Optical Flow Programming Assignment
Kareem Tiwari
CSC340

For the Optical Flow programming assignment, the goal of the assignment was to take the pairs of images that were given and to find the flow between the small amount of motion that exists between each of the image pairs. Once that was completed we were then tasked with trying to create a visualization of the flow color as well as using arrows placed on the image to show the flow between each pair of images.

First, I created all of the empty images that I needed as well as loading both of the images in the image pair using OpenCV as well as creating other variables that are used later on in the program. This is shown directly below.

```
# create

####
import cv2
import numpy as np
import math

im = cv2.imread("tree1.jpg",0) # Grayscale image
im_2 = cv2.imread("tree2.jpg",0) # image 2
numRows = im.shape[0] # x value
numCols  = im.shape[1] # y value
emptyim_Ix = np.zeros((numRows,numCols,1),np.float32)#create  an empty image
emptyim_IxIx = np.zeros((numRows,numCols,1),np.float32)#create  an empty image
emptyim_Iy = np.zeros((numRows,numCols,1),np.float32)#create  an empty image
emptyim_IyIy = np.zeros((numRows,numCols,1),np.float32)#create  an empty image
emptyim_IxIy = np.zeros((numRows,numCols,1),np.float32)#create  an empty image
emptyim_It = np.zeros((numRows,numCols,1),np.float32)
emptyim_IxIt = np.zeros((numRows,numCols,1),np.float32)
emptyim_IyIt = np.zeros((numRows,numCols,1),np.float32)
cornerness_im =  np.zeros((numRows,numCols,1),np.float32)
emptyim_u =  np.zeros((numRows,numCols,1),np.float32)
emptyim_v =  np.zeros((numRows,numCols,1),np.float32)
emptyim_mag =  np.zeros((numRows,numCols,1),np.float32)
emptyim_arrow = np.zeros((numRows,numCols,3),np.float32)
emptyim_rgb = np.zeros((numRows,numCols,3),np.float32)
width = numRows/5
height =  numCols/5
count = 0
```

Second, I got and stored all of the gradient values from every pixel in every image that was created using a for loop. Shown below.

```
# a = i and b = j( when looking at the Harris Corner coding notes on gradient
PixelGrad_Array = [] # create an array to store the gradient values from every p
for i in range(1,numRows-1):
    for j in range(1,numCols-1):
        emptyim_Iy[i,j] = int(im[i+1,j]) - int(im[i-1,j]) # gradient x

        emptyim_Ix[i,j] = int(im[i,j+1]) - int(im[i,j-1]) #gradient y

        emptyim_It[i,j] = int(im_2[i,j]) - int(im[i,j]) # t value / differnece i

        emptyim_IxIx[i,j]= emptyim_Ix[i,j] *  emptyim_Ix[i,j]

        emptyim_IyIy[i,j] = emptyim_Iy[i,j] * emptyim_Iy[i,j]

        emptyim_IxIy[i,j] = emptyim_Ix[i,j] * emptyim_Iy[i,j]

        emptyim_IxIt[i,j] = emptyim_Ix[i,j] * emptyim_It[i,j]

        emptyim_IyIt[i,j] = emptyim_Iy[i,j] * emptyim_It[i,j]
```

Third, I iterate through the pixels of the image using the block size that is deemed correct. While in this loop I am setting the sums of the gradience values from every pixel in the block size that I had set. As well as calculating the determinant and Trace values in the same loop for every pixel.

