MIT CSAIL

6.819/6.869 Advances in Computer Vision

Spring 2022

Problem Set 2

Posted: Tuesday, February 15, 2022 **Due:** Thursday, February 24, 2022

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⟩.ipynb of the python notebook you filled in with the cells run.

Important!! For any questions involving code, copy the relevant lines into your PDF writeup. Also include any relevant image outputs directly in the PDF. Please draw a box around your solution (`\fbox`).

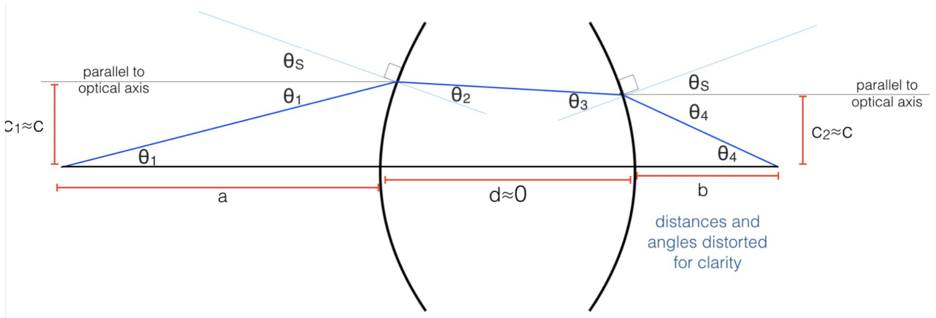
Late Submission Policy: If your pset is submitted within 7 days (rounding up) of the original deadline, you will receive partial credit. Such submissions will be penalized by a multiplicative coefficient that linearly decreases from 1 to 0.5

Problem 1: Lenses

In class, and in the imaging chapter of the class notes, we derived the lensmaker's equation for the case of a symmetric thin lens, of radius of curvature R for the front and back surfaces of the lens. Suppose, instead, that we have a "plano-convex" lens, with the front surface shape with a radius of curvature R, as before, but with a flat back surface.

(a) (1 point) Write down the 5 equations relating the angles of rays passing through a plano-convex lens. The symmetric lens case is shown in Figure 1 (it was also on slide 49 of lecture 2 and page 12 of the lecture 2 (chapter 4) notes). You may use the same variables and assumptions. A simplified diagram of the plano convex lens is shown in Figure 2.

(b) (1 point) Using the equations from part (a) and the lensmaker's formula, find the expression for the lens focal length for this case.



Angle	Description	Relation	Reason
θ_1	initial angle from optical axis	$\theta_1 = \frac{c}{a}$	small angle approx.
θ_2	angle of refracted ray wrt front surface normal	$n\theta_2 = \theta_1 + \theta_S$	Snell's law, small angle approx.
θ_3	angle of refracted ray wrt back surface normal	$2\theta_S = \theta_2 + \theta_3$	symmetry of lens, thin lens approx.
$\theta_S + \theta_4$	angle of ray exiting lens wrt back surface normal	$n\theta_3 = \theta_S + \theta_4$	Snell's law, small angle approx.
θ_4	final angle from optical axis	$\theta_4 = \frac{c}{b}$	small angle approx.

Figure 1: Top: Diagram of a light ray passing through a symmetric thin lens. Bottom: Relations between angles of rays passing through the lens.

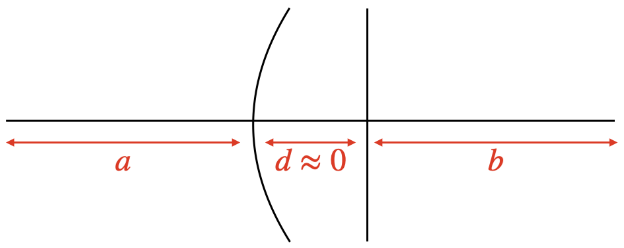


Figure 2: Partial diagram of a plano-convex lens.

Problem 2: Structured light

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 3. 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. We will simulate a scene illuminated by projected light and then show how depth can be inferred from simulated photographs of this scene.

