Homework assignment 6

15-463, 15-663, 15-862 Computational Photography, Fall 2023
Carnegie Mellon University

Due Friday, Dec. 8, at 11:59pm ET

The purpose of this assignment is to use structured light to 3D scan an object. Rather than use a projector, you will rely on shadows to create structured illumination, a technique known as "weakly structured light".

In particular, you will implement the "desktop 3D scanner" of Bouguet and Perona [1]. As Figure 1 shows, this 3D scanner has five primary components: a camera, a point-like light source (e.g., desk lamp, cell phone flash), a stick, two planar surfaces, and a calibration checkerboard. By waving the stick in front of the light source, you cast line shadows into the scene. As Bouguet and Perona show, this makes it possible to recover the depth at each pixel using simple geometric reasoning.

We strongly encourage you to carefully go through Bouguet and Perona [1] before starting the assignment. However, note that the calibration and reconstruction procedures (Parts 1.2 and 1.3) we use in the assignment are significantly different from (and simpler than) those in the paper.

Towards the end of this document, you will find a "Deliverables" section describing what you need to submit. Throughout the writeup, we also mark in red questions you should answer in your submitted report. Lastly, there is a "Hints and Information" section at the end of this document that is likely to help. We strongly recommend that you read that section in full before you start to work on the assignment.

1 Implementing structured-light triangulation (100 points)

For the first part of this assignment, you will be using two image sequences contained in the `./data` directory of the assignment ZIP archive. The `calib` directory contains a calibration sequence includes ten images of a checkerboard at various poses; you will use this sequence to estimate the intrinsic and extrinsic parameters of the camera. The `frog` directory contains a sequence of images of the frog object in Figure 1; you will use this sequence for 3D reconstruction. For each sequence we have provided both low-resolution (512×384) and high-resolution (1024×768) versions. You can use the former during development and debugging, and the latter for your final results. You should convert these color images to grayscale, e.g., using `rgb2gray`.

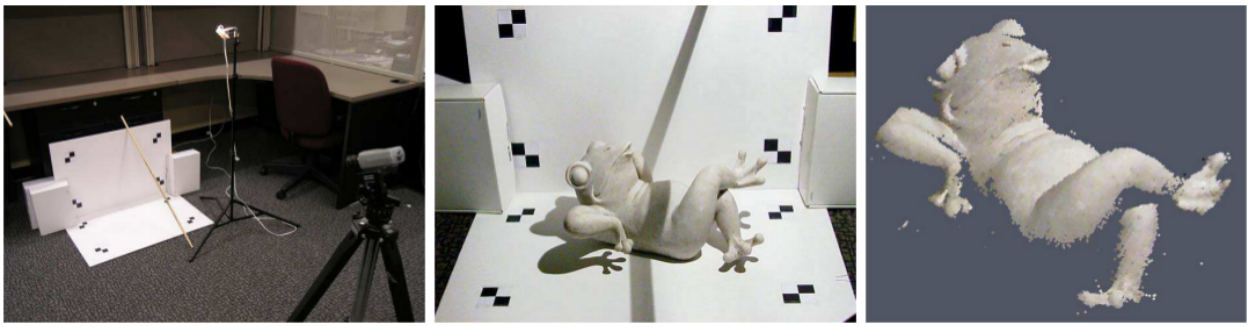
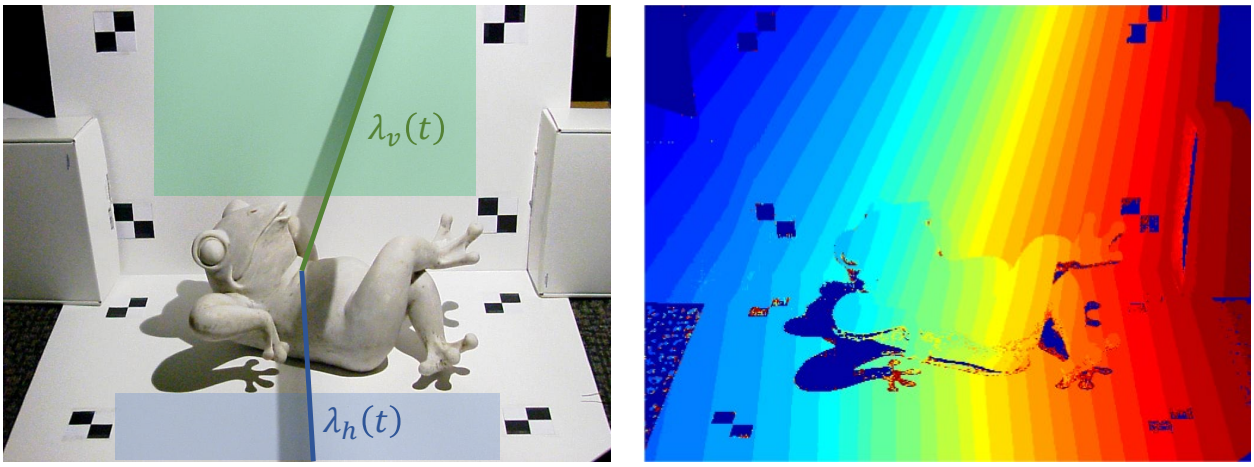


