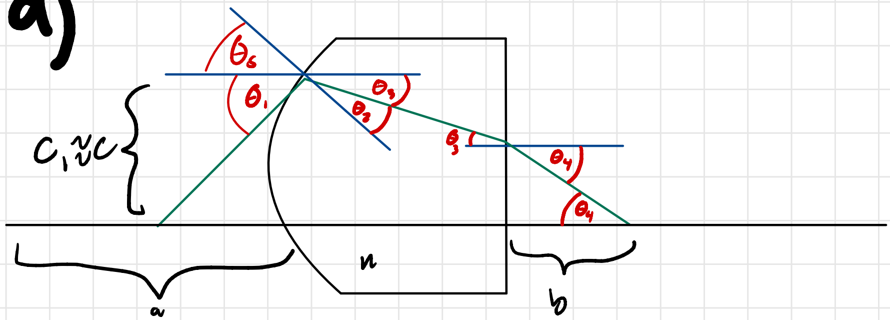
① a)



1) $\frac{c}{a} = \tan(\theta_1) \approx \theta_1 \rightarrow \boxed{\theta_1 = \frac{c}{a}}$

2) $n_1(\theta_1 + \theta_5) = n_2(\theta_2) \rightarrow \boxed{\theta_1 + \theta_5 = n\theta_2}$

3) $\boxed{\theta_5 = \theta_2 + \theta_3}$

4) $\boxed{n\theta_3 = \theta_4}$

5) $\boxed{\theta_4 = \frac{c}{b}}$

b)

$$\frac{1}{a} + \frac{1}{b} = \frac{1}{f}$$

$$\frac{1}{\frac{c}{\theta_1}} + \frac{1}{\frac{c}{\theta_4}} = \frac{\theta_1}{c} + \frac{\theta_4}{c} = f$$

$$\frac{1}{c}\left(n\theta_2 - \theta_5 + n\theta_3\right)$$

$$= \frac{1}{c}\left(n(\theta_2+\theta_3) - \theta_5\right)$$

$$= \frac{1}{c}\left(n\theta_5 - \theta_5\right)$$

$$= \frac{\theta_5}{c}(n-1)$$

$$\boxed{\frac{1}{f} = \frac{1}{R}(n-1)}$$

where $R = \frac{c}{\theta_5}$

2 a) Using similar triangles

$$\frac{u_p}{f} = \frac{X_p}{Z}, \quad \frac{u_c}{f} = \frac{X_c}{Z}$$

From the previous equalities

& $x_c - x_p = d$

$$\frac{u_c z}{f} - \frac{u_p z}{f} = d$$

$$\boxed{z_p = \frac{df}{u_c - u_p}}$$

Then,

$$\frac{f}{u_c} = \frac{\frac{df}{u_c - u_p}}{x_c}$$

$$\boxed{x_c = \frac{d u_c}{u_c - u_p}}$$

Finally,

$$\frac{f}{v_c} = \frac{\frac{df}{u_c - u_p}}{y_c} \rightarrow \boxed{y_c = \frac{d v_c}{u_c - u_p}}$$

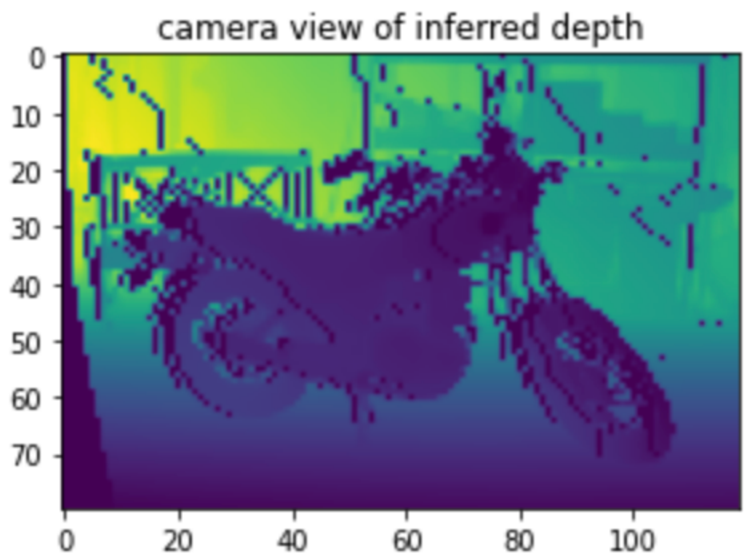
2b)

```python
T1 = np.array([[1/f,0,0,0],
               [0,1/f,0,0],
               [0,0,0,1],
               [0,0,1,0]])

"""
map (X_p,Y_p,Z_p) to (X_c,Y_c,Z_c)

| ? ? ? ? | | X_p |   | X_c |
| ? ? ? ? |*| Y_p | = | Y_c |
| ? ? ? ? | | Z_p |   | Z_c |
| ? ? ? ? | |  1  |   |  1  |
"""

T2 = np.array([[1,0,0,d],
               [0,1,0,0],
               [0,0,1,0],
               [0,0,0,1]])

"""
 map (X_c,Y_c,Z_c) to (u_c,v_c)

| ? ? ? ? | | X |   | au |                                |u|
| ? ? ? ? |*| Y | = | av |, which represents same point as |v|
| ? ? ? ? | | Z |   |  a |                                |1|
            | 1 |
"""
T3 = np.array([[f,0,0,0],
               [0,f,0,0],
               [0,0,1,0]])


# map (u_p,v_p,1/Z_p) to (u_c,v_c)
T = np.dot(T3,np.dot(T2,T1))
```

