shadow edge localization



(b) temporal shadow edge localization

Figure 2: Spatial and temporal shadow edge localization. (a) We determine the shadow edges by fitting a line to the set of zero-crossings, along each row in the planar regions, of the difference image $\Delta I(x, y, t)$. (b) We determine the shadow times (quantized to 32 values here) by finding the zero-crossings of the difference image $\Delta I(x, y, t)$ for each pixel (x, y) as a function of time t.

Calibration of ground planes. From the previous step, you have an estimate of how pixels can be converted into normalized coordinates (and subsequently rays in world coordinates, originating at the camera center). You will also need to know how to convert 3D coordinate vectors between three coordinate systems: One describing the camera, one at the horizontal plane, and one at the vertical plane. See Figure 3 for a visualization of the three coordinate systems.

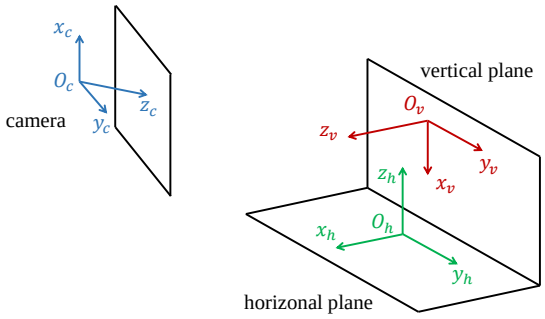


Figure 3: The three world coordinate frames we are concerned with.

You will do this by running `computeExtrinsic()` twice, once for each ground plane. An example of this is shown in `calibrationDemo.py`.

In each run, the demo will allow you to select four corners on the scene plane, to determine the Euclidean transformation from this plane to the camera reference frame. (Always start by selecting the corner in the bottom-left and proceed in a counter-clockwise order. For your reference, the corners define a 558.8 mm × 303.2125 mm rectangle.) Once run, the demo will provide you with a rotation matrix R and translation matrix T that allows you to convert from the camera coordinate system to the plane's coordinate system as

$$P_{\text{plane}} = R^T(P_{\text{camera}} - T), \tag{5}$$

where P_{camera} and P_{plane} are heterogeneous 3D coordinate vectors. The demo shows an example of this, by

using T and R to convert the camera center $O_c = [0, 0, 0]^T$ from the camera to the plane coordinate system.

Report the rotation and translation matrices you obtain for each of the two planes.

Calibration of shadow lines. Now that you have calibrated the two ground planes, you need to, for each frame, find 3D parameterizations for the shadow lines cast on these planes by the moving stick. Figure 4 visualizes the steps for this part.

At each frame, it is sufficient to find two 3D points on each of the two shadow lines, for a total of four 3D points per frame (points P_1, P_2, P_3, P_4 in Figure 4). We describe how to find the points P_1 and P_2 on the shadow line of the horizontal plane, and the procedure for points P_3 and P_4 follows exactly analogously.

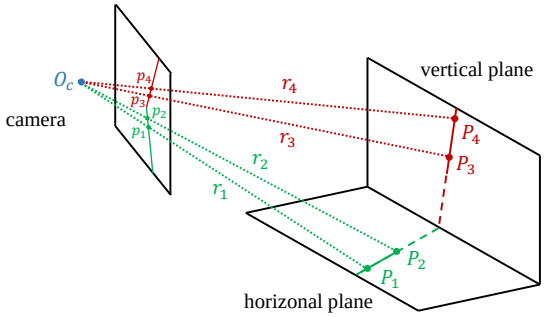


Figure 4: Calibration of shadow lines.

First, select two points p_1 and p_2 on the image that lie on the horizontal shadow line. You computed the equations for this line in Part 1.1, so you can simply interpolate two points using that equation. The extrinsics demo shows how to backproject these pixels into 3D rays, r_1 and r_2, expressed in the camera 3D coordinate system (see provided function `pixel2ray`), as well as how to convert these rays into the 3D coordinate system of the horizontal plane.

You can now determine the 3D point P_1 by performing an intersection, between ray r_1 and the horizontal plane. You should make sure that you use consistent coordinate systems for the intersection (i.e., both the ray and plane are in the coordinate system of the horizontal plane; or both the ray and plane are in the coordinate system of the camera). Once you have P_1, you should convert it to the camera coordinate system as described earlier. Repeat the same procedure for P_2.