Figure 1: 3D photography using planar shadows. From left to right: the capture setup, a single image from the scanning sequence, and a reconstructed object (rendered as a colored point cloud).

1.1 Video processing (25 points)

Throughout this part, you will use the `frog` image sequence. We will index the frames in this sequence using $t \in 1, \ldots, T$, where T is the total number of frames. We will often refer to the frame index t as "time". Your first task is to estimate two important quantities this sequence: (i) the *per-frame shadow edges*, which are

the parameters of the horizontal and vertical shadow lines within the horizontal and vertical planar regions, respectively, of each frame t; and (ii) the *per-pixel shadow times*, which are the frames t when a shadow edge first crosses each pixel. The following sections outline how to perform these tasks. You can consult Section 2.4 in Bouguet and Perona [1] for additional information.



(a) per-frame shadow edge estimation (b) per-pixel shadow time estimation

Figure 2: Estimation of per-frame shadow edges and per-pixel shadow times. (a) For each frame t, we determine the per-frame shadow edges by fitting lines to the zero-crossing points of the difference image $\Delta I(x, y, t)$, along each row in the unobstructed horizontal and vertical planar regions (highlighted in blue and green, respectively). (b) For each pixel (x, y), we determine the per-pixel shadow times by finding the zero-crossings of the difference image $\Delta I(x, y, t)$ as a function of time t. (The figure shows the per-pixel shadow times quantized to 32 values, with blue indicating earlier times and red indicating later ones.)

Per-frame shadow edge estimation. As Figure 2(a) shows, for each frame t, you need to estimate the lines $\lambda_h(t)$ and $\lambda_v(t)$, running along the right shadow edges on the unobstructed 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.

To describe the procedure for estimating the shadow edges, we begin by defining the maximum and minimum intensity observed at each pixel (x, y):

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

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

We also define a per-pixel *shadow threshold image* as:

$$I_{\text{shadow}}(x, y) \equiv \frac{I_{\max}(x, y) + I_{\min}(x, y)}{2}. \tag{3}$$

Lastly, for each time t and pixel (x, y), we define the *difference image* as:

$$\Delta I(x, y, t) \equiv I(x, y, t) - I_{\text{shadow}}(x, y). \tag{4}$$

To estimate the per-frame shadow edges, you will work with the portion of the frames where the horizontal and vertical planar regions are unobstructed by the object you are scanning. Figure 2(a) shows these unobstructed regions. These unobstructed regions are the same across frames, so you can approximately estimate their x and y limits once, then use these limits across all frames.

For each frame t, you can estimate its horizontal shadow edge $\lambda_h(t)$ as follows: For every row y in the unobstructed horizontal planar region, find the column x where the difference image $\Delta I(x, y, t)$ changes sign from negative to positive. This will give you a list of *zero-crossing* locations $\{(x_n, y_n)\}_{n=1}^{N}$, where N is the number of rows in the unobstructed horizontal planar region. Lastly, estimate the shadow edge by fitting a line to these zero-crossing locations. For this, represent the horizontal shadow edge as a vector, $\lambda_h(t) \equiv (a_h(t), b_h(t), c_h(t))$, corresponding to the line $a_h(t)x + b_h(t)y + c = 0$. Compute this vector by solving the *homogeneous* linear system:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & 1 \end{bmatrix} \begin{bmatrix} a_h(t) \\ b_h(t) \\ c_h(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{5}$$

You can estimate the vertical shadow edge $\lambda_v(t) \equiv (a_v(t), b_v(t), c_v(t))$ for frame t by repeating the exact same procedure for the unobstructed vertical planar region.

Repeat the above procedure for all frames $t \in 1, \ldots, T$, and store the horizontal and vertical shadow edges you estimate. Figure 2(a) shows the desired output of this step for one of the frames, overlaying the estimated shadow edges on the original frame.