2c) The laser wiggles in the horizontal direction because of the shifting of perspectives. This phenomenon is demonstrated by placing a finger in front of your face and then closing one eye and then the other. When moving your finger up and down you will replicate this diagonal oscillating motion. However, when you scan horizontally, you won't notice the skipping caused by the change in perspective because it is parallel to the direction of scanning. It appears like a hopping back and forth but "the wiggle" is always in the horizontal direction.

```
for y_p in range(1, height, 4):
     for x_p in range(1, width, 20):
```

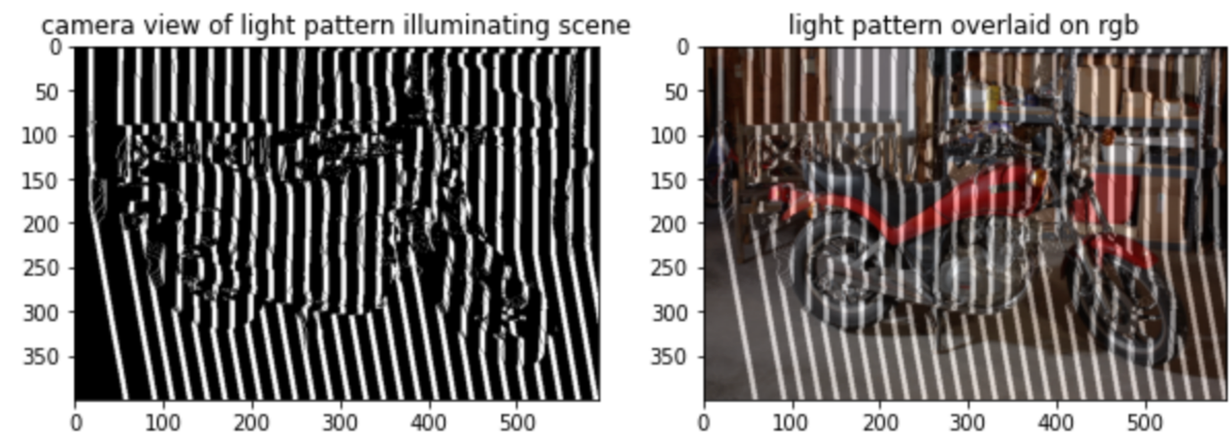
2d) Clearly, the computer recognizes different depths of the picture. Images nearer are dark purple and as the depth increases you slowly get green and then yellow tones. However, the computer does not recognize miniscule differences in depth. For example, most piece of the motorcycle's body appear to be the same depth so there is room to improve.



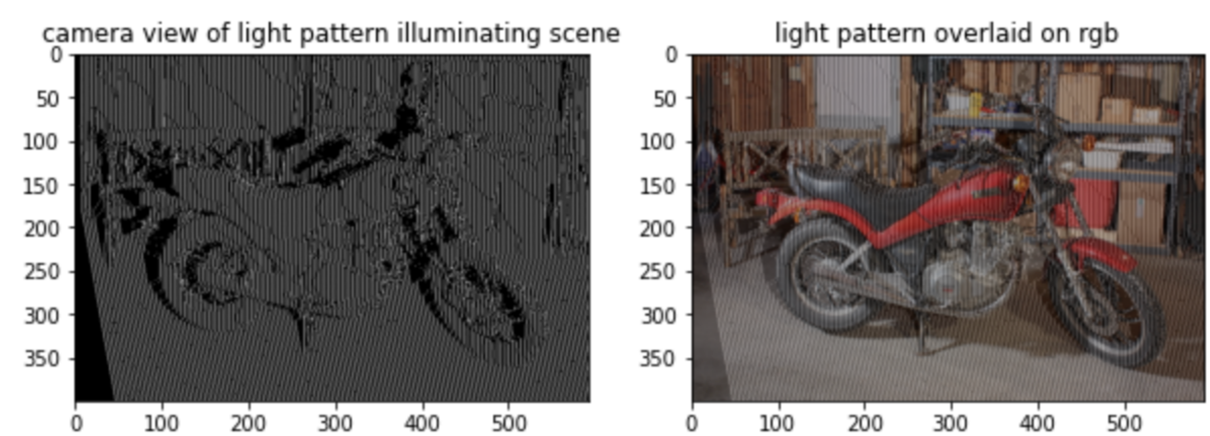

```
Z_c = (f*d)/(np.subtract(uv_c[:,0],uv_p[:,0]))
```