Look at the geometry in Figure 3. The projector emits a light ray that intersects its virtual image plane at location (x_p, y_p), and the camera sees the reflected light on its image plane at location (x_c, y_c).

For math simplicity, we will shift xy-coordinates into uv-coordinates, which turns the image

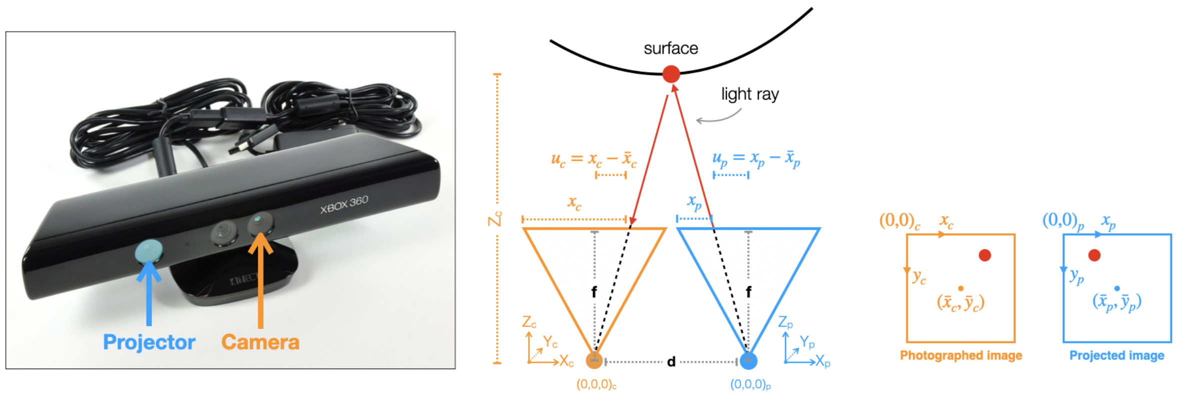


Figure 3: Left: A Kinect depth sensor, which consists of a projector and a camera (note: there is also a second camera used to get better quality color images) [image source: https://www.ifixit.com/Teardown/Microsoft-Kinect-Teardown/4066/1]. Middle: The geometry of the projector-camera system we will consider. Right: The images seen from the perspective of the camera and the projector, when we project a single red laser.

center into the image origin. We define $(u_p, v_p) = (x_p - \bar{x}_p, y_p - \bar{y}_p)$, where $(\bar{x}_p, \bar{y}_p)$ is the principal point (or image center) of the projected image. Similarly, we define $(u_c, v_c) = (x_c - \bar{x}_c, y_c - \bar{y}_c)$, where $(\bar{x}_c, \bar{y}_c)$ is the principal point of the photographed image. We would like to use this pair of image coordinates to determine the 3D coordinates of the surface point causing the reflection.

The camera and projector are offset a distance d from each other *only* in the X direction. The focal length of the camera is f, measured in the same units as xy-coordinates (in the code, we will use pixels as the units). We distinguish between two kinds of 3D coordinates: (X_c, Y_c, Z_c) are coordinates relative to the camera center (i.e. the camera center, which is the orange dot, is at the origin in this reference frame). (X_p, Y_p, Z_p) are coordinates relative to the projector center. We want to find the location of the red dot in the world given the appearance of the red dot on the camera.

(a) **Triangulation** (1 point): We shine a ray of light out of the projector at coordinate (u_p, v_p), then find where the camera observes it, in camera image coordinates (u_c, v_c), in uv-coordinates. These coordinate pairs allow us to triangulate the 3D coordinates of the object that the ray reflected off of (the red dot in Figure 3).

Write expressions for the 3D coordinates of the red dot relative to the camera (X_c, Y_c, and Z_c) 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!

(b) **Simulating a projected laser** (2 points): Open the jupyter notebook provided (we recommend using Google Colab). We want to first 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 3).

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 (u_p, v_p) to (u_c, v_c) given known scene geometry Z_p.

Note: If the projector and camera views differed only by a rotation, we could simply use a homography and would not need to know Z_p to perform this transformation. Because there is a translation involved, the transformation becomes depth-dependent.