Per-pixel shadow time estimation. To perform 3D reconstruction as in Section 1.3, you need to also estimate the per-pixel shadow times. For each pixel (x, y), you can estimate its shadow time $t_{\text{shadow}}(x, y)$ by finding the zero-crossing of the difference image $\Delta I(x, y, t)$ as a function of time t. Figure 2(b) shows the desired output of this step, after quantizing the shadow crossing times to 32 values.

In your submission, show a few examples of per-frame shadow edge estimates and per-pixel shadow time estimates, similar to those in Figure 2.

1.2 Intrinsic and extrinsic calibration (50 points)

You will need the intrinsic and extrinsic parameters of the camera and scene, to map 2D image locations into 3D points. The `./src` directory of the assignment ZIP archive includes a modified version of the OpenCV camera calibration implementation. This implementation uses several images of a checkerboard captured at various poses to estimate the intrinsic and extrinsic parameters of the camera. The file `cp_hw6.py` has most relevant functions, and the file `calibrationDemo.py` has a demo of the process on the low-resolution calibration image sequence `calib-lr`.

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 the checkerboard. It then extracts and visualizes corners for each checkerboard image. The function only detects the inner corners of the checkerboard, so the provided checkerboard dimensions should *not* be the number of squares but the number of inner corners (6×8 for the example calibration sequence). If the detected corners and the checkerboard do not align well, you will need to adjust the size of the search window used to refine the detected corner locations.

In your submission, include an `.npz` file with the resulting intrinsic calibration parameters.

Calibration of ground planes. From the previous step, you have an estimate of the camera intrinsic parameters, which you can use to convert image pixels into 3D rays in the camera 3D coordinate system. You will also need to know how to convert coordinates of 3D points and directions between three 3D coordinate systems: the camera coordinate system, a coordinate system at the horizontal plane, and a coordinate system at the vertical plane. Figure 3 visualizes the three 3D coordinate systems.

To estimate the parameters needed to perform these conversions, you will run `computeExtrinsic()` twice, once for each ground plane. The file `calibrationDemo.py` shows an example of this.

In each run, the demo allows you to select four corners on the calibration markers attached to the scene plane. Always start by selecting the corner at the bottom-left of the plane, and proceed in a counter-clockwise order. For your reference, the corners define a 558.8 mm $\times$ 303.2125 mm rectangle. Note that you need to select the corner at the center of each calibration marker, as Figure 4 shows.

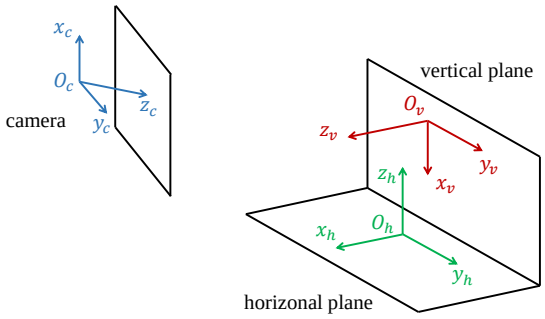


Figure 3: The three world coordinate frames we are concerned with.

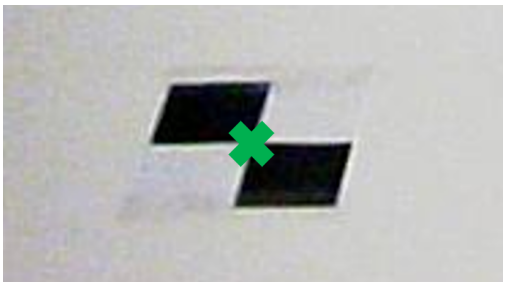


Figure 4: For each calibration marker on the planes, select the marked corner at its center.

Once run, the demo will provide you with a rotation matrix R and a translation matrix T. You can use these to convert 3D points from the camera 3D coordinate system to the plane 3D coordinate system as

$$P_{\text{plane}} = R^{\top}(P_{\text{camera}} - T), \tag{6}$$

where P_{camera} and P_{plane} are heterogeneous 3D coordinate vectors. You can also apply the inverse transform to convert 3D points from the plane 3D coordinate system to the camera coordinate system.

In your submission, include an `.npz` file with the rotation and translation matrices for each ground plane.

Calibration of shadow lines. Having calibrated the two ground planes, now you need to, for each frame, estimate the 3D shadow lines cast on these planes by the stick. Figure 5 visualizes the steps for this part.

For each frame t, it is sufficient to find two 3D points on each of the two 3D shadow lines, for a total of four 3D points per frame (points P_1, P_2, P_3, P_4 in Figure 5). 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.