2e) In the first set of images, notice how the motorcycle occluding the ground blocks the projected light. This is also very apparent in the handlebars.

```
    x_p_img, y_p_img = np.meshgrid(np.arange(img_size[1]),
np.arange(img_size[0]))
    xy_p = np.hstack([flatten(x_p_img), flatten(y_p_img)])
    Z = np.array([[Z_p_img[xy[1], xy[0]]] for xy in xy_p])
    xy_c = transform_xy_p_to_xy_c(xy_p, Z, cx_p, cy_p, cx_c, cy_c, T)
```

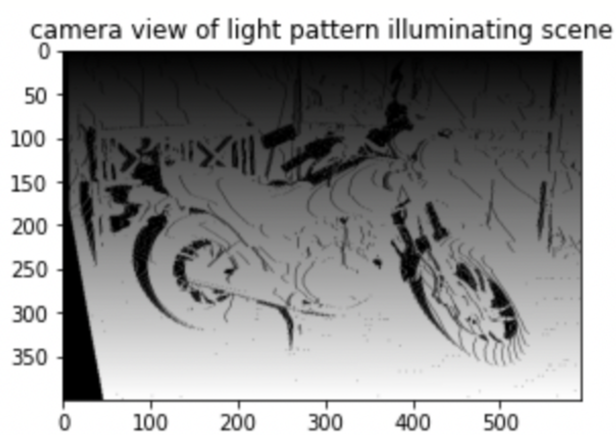


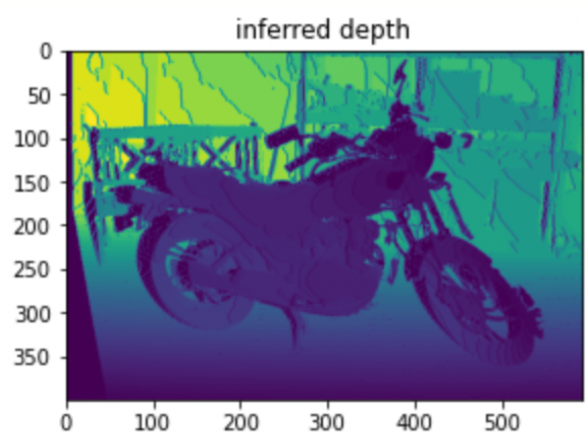

```
L_p_img.shape
L_p_img = np.zeros((400,593),dtype=int)
L_p_img[1::2,::4] = 255
L_p_img[::2,1::4] = 255
```



2f) The structure light pattern I used assigned intensity to each entry in the matrix like reading a book: row-wise left to right. Doing the same think right to left would work or even a snaking pattern column-wise. Any systematic way that is easy to obtain the row and column number could be easily implemented.

```
def getStructuredLight(img_size):
    L_p_img = np.zeros([img_size[0],img_size[1]])
    i = 0
    for x in range(0,img_size[0]):
      for y in range(0,img_size[1]):
          L_p_img[x,y] = i
          i+=1

    return L_p_img
def F(L_c, xy_c):
    # L_c is Nx1
    # xy_c is Nx2
    # xy_p should be Nx2

      x_p = L_c - ((L_c // img_size[1]) * img_size[1])
      y_p = L_c // img_size[1]
      xy_p = np.hstack([x_p, y_p])

      return xy_p
```





2g) If we render the image in a more realistic way, my pattern and decoder would not work. Since different types of surfaces at different angles will change the intensity, one channel is no longer sufficient. We utilize the red, green, and blue channel in order to uniquely identify a single pixel by inferring the change of intensity.

```
def getStructuredLight_Lambertian(img_size):
    L_p_img = np.zeros([img_size[0],img_size[1],3])

    for x in range(0,img_size[0]):
      for y in range(0,img_size[1]):
          L_p_img[x,y,0] = y

    for x in range(0,img_size[0]):
      for y in range(0,img_size[1]):
          L_p_img[x,y,1] = x

    for x in range(0,img_size[0]):
      for y in range(0,img_size[1]):
          L_p_img[x,y,2] = 1
    return L_p_img
def F_Lambertian(L_c, xy_c):
    L_c_array = np.array(L_c)
    scale = L_c_array[:,2]
    Red = L_c_array[:,0]/scale
    Green = L_c_array[:,1]/scale
    xy_p = np.dstack((Red,Green))[0]

    return xy_p
```