You can follow the exactly analogous procedure for extracting points P_3 and P_4, and converting them to the camera coordinate system. You will need to perform this procedure for each frame in your video sequence.

Store the reconstructed 3D points for all the shadow lines in a `.npz` file, and include that in your submission.

Calibration of shadow planes. Finally, you will need to use the four 3D points you computed per frame, to calibrate the corresponding 3D shadow plane. Figure 5 visualizes the steps for this part.

A frame's four points P_1, P_2, P_3, P_4 all lie on that frame's shadow plane. Then, you can use them to express the shadow plane as:

$$S : (P - P_1) \cdot \hat{n} = 0, \text{ where } \hat{n} = \text{normalize}[(P_2 - P_1) \times (P_4 - P_3)], \tag{6}$$

where P is a generic point on the shadow plane. Make sure that the shadow plane is in the camera coordinate system (i.e., by first converting all points $P_1, \ldots, P_4$ to the camera coordinate system).

Store the equations for all 3D lines in a `.npz` file, and include that in your submission. Once you have computed the shadow plane for each frame, you have completed the extrinsic calibration.

1.3 Reconstruction (25 points)

At this point, you have estimated all the parameters required to recover the depth of each pixel in the image (or at least those pixels where the shadow could be observed). See Figure 6 for a visualization of what you

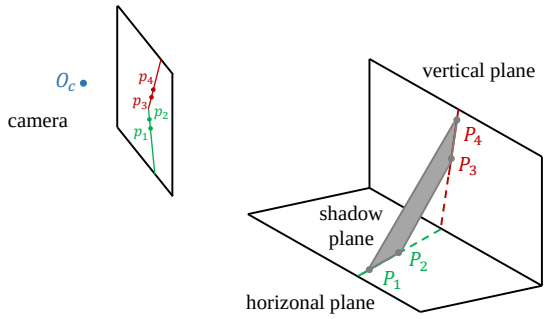


Figure 5: Calibration of shadow planes.

need to do in this section.

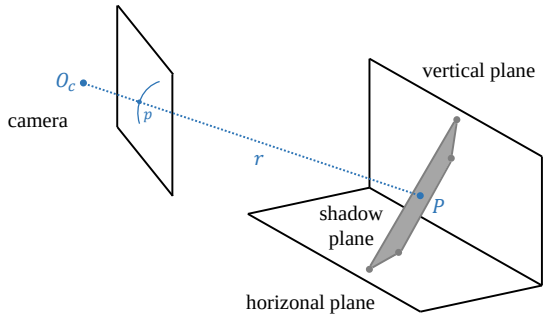


Figure 6: Reconstruction of 3D points.

First, crop the part of the image you want to reconstruct (e.g., a rectangle that includes the whole object). Then, for each pixel $p = (x, y)$ in this rectangle, find the associated frame t where this pixel is on a shadow line—you computed these frames in Part 1.2. Next, fetch the shadow plane $S(t)$ for that frame—you computed these planes in part 1.2.

Backproject the pixel p into a 3D ray r. Finally, intersect this ray with the shadow plane $S(t)$. The resulting intersection point P is your reconstructed 3D point.

Repeat this process for all pixels in your crop, to recover a 3D point cloud, which you will need to visualize. You can use `matplotlib`'s `scatter` function on a 3D plot to convert the reconstructed 3D points into a point-cloud structure. (You can additionally "color" each point in your point cloud, by assigning to it its intensity in a frame where it is not shadowed.) You can then display this structure using the command `imshow`. To give you some expectation of reconstruction quality, Figure 7 shows the results we obtained with our reference implementation. Note that there are several choices you can make in your implementation; some of these may allow you to obtain additional points on the surface or increase the reconstruction accuracy. Please document the methods you used to optimize your reconstruction.

2 Building your own 3D scanner (100 points)

You will now build your own version of the weakly-structured light 3D scanner. You can replicate the setup of Figure 1, using a desk lamp, and the class camera and tripod.

You will additionally need to print a checkerboard for performing camera calibration on your own. We recommend using the same checkerboard configuration (in terms of number of boxes and their dimensions)

Figure 7: Reconstruction results for the frog sequence.

as in the data sequence provided with the homework.