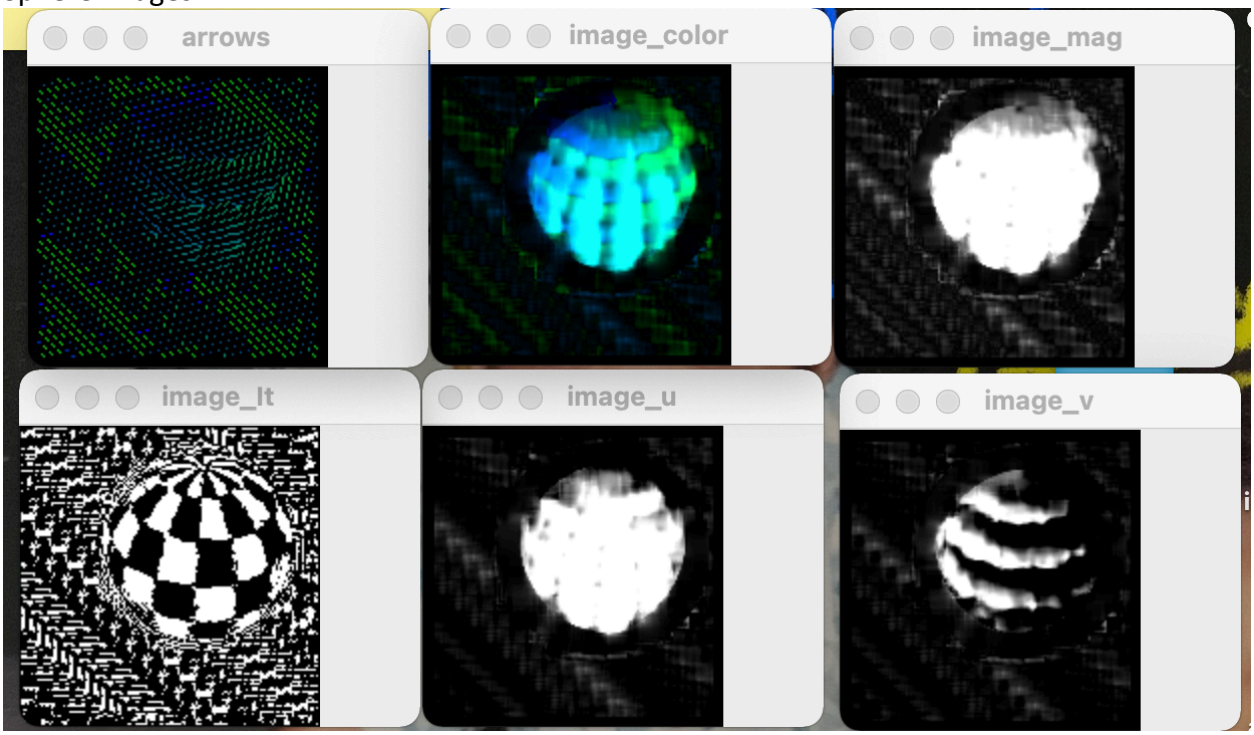
```
for  i  in range(6,numRows-5):
    for j in range(6,numCols-5):
        Sum_IxIx = 0
        Sum_IxIy = 0
        Sum_IyIy = 0
        Sum_IyIt = 0
        Sum_IxIt = 0
        for a in range(-5,5):
            for b in range(-5,5):
                Sum_IxIx += emptyim_IxIx[i+a,j+b]
                Sum_IxIy += emptyim_IxIy[i+a,j+b]
                Sum_IyIy += emptyim_IyIy[i+a,j+b]
                Sum_IyIt += emptyim_IyIt[i+a,j+b]
                Sum_IxIt += emptyim_IxIt[i+a,j+b]

##        Sum_IxIx = emptyim_IxIx[i,j]+emptyim_IxIx[i+1,j] +emptyim_IxIx[i-1,j] + emptyim_IxIx[i,j-1] +emptyim_IxIx[i,j+1] +e
##        Sum_IxIy = emptyim_IxIy[i,j]+emptyim_IxIy[i+1,j] +emptyim_IxIy[i-1,j] + emptyim_IxIy[i,j-1] +emptyim_IxIy[i,j+1] +e
##        Sum_IyIy = emptyim_IyIy[i,j]+emptyim_IyIy[i+1,j] +emptyim_IyIy[i-1,j] + emptyim_IyIy[i,j-1] +emptyim_IyIy[i,j+1] +e
##        Sum_IyIt = emptyim_IyIt[i,j]+emptyim_IyIt[i+1,j] +emptyim_IyIt[i-1,j] + emptyim_IyIt[i,j-1] +emptyim_IyIt[i,j+1] +e
##        Sum_IxIt = emptyim_IxIt[i,j]+emptyim_IxIt[i+1,j] +emptyim_IxIt[i-1,j] + emptyim_IxIt[i,j-1] +emptyim_IxIt[i,j+1] +e


##        Sum_IxIx = emptyim_IxIx[i+a,j+b]



        M_det = (Sum_IxIx*Sum_IyIy) -(Sum_IxIy *Sum_IxIy) #This gets the determinant value for every pixel  in the  image
        M_trace = (Sum_IxIx +Sum_IyIy) #This gets the trace value for every pixel in the image
```

Then, I make sure for my flow values u and v that they are not dividing by Zero. Then I set my u and v flow values to the empty images at point I and j as well as getting and storing the Magnitude, Theta, Green color value, and Blue color value for every pixel in the image. I also set the blue and green color values to the “rgb” images that are visualized. Then I set up a counter to make sure that when visualizing the “arrowedLine” that there would be enough room to be able to see the arrows clearly.