First, compute two 2D image points p_1 and p_2 on the horizontal shadow edge $\lambda_h(t)$ you estimated in Part 1.1. Use the provided function `pixel2ray` to *backproject* these 2D points into 3D rays, r_1 and r_2 in the camera 3D coordinate system. Then, convert these rays in 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). Once you have P_1, convert it to the camera coordinate system as described earlier. Repeat the same procedure for P_2.

Use the exactly analogous procedure to extract points P_3 and P_4, and convert them to the camera coordinate system. You will need to perform this procedure for each frame t in your image sequence.

In your submission, include an `.npz` file with the reconstructed 3D points for all frames.

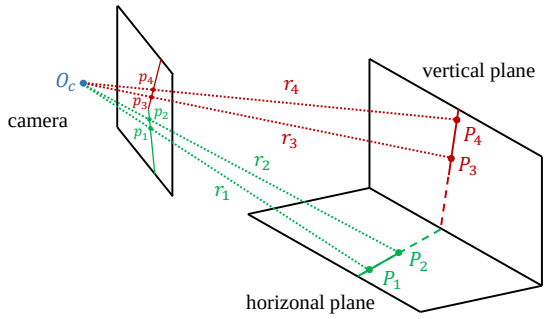


Figure 5: Calibration of shadow lines.

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 6 visualizes the steps for this part.

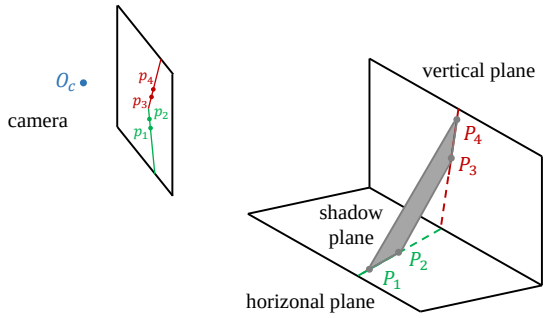


Figure 6: Calibration of shadow planes.

A frame's points P_1, P_2, P_3, P_4 lie on that frame's shadow plane. You can express the shadow plane as:

$$S : (P - P_1) \cdot \hat{n} = 0, \text{ where } \hat{n} = \text{normalize}[(P_2 - P_1) \times (P_4 - P_3)], \tag{7}$$

where P is any 3D point on the shadow plane. Make sure that the shadow plane is in the camera coordinate system, by first converting all points $P_1, \ldots, P_4$ to the camera coordinate system.

In your submission, include an `.npz` file with the estimated shadow plane parameters (points P_1 and normals $\hat{n}$) for all frames. After computing the shadow planes, you have completed the extrinsic calibration.

1.3 Reconstruction (25 points)

You have estimated all the parameters you need to recover the depth of each pixel in the image (or at least those pixels where the stick shadow can be observed). Figure 7 visualizes what you need to do in this section.

First, crop the part of the image you want to reconstruct (e.g., a rectangle around the entire object). For each rectangle pixel $p = (x, y)$, fetch its shadow frame $t_{\text{shadow}}(x, y)$—you computed these frames in Part 1.1. Next, fetch the shadow plane $S(t_{\text{shadow}}(x, y))$ 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 $S(t_{\text{shadow}}(x, y))$. The resulting intersection point P is the reconstructed 3D point corresponding to pixel p.

Repeat this process for all pixels in your cropped image, to recover a 3D point cloud. Then, use `matplotlib`'s `scatter` function to visualize the point cloud in 3D. You should "color" each point in your

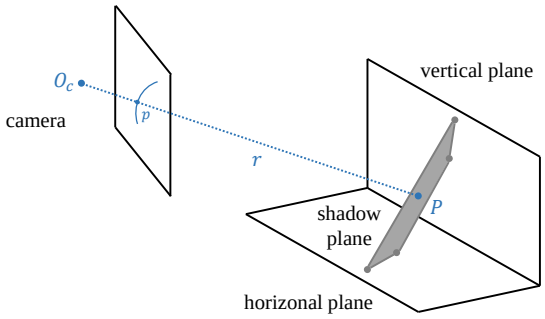


Figure 7: Reconstruction of 3D points.

point cloud, by assigning to it its intensity value in a frame where the corresponding pixel is not shadowed. Figure 8 shows the results we obtained with our reference implementation. Note that your reconstruction may differ significantly from the one in the figure, depending on various choices you make in your implementation. In your submission, make sure to document any such choices you made to improve your reconstruction.

Figure 8: Reconstruction results for the frog sequence.

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 or cell phone flash as the light source, and the class camera and tripod.

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

Finally, you will need to create the two planes. You should use appropriate holders (e.g., thick books) to make the vertical plane as close to orthogonal to the horizontal plane (which is on the floor) as possible. You should also mark the corners of a rectangle of known dimensions on each plane.