Finally, in setting up the scanner, you will need to create the configuration of the two planes. You should use appropriate holders (e.g., thick books) to ensure that the vertical plane is as close to orthogonal to the floor as possible. You should also mark the corners of a rectangle of known dimensions on each plane, to simplify calibration.

Use your 3D scanning setup to scan at least two objects, and include images of the scanned images and the final reconstruction. Additionally, include a photograph of the setup you built.

3 Bonus: Implement the dual-space geometry procedure (50 points)

The algorithms described in Bouguet and Perona [1] for the calibration and reconstruction parts of the 3D scanning procedure are significantly different from those you were asked to implement above. In particular, they use a so-called *dual-space geometry* formulation, that provides increased robustness. For example, when calibrating shadow lines and planes, you used only two points per line and four points per plane, despite the fact that you had two entire lines available to you, resulting in discounted information and decreased robustness. Dual-space geometry is an elegant formulation for making full use of all the information available to you in this problem.

Read through the paper, and implement the proposed algorithm for calibration and reconstruction. In your write-up, provide a short explanation of how the procedure works, and show the reconstructions you obtain by applying it on the dataset provided with the homework, as well as the two datasets you capture with your own 3D scanner. Compare with the 3D reconstructions you obtained earlier, and discuss your observations.

4 Bonus: Implement direct-global separation (100 points)

As we discussed in glass, Nayar et al. [2] show how to use the stick-shadow procedure to produce pairs of direct-only and global-only images for a scene. Read through this paper, and implement the procedure discussed in Section 4.2 to produce direct-only and global-only images for two scenes (you do not need to implement the photometric stereo part discussed in that section).

Note that applying the technique of this paper requires having access to radiometrically linear images. Unfortunately, the class camera does not allow you to capture RAW video. Therefore, in order to apply the equations used in the paper, you will need to perform radiometric calibration and convert the non-linear frames you extract from the video sequences into linear ones. In Homework 2, you implemented radiometric calibration on your own, but here, you can use OpenCV's implementation (see function `createCalibrateDebevec`). This function requires as input a non-linear exposure stack, which you can capture with your camera. Make sure that you capture the exposure stack under the exact same exposure settings as the ones you will use for

your videos. Show the camera response functions you captured, as well as a collage of your exposure stack before and after linearization.

Apply the linearization and direct-global separation procedures on two scenes, and show for each scene four images: A regular image of the scene captured with your camera and without any shadows; an image showing the stick shadow; and the pair of direct-only and global-only images. The total number of points you will get for this part will depend on how visually compelling the direct-global separation results are.

Deliverables

When submitting your solution, make sure to follow the homework submission guidelines available on the course website (http://graphics.cs.cmu.edu/courses/15-463/assignments/submission_guidelines.pdf). Your submitted solution should include the following:

- A PDF report explaining what you did for each problem, including the various visualizations of shadow planes and point clouds that are requested throughout Parts 1 and 2, as well as answers to all questions asked throughout both problems. The report should include any figures and intermediate results that you think may help. Make sure to include explanations of any issues that you may have run into that prevented you from fully solving the assignment, as this will help us determine partial credit.
- All of your Python code, as well as a `README` file explaining how to use the code.
- All `.npz` files requested through Parts 1 and 2.
- If you do Bonus Part 3: all your python code, as well as the discussion and figures requested in that part.
- If you do Bonus Part 4: all your python code, as well as the discussion and figures requested in that part.

Hints and Information

- We recommend that you use version 4.5.0 or newer of OpenCV.
- The comments of file `calibrationDemo.py` include specifications for the checkerboard you should use.
- Your reconstruction results may look very off because of very many small bugs. A good way to debug this assignment is as follows: After each step, make sure to visualize in 3D the reconstructed points, lines, and planes. For example, after computing the 3D points on the shadow lines, plot them in 3D and make sure they look reasonable. After computing the 3D shadow lines themselves, plot them in 3D and make sure they look reasonable. After computing the 3D shadow planes, plot them in 3D and make sure they look reasonable.
- The 3D visualization of your reconstructed points can be strongly affected by how many outliers you are leaving unpruned. For example, if there is one outlier that ends up having a very large depth, it may cause everything else in your 3D visualization to appear to be on a single plane. This is because the outlier ends up forcing `matplotlib` to use a very large depth range for the z axis, and thus compressing the depth of correctly reconstructed points. You should make sure to experiment with different contrast thresholds, and thus different amounts of outlier pruning, while evaluating your reconstruction results.
- When extracting points from the calibration markers for plane calibration, you should extract the middle corner. Do not use the outer four corners.
- Figure 3 shows what plane each coordinate system corresponds to. The origins and axes shown in the figure are not necessarily representative of where they will be in reality. You should use whatever origins/axes match the result of the calibration.