I Finally was able to print out all versions of the images visualizing the flow between the image pairs using the cv2.imshow method. This is all shown in the code segment below.

```
        if M_det != 0:
            u = ((-Sum_IyIy*Sum_IxIt)+(Sum_IxIy *Sum_IyIt))/(M_det)
            v = ((Sum_IxIy*Sum_IxIt)-(Sum_IxIx*Sum_IyIt))/(M_det)

            emptyim_u[i,j] = u
            emptyim_v[i,j] = v



##        np.seterr(invalid='ignore')


##        Visualization
            mag = math.sqrt((u**2)+(v**2))
            theta  = math.atan2(v,u)
            b_cv = (255*((theta+math.pi)/(2*math.pi)))
            g_CV = (255-b_cv)

            emptyim_mag[i,j] = mag



            emptyim_rgb[i,j,0] =  b_cv *mag
            emptyim_rgb[i,j,1] = g_CV *mag

            count+= 1
            if count == 35:

                cv2.arrowedLine(emptyim_arrow , (int(j), int(i)), (int(j+u), int(i+v)), (b_cv, g_CV, 0))
                count = 0




cv2.imshow("image_It",emptyim_It)
cv2.imshow("image_u",emptyim_u) #  this gives me the flow y image
cv2.imshow("image_v",emptyim_v)# this gives me the flow x image
cv2.imshow("image_mag",emptyim_mag)
cv2.imshow("image_color",emptyim_rgb/255)
##cv2.imshow("image_RGB",emptyim_rgb)

##cv2.arrowedLine(emptyim_arrow , (int(j), int(i)), (int(j+u), int(i+v)), (b_cv,
##g_CV,0))
cv2.imshow("arrows",emptyim_arrow/255)
```

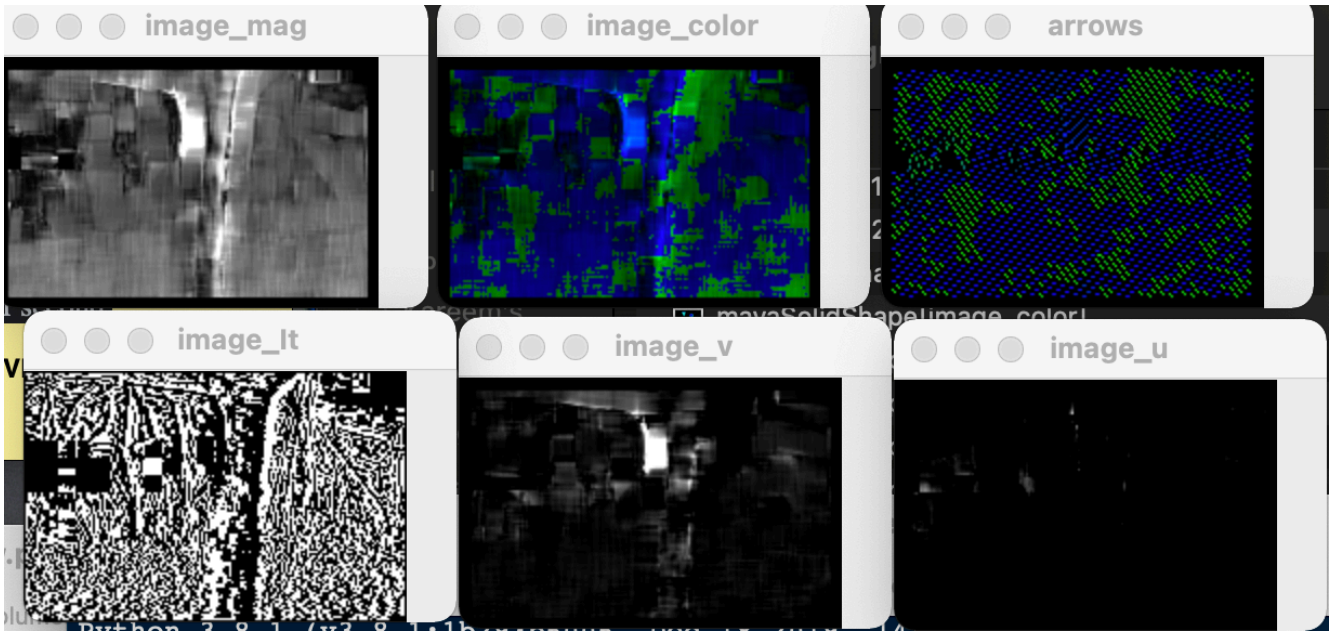
Sphere images:



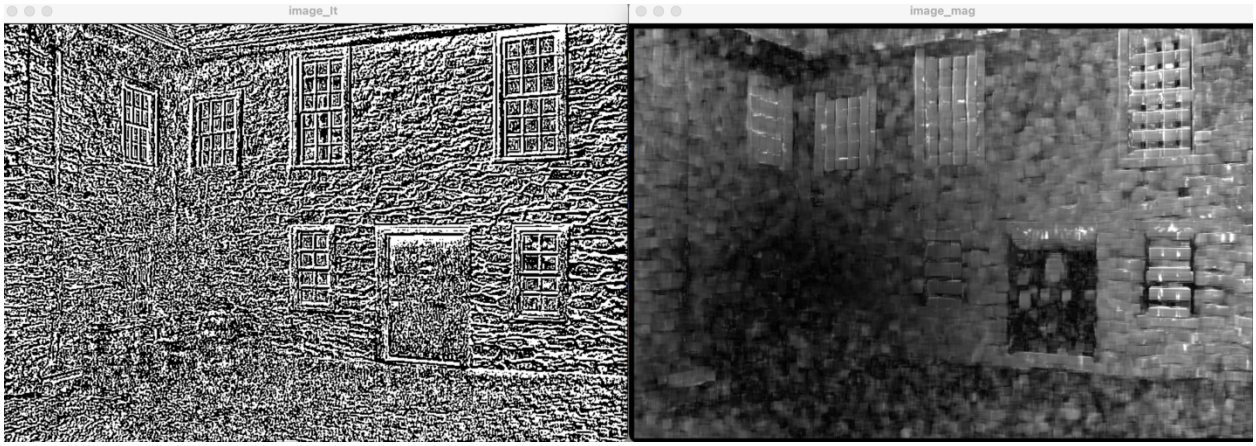
The problems that I encountered while trying to do these flow calculations and visualizations are below:

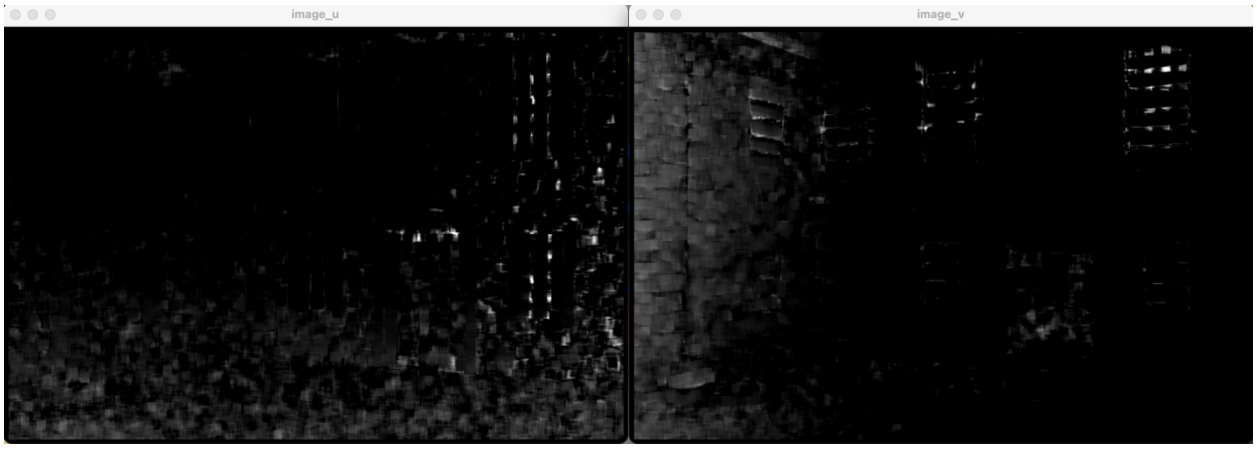
1.) For the image color visualization my image was coming out only as a black image and then only as a greyscale image but the problem was that I was not storing the color values that I was supposed to be storing.
2.) For the arrows image I was getting only a teal image and so I had to implement a loop to only give an arrow image when for every certain number of comparisons as to not overwhelm the image with lines.
3.) Expanding on the arrow image going forward That would be the only number that I changed would be the number of lines before the arrow would be place on the image as to make sure the readability of the image is clear.