We will concatenate the value $1/Z_p$ to our projector coordinates (u_p, v_p). It turns out that the mapping from $(u_p, v_p, 1/Z_p)$ to (u_c, v_c) can be expressed as a fixed linear transformation, and your job will be to find it by composing a series of intermediate mappings.

Note: $(u_p, v_p, 1/Z_p)$ should not be thought of as a coordinate in space.

The notebook contains four missing transformation matrices, operating on homogeneous coordinates. Homogeneous coordinates means that the 3x1 vector $(u_p, v_p, 1/Z_p)$ is equivalent to the 4x1 vector $(u_p, v_p, 1/Z_p, 1)$ where the final element is an arbitrary scale factor. The four mappings are:

- $T_1 : (u_p, v_p, 1/Z_p)$ to (X_p, Y_p, Z_p) $\triangleleft$ projector image to projector world
- $T_2 : (X_p, Y_p, Z_p)$ to (X_c, Y_c, Z_c) $\triangleleft$ projector world to camera world
- $T_3 : (X_c, Y_c, Z_c)$ to (u_c, v_c) $\triangleleft$ camera world to camera image
- $T : (u_p, v_p, 1/Z_p)$ to (u_c, v_c) $\triangleleft$ projector image to camera image

In the notebook, fill in the matrices T_1, T_2, and T_3 and compose them to form matrix T. The ability to do this kind of algebra on transformations is one of the main reasons we like homogeneous coordinates. Copy your code for the four matrices into your PDF writeup (please keep it concise and only include what is relevant).

(c) **Laser scan of a scene** (1 point): We use the transformations you defined to simulate the laser scanning over a room. Why does the path of the laser wiggle in the horizontal direction from the camera's view?

Try changing the scan path (scan left to right rather than top to bottom). Does the laser wiggle in the vertical direction? 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) **Inferring camera depth** (1 point): Now we will try to infer camera depth from the simulated image of the laser scanning over the room. As before, we shine the laser through (u_p, v_p), and observe it in the camera at (u_c, v_c), in uv-coordinates. Use your expression from part (a) to fill in the code for function `inferDepthFromMatchedCoords`, which computes Z_c.

Use the function you wrote to infer the depth of the bicycle scene. Describe what you observe and share a screenshot of the result in your PDF writeup.

(e) **Projecting structured light** (1 point): 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 (x_p, y_p) created the illumination observed by the camera at each point (x_c, y_c), in xy-coordinates. 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. Complete the function `getImgCoordinatePairs` to generate these corresponding image coordinates.

We have provided a pattern of stripes `stripe_lights.png`. Illuminate the scene with this pattern of light. To do this, we have provided a function `util.render`, which requires lists of corresponding projector and camera image coordinates, and the intensity of light to project at each of these pixels (represented by `L_p_img`). Show your result in your PDF writeup.

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

Note 1: `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.*

Note 2: `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.*

(f) **Inferring depth from structured light** (2 points): 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

$(x_p, y_p) = F(L_c((x_c, y_c)), (x_c, y_c))$,

where $L_c((x_c, y_c))$ is the intensity of light the camera sees at pixel (x_c, y_c), and this pixel was illuminated by the ray that was projected through (x_p, y_p). F can be a trivial function if you choose a certain pattern. Write code for F, and write code that generates the light pattern (an image) in `getStructuredLight`.

Use 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)! Copy your code for F and an image of the decoded depth into the PDF writeup.

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

(g) *6.869 only* – **Lambertian rendering** (2 points): 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 `util.render_Lambertian`. This renderer uses a Lambertian reflectance model:

$$I_{out} = N \cdot L * I_{in}$$

where I_{out} 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, I_{in} is the intensity of the incoming light ray from the projector, 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 pattern and decoder from (f) still work? Why or why not?

Design a new light pattern and new decoder F that works better in the presence of Lambertian reflectance. Hint: try illuminating the scene with colored light. Copy your code for F and an image of the decoded depth into the PDF writeup.