Use your 3D scanning setup to scan at least two objects, and include images of the scanned objects 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 implemented in Parts 1.2 and 1.3. In particular, Bouguet and Perona use a *dual-space geometry* formulation, which provides increased robustness to small calibration errors. For example, when calibrating shadow lines and planes, you used only two points per line and four points per plane, even though you had two entire lines available; the consequence of this is discounted information and decreased robustness. Dual-space geometry is an elegant formulation for making full use of all the information available to you for 3D reconstruction.

Read through Bouguet and Perona [1], and implement the algorithm they propose for calibration and reconstruction. In your write-up, provide a short explanation of how the algorithm works, and show the reconstructions you obtain by applying it on the dataset we provide in the assignment ZIP archive, and 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-indirect 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 indirect-only images for a scene. Read through this paper, and implement the procedure of Section 4.2 to produce direct-only and indirect-only images for two scenes. (You do not need to implement the photometric stereo part that section discusses.)

Applying the technique of this paper requires radiometrically linear images. Unfortunately, the class camera does not allow you to capture RAW video. Therefore, to apply the equations from 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 Assignment 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 same exposure settings as those you will use for your videos. Show the camera response functions you captured, and a collage of your exposure stack before and after linearization.

Apply the linearization and direct-indirect 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 indirect-only images. The total number of points you will get for this part will depend on how visually compelling the direct-indirect 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 requested throughout Parts 1 and 2, as well as answers to all questions asked throughout both parts. 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, and the discussion and figures requested in that part.
- If you do Bonus Part 4: all your Python code, and the discussion and figures requested in that part.

- For the photography competition: Submit an image of one of the reconstructed objects you scanned for Problem 2, named as `competition_entry.png`.

Hints and Information

OpenCV version. We recommend that you use version 4.5.0 or newer of OpenCV.

Checkerboard. The comments in `calibrationDemo.py` include specifications for the checkerboard.

Debugging. 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, 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.

Outliers. The 3D visualization of your reconstructed point cloud can be strongly affected by outliers. 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 forces `matplotlib` to use a very large depth range for the z axis, and thus compress the depth of correctly reconstructed points.

These outliers are primarily due to points that are never shadowed by the stick (e.g., pixels under the shadow of the scanned object). You can remove these outliers by eliminating 3D points for any pixels (x, y) with contrast $I_{\max}(x, y) - I_{\min}(x, y)$ below some threshold. You should experiment with different contrast thresholds, and thus different amounts of outlier pruning, while evaluating your reconstruction results.

Writeup figures. 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 your results. You should use whatever origins and axes match the result of the calibration.

Interpolation artifacts. If you use `matplotlib` to visualize the per-pixel shadow times, you may see unexpected values at the junction between valid and outlier 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 these spurious values. You can disable this by setting `interpolation=None`.

Redundant frames. You can skip frames where the stick shadow does not intersect the object.

Shadow edges. When you extract per-frame shadow edges, it helps to crop two parts of the frame sequence, for the vertical and horizontal plane, where there are no occlusions or shadowing by the object, or texture by the calibration markers. Then, you can estimate shadow edges by detecting zero-crossings in only these crops.

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

Building your own 3D scanner. Below are a few tips to help you build your own 3D scanner.

First, it is important that every pixel be shadowed at some frame in the video sequence you capture. Achieving this will require that you move the stick at a slow enough pace.

Second, you should reduce any ambient illumination (i.e., light that is not coming from the light source in your setup). Otherwise, ambient light can reduce the accuracy of shadow planes estimation. If the stick casts multiple shadow lines, or their contrast is very low, then this is likely a problem with ambient illumination.

Third, your light source must be sufficiently bright to allow reasonable exposures and low ISO. Otherwise, sensor noise will corrupt the final reconstruction. Additionally, you light source should be sufficiently point-like, to produce sharp stick shadows. Otherwise, your shadow plane estimation will not be accurate. A good source to use is a small desk lamp, or the flash of your cell phone (if it is bright enough).

Fourth, when calibrating your own camera, the checkerboard pattern you use must be sufficiently planar. We recommend that you stick the printed checkerboard pattern on a flat surface (e.g., a wooden panel). Additionally, you should make sure to 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 assignment ZIP archive should give you a sense of what sort of images you need.

Fifth, in a departure from previous 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 plane estimation, and therefore inaccurate reconstruction. Additionally, all lens and camera parameters should remain constant throughout capture, so make sure to disable autofocus, disable auto white balancing, and set the exposure mode to manual.

Credits

This assignment is directly adapted from the 3D photography class 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.