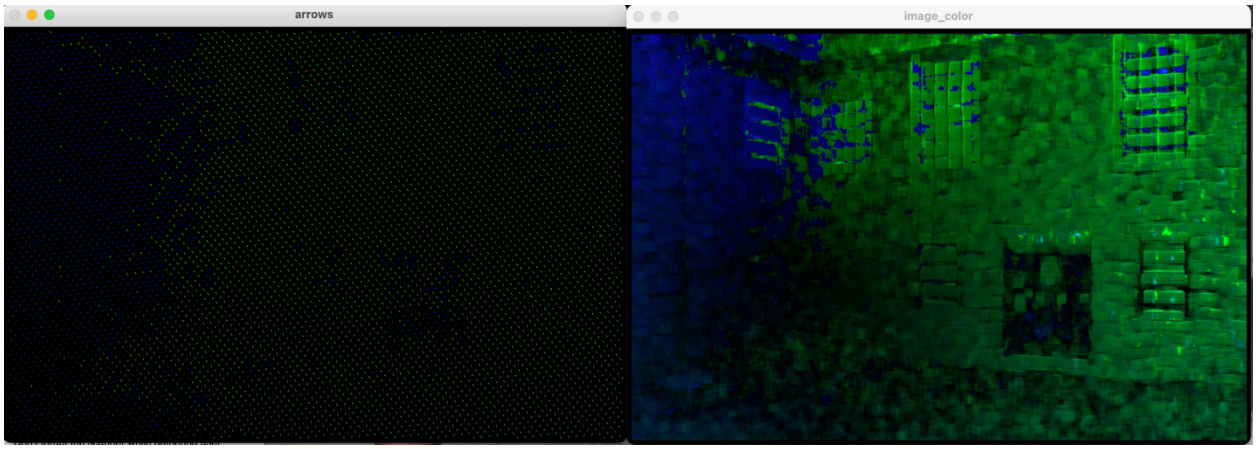
Tree Images:



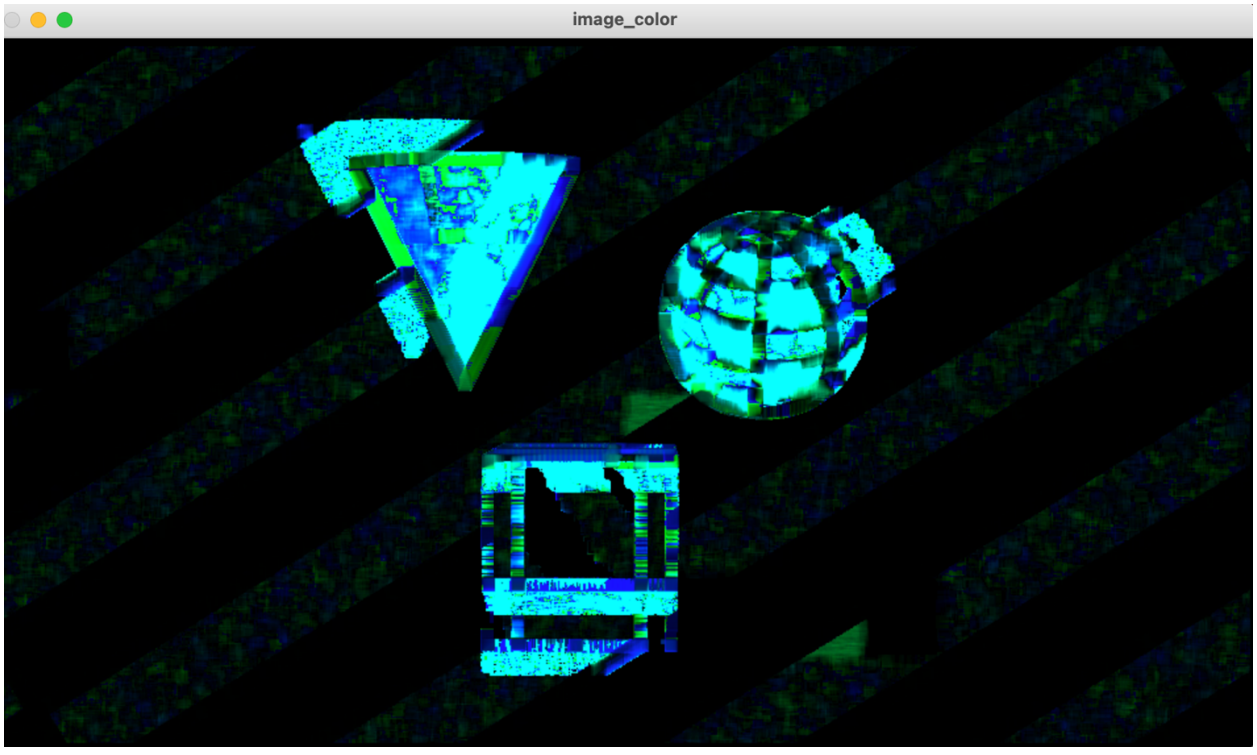
House Images:





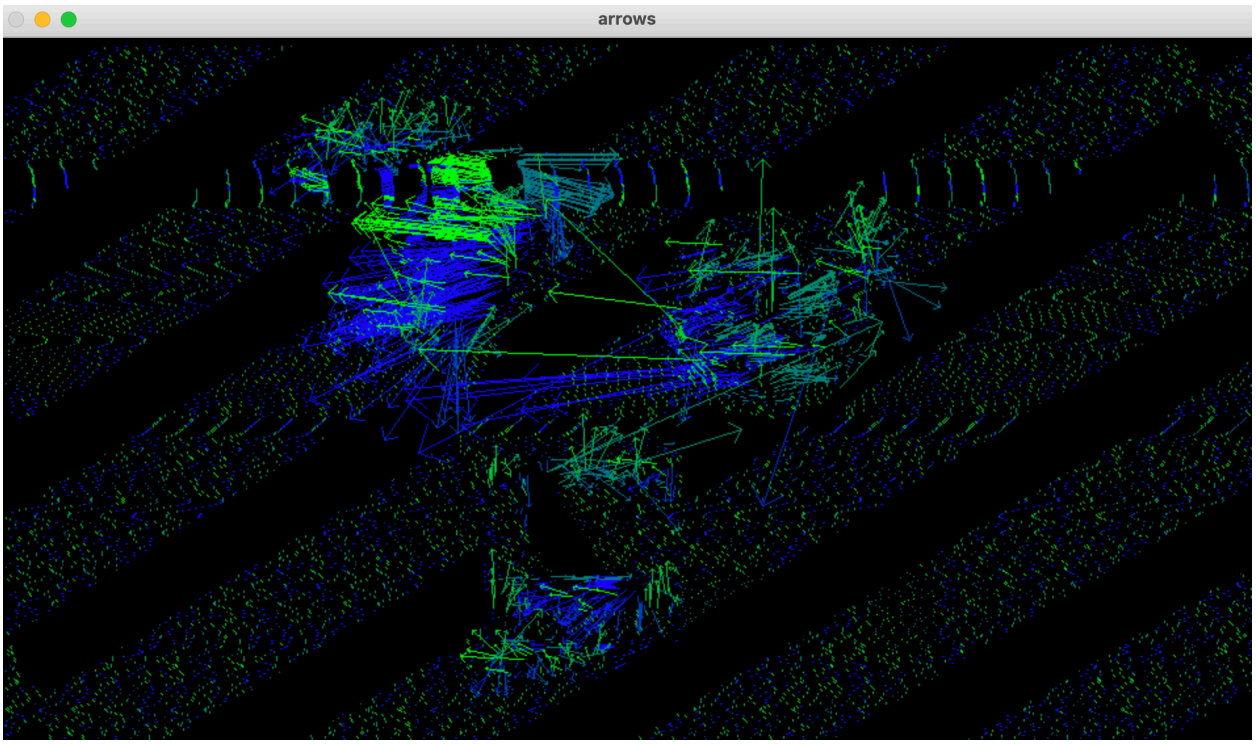


Striped Shapes images:

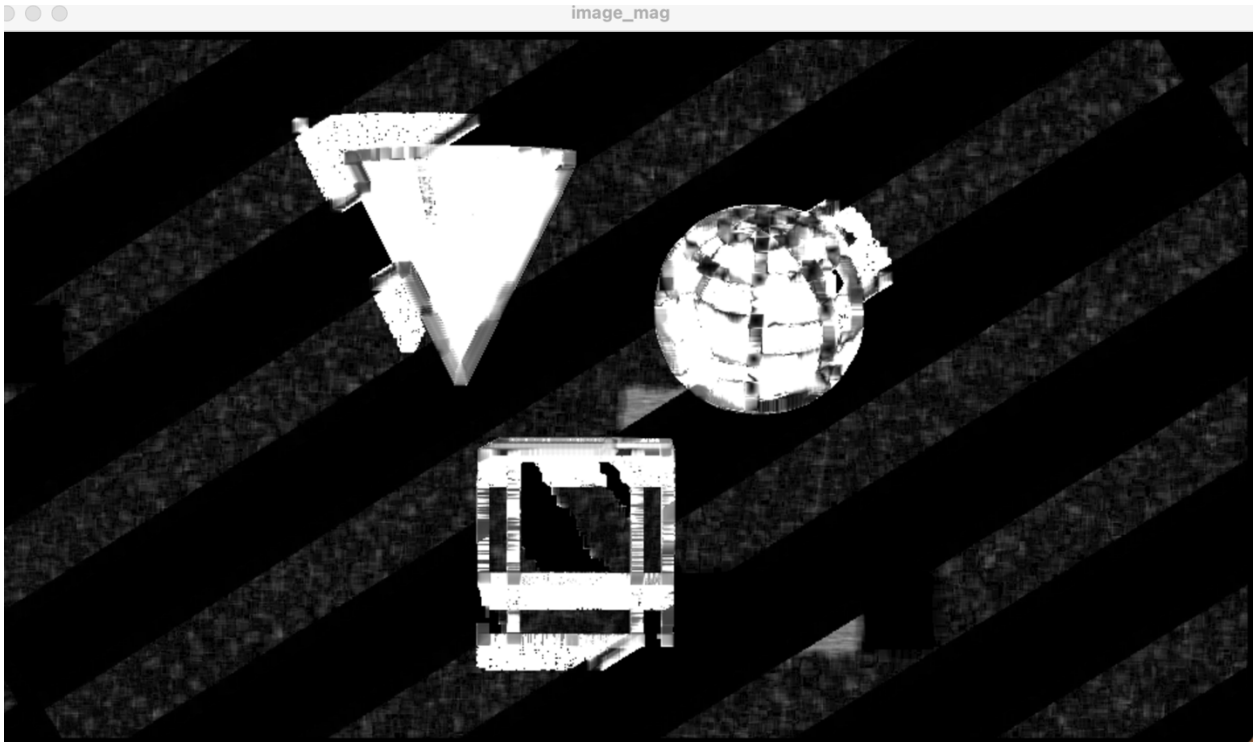


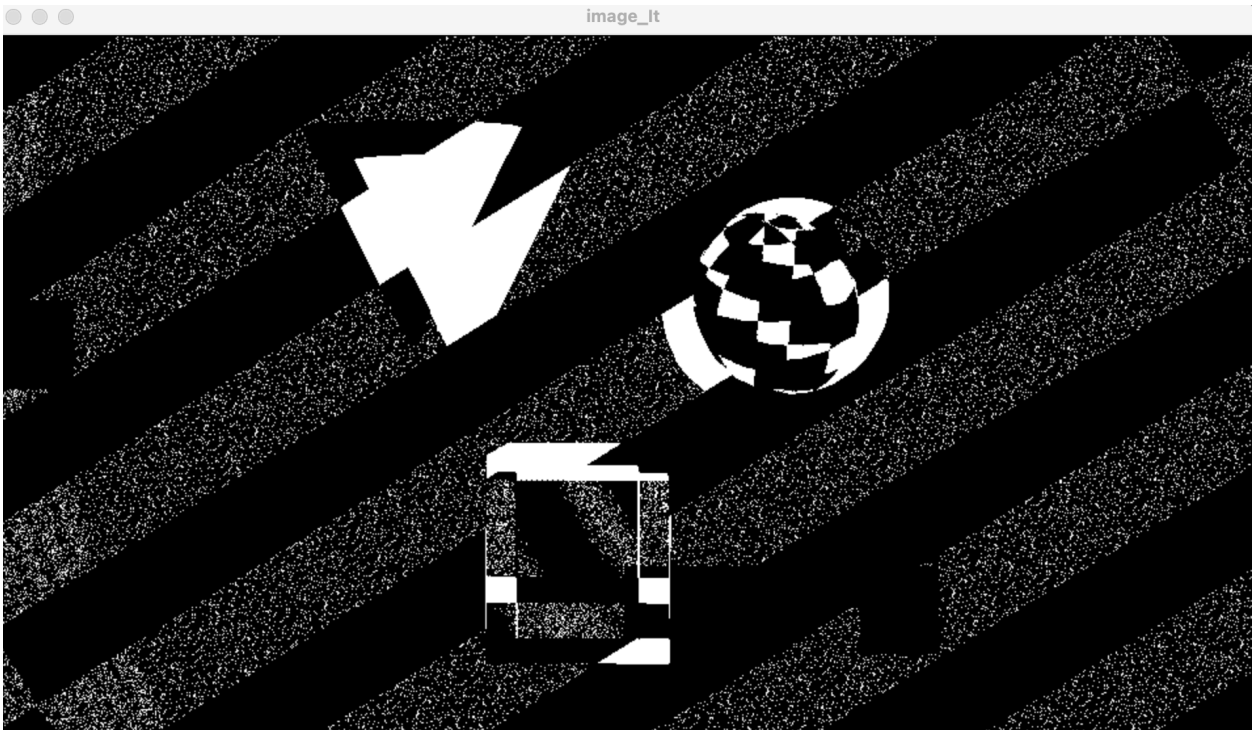

