MIT CSAIL

6.869 Advances in Computer Vision

Fall 2018

Problem Set 3

Posted: Thursday, September 20, 2018 **Due:** Thursday, September 27, 2018

Submission Instructions: Please submit **two separate files:** 1) a report named ⟨your_kerberos⟩.pdf, including your responses to all required questions with images and/or plots showing your results, 2) a file named ⟨your_kerberos⟩.zip, containing relevant source code. **Submissions that do not adhere to these instructions are subject to an additional penalty.**

Late Submission Policy: We do not accept late submissions. The submission deadline has a 50-minute soft cut-off; after midnight Thursday, submissions are penalized 2% per minute late.

Collaborators: You are free to discuss problems with other students but all writing must be done individually. Please list all collaborators at the top of your report.

Problem 1: Color

(a) Given a set of color primaries in the matrix, P, and given the color matching functions for the eye responses, C_{eye}, write an expression for the color matching functions, C, associated with the primaries P.

(b) Given a set of color matching functions, C, find an expression for an associated set of color primaries, P, in terms of the eye response functions, C_{eye}.

(c) Show that if the spectral response curves of the eye (assumed to be nonnegative) were orthogonal to each other (with a zero dot product), there would exist a corresponding set of primaries with power spectra that were nonnegative. *Hint:* Try to construct nonnegative primaries from the color matching functions.

(d) *6.869 only:* For a set of primaries, P, and an associated set of color matching functions, C: If the primaries P are specified, are the color matching functions C uniquely determined? If the color matching functions are specified, are the corresponding primaries P uniquely determined?

Problem 2: Structured light

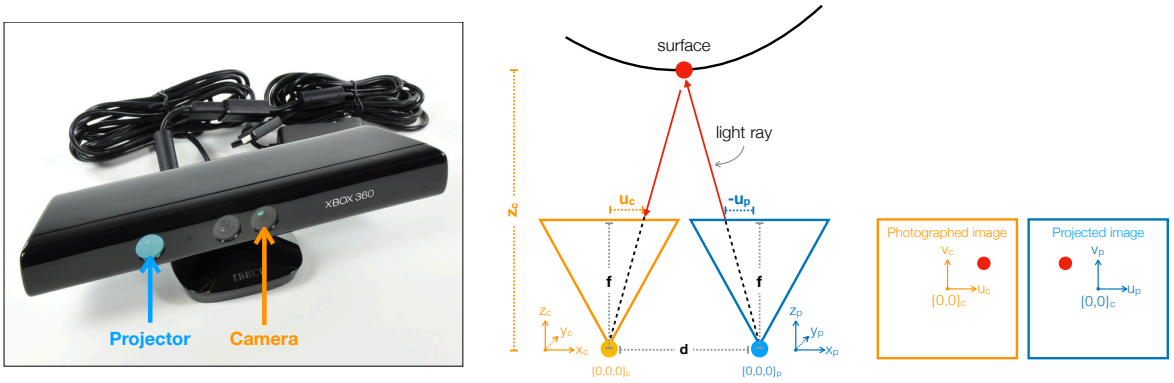

Figure 1: Left: A Kinect depth sensor, which consists of a projector and a camera (note: there is also a second camera used to get better quality color images) [image source: https://www.ifixit.com/Teardown/Microsoft-Kinect-Teardown/4066/1]. Middle: The geometry of the projector-camera system we will consider. Right: The images seen from the perspective of the camera and the projector, when we project a single red laser.

In this problem we will simulate a structured light depth camera, like the Microsoft Kinect. The Kinect consists of a camera and a projector, as shown in Figure 1. The projector shines a structured pattern of light on the world (in the infrared spectrum so that it is invisible to our eyes). The camera can see this pattern and observes how the light warps over the scene geometry. From the pattern of distortions, it is possible to recover a depth map. Let's see how this can be done.

(a) Using the geometry in Figure 1, write an expression for the world coordinates of the indicated surface point, given that the projector emits a light ray that intersects its virtual image plane at location $\{u_p, v_p\}$, and the camera sees the reflected light on its image plane at location $\{u_c, v_c\}$. Assume the camera and projector are offset a distance d from each other, in the x direction. The focal length of the camera is f. uv-coordinates are represented in the same units as f (in the code, we will use pixels as the units). We will consider two kinds of world coordinates: $\{x_c, y_c, z_c\}$ are coordinates relative to the camera center (i.e. the camera center, which is the yellow dot, is at zero in this reference frame). $\{x_p, y_p, z_p\}$ are coordinates relative to the projector center. Write expressions for both $\{x_c, y_c, z_c\}$ and $\{x_p, y_p, z_p\}$ as a function of $\{u_c, v_c\}$, $\{u_p, v_p\}$, d, and f.

This equation shows the basic principle we will use to infer depth! We will shine a ray of light out of the projector at coordinate $\{u_p, v_p\}$, then find where the camera observes, in camera image coordinates $\{u_c, v_c\}$. These two coordinates allow us to triangulate the world coordinates of the object that the ray reflected off of.

(b) Now we will start simulating this setup in Matlab. Starter code is provided in `code.m`. You will be filling in pieces of missing code.

Your first task is to simulate what the camera will see if we turn off all the lights and shine a "laser" out of the projector (i.e. have the projector project a single ray of red light, like in Figure 1). You are given a depth map z_p and the direction $\{u_p, v_p\}$ in which we project the laser. To simulate what the camera sees, we need to find where, in the camera image,

the surface point illuminated by the laser shows up. This means we need to transform from $\{u_p, v_p\}$ to $\{u_c, v_c\}$ given known scene geometry z_p.