- If you use `matplotlib` to view your temporal shadow edge localization image, you may see some unexpected values at the junction between valid and invalid pixels (e.g., some green between dark blue and red values in the jet colormap). By default, `matplotlib` will use interpolation to smooth out the image for display, thus producing spurious values that aren't actually in your result. You can disable this by setting `interpolation=None`.

- When you first implement temporal shadow localization, it is OK to skip interpolation for the zero-crossings—the localization results without interpolation should be accurate enough for the rest of the pipeline to produce decent 3D reconstruction results. However, for full credit, your solution should include interpolation, so you should remember to come back and fix that part after you have implemented everything else.

- You do not need to process frames where the shadow does not intersect at all with the 3D object.

- When extracting shadow lines, it typically helps to crop two parts of the frame sequence, one for the vertical and one for the horizontal plane, where there are no occlusions or shadowing by the 3D object, or texture by the calibration markers. Then, for each frame, you can fit shadow lines by detecting shadow-nonshadow transitions only in this cropped part.

- When visualizing your point cloud, you should set the axes aspect ratio to be equal, otherwise it will be difficult to see whether your results are truly reasonable. In order to do this in `matplotlib`, you must use function `set_box_aspect` to match the aspect ratio of the displayed 3D grid. We have provided the function `set_aspect_equal`, which will attempt to find reasonable axes limits for the plotted data that will match a $(1, 1, 1)$ box aspect ratio. You may want to expand upon this function.

- When building your own version of the 3D scanner, you should note some practical issues associated with this approach.

  First, it is important that every pixel be shadowed at some point in the sequence. As a result, you must move the stick slowly enough to ensure that this condition holds.

  Second, the reconstruction method requires reliable estimates of the plane defined by the light source and the edge of the stick. Ambient illumination must be reduced so that a single planar shadow is cast by each edge of the stick, otherwise your shadow estimates will be off.

  Third, The light source you use must be sufficiently bright to allow the camera to operate with reasonable exposures and minimal gain, otherwise sensor noise will corrupt the final reconstruction. It is best to use a small lamp, such as a desk lamp or similar. This ensures that the light source is sufficiently point-like to produce abrupt shadow boundaries. Otherwise, the estimate of the shadow plane will not be reliable.

  Fourth, when calibrating your own camera, it is important to ensure planarity of the checkerboard pattern. We recommend that you stick the pattern on a flat surface (e.g., a wooden panel). Additionally, it is important that you capture a sufficient number of images, spanning a large variety of checkerboard poses everywhere in the field of view of your camera. The calibration sequence we provide in the homework should give you a sense of what sort of images you need.

  Fifth, in a departure from previous homework assignments, here it is not necessary to use RAW images. The class camera allows you to capture non-RAW video, from which you can extract frames using either Python, or a utility such as `ffmpeg`. (If you do Bonus Part 4, see the discussion there about radiometric calibration.)

  Finally, you should set the focal length, focus, and aperture settings of your lens appropriately, so that all the scanning setup is within your field of view and sharply in focus. Blurry regions will result in poor shadow estimates, and therefore inaccurate reconstruction. Additionally, all lens parameters should remain constant throughout capture, so make sure to set everything to manual.

Credits

This homework is directly adapted from the 3D photography class offered by Gabriel Taubin at Brown and modified for python using the OpenCV camera calibration tutorial. This includes the write-up, figures, and data.

References

[1] J.-Y. Bouguet and P. Perona. 3d photography using shadows in dual-space geometry. *International Journal of Computer Vision*, 35(2):129–149, 1999.

[2] S. K. Nayar, G. Krishnan, M. D. Grossberg, and R. Raskar. Fast separation of direct and global components of a scene using high frequency illumination. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 935–944. ACM, 2006.