image_v


image_u

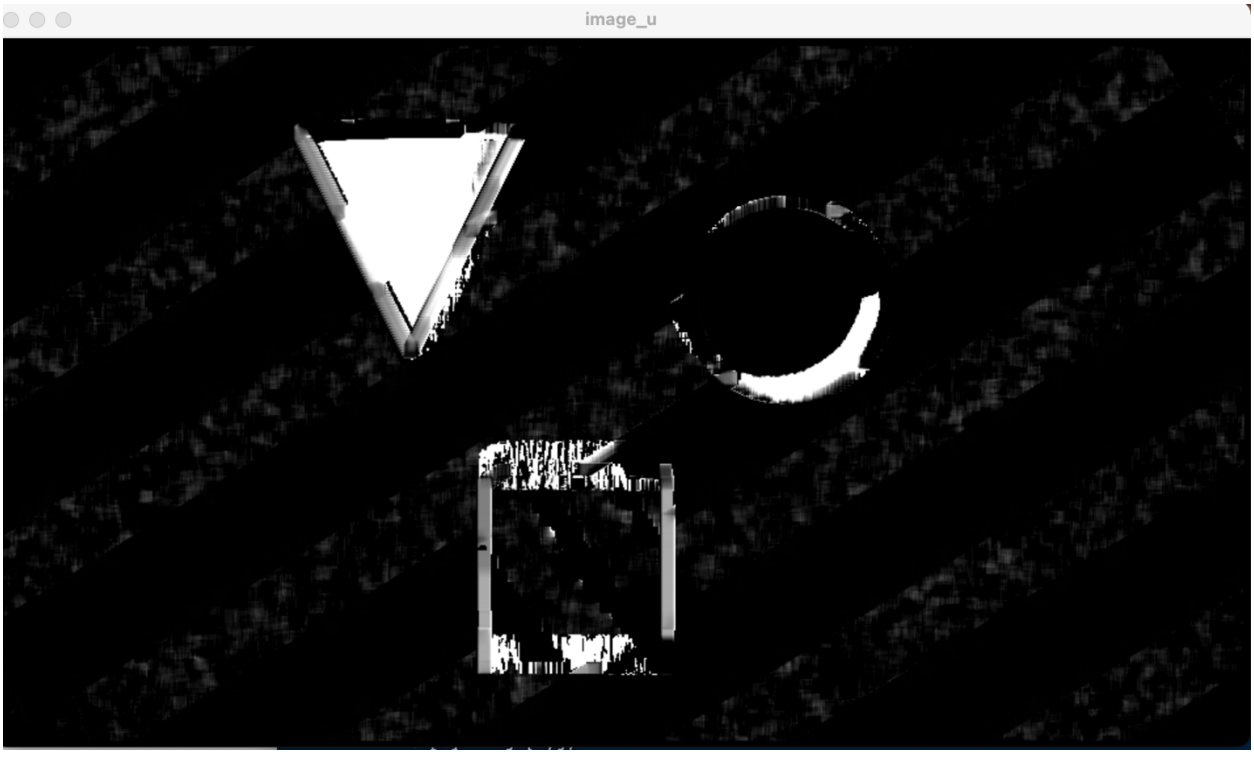