Note: for now, we are pretending that no rays are occluded, we will demonstrate how to deal with occlusions later in the pset.

As an exercise, we will perform this transformation in homogeneous coordinates. The file `getTransformationMatrices.m` contains four missing transformation matrices, which operate on homogeneous coordinates. In turn, these mappings are:

- $T_1 : \{u_p, v_p, 1/z_p\}$ to $\{x_p, y_p, z_p\}$ ◁ projector image to projector world
- $T_2 : \{x_p, y_p, z_p\}$ to $\{x_c, y_c, z_c\}$ ◁ projector world to camera world
- $T_3 : \{x_c, y_c, z_c\}$ to $\{u_c, v_c\}$ ◁ camera world to camera image
- $T_4 : \{u_p, v_p, 1/z_p\}$ to $\{u_c, v_c\}$ ◁ projector image to camera image

You should write the matrix T_4 as a composition of T_1, T_2, and T_3. The ability to do this kind of algebra on transformations is one of the main reasons we like homogeneous coordinates.

(c) Now, run the code in the section `%% (c) laser experiment`, which will use the transformations you defined to simulate the laser scanning over a room.

Why does the path of the laser wiggle around from the camera's view? Try changing the scan path (scan left to right rather than top to bottom). Does the laser still wiggle? Why or why not? Write a short explanation to explain why the path of the laser is qualitatively different when you scan left to right compared to when you scan top to bottom.

(d) Now we will try to infer depth from the simulated image of the laser scanning over the room. Suppose we shine the laser through $\{u_p, v_p\}$, and we observe it in the camera at location $\{u_c, v_c\}$. Use your expression from part (a) to fill in the code in the file `inferDepthFromMatchedCoords.m`. This function computes z_c from $\{u_p, v_p\}$, $\{u_c, v_c\}$, f, and d.

(e) In Matlab, run the code in the section `%% (e) inferring depth in the laser illuminated world`, which will use the function you wrote for part (d). Verbally describe what you observe, and share a screen capture of the result.

(f) Next, we will move away from lasers and describe "structured light". Scanning across a scene with a laser is a slow process, and lasers are expensive. The idea of structured light is to project many rays of light at once – a whole projected image of light – in a pattern that allows us to identify which ray of light ($\{u_p, v_p\}$) created the illumination observed by the camera at each point $\{u_c, v_c\}$. Recall from above, that if we know corresponding points in the projector's image and the camera's image, then we can solve for depth.

In the Matlab code section `%% (f) structured light`, we have provided a pattern of stripes `L_p_img`. Your task is to illuminate the scene with this pattern of light.

We have also provided a function `render`. This function takes as input a list of the coordinates of all pixels in the image to be projected (each such pixel has a coordinate $\{u_p, v_p\}$), and the intensity of light projected through each of these pixels (represented via the image `L_p_img`). `render` also requires the list of coordinates that these pixels map to in the camera image frame (the list of corresponding $\{u_c, v_c\}$). You should fill in `setupForRendering.m` to generate these lists.

`render` is very similar to the laser rendering code you used above, but handles occlusions using a "depth buffer", which checks for each surface point if it is the nearest point to the camera among all points at that camera coordinate.

Try illuminating the scene with a different pattern of light (optional: try projecting a moving pattern of lights, it looks cool). Show your results and describe or mark some regions of the camera's image where occlusion blocks the projected light.

Note: `render` assumes an unrealistic reflectance model, where the amount of reflected light does not depend on surface orientation. In the final part of this pset, we will see what happens if we move to a more realistic, Lambertian, reflectance model.

(g) The next step is to infer depth from the structured light image. Here, you must come up with a pattern of light and a function F, such that you can decode

$$\{u_p, v_p\} = F(L_c(\{u_c, v_c\}), \{u_c, v_c\}),$$

where $L_c(\{u_c, v_c\})$ is the intensity of light the camera sees at pixel $\{u_c, v_c\}$, and this pixel was illuminated by the ray that was projected through $\{u_p, v_p\}$. (Hint: F can be a trivial function if you choose a certain pattern). Write code for F in `F.m`, and write code that generates the light pattern (an image) in `getStructuredLightPattern.m`. Demonstrate success by using the decoded coordinates to estimate depth z_c. Notice that we were able to infer depth just from a single photograph of the scene taken by the camera (we've denoted this photo as L_c)!

Can you think of other light patterns, paired with decoders F, that could be used? Describe a few general strategies (you don't have to implement these additional strategies).

(h) *6.869 only*: As noted above, our simulation is not quite realistic because it assumed all surfaces in the scene reflected the same amount of light regardless of their orientation. We have provided a more realistic renderer in the function `render_Lambertian.m`. This renderer uses a Lambertian reflectance model: $I = N \cdot L$, where I is the intensity of reflected light, N is the surface normal vector, L is the direction of the incoming light ray with respect to the surface normal, and $\cdot$ is the dot product. This equation says that surfaces that are illuminated head on will look brighter than surfaces that are illuminated at an angle.

If we render the scene in this more realistic way, does your code from (g) still work? Why or why not? You can use the `render_Lambertian` function provided in section `%% (h) Lambertian rendering` of the code.

Your task here is to design a new light pattern and new function F that works better in the

presence of Lambertian reflectance. Hint: try illuminating the scene with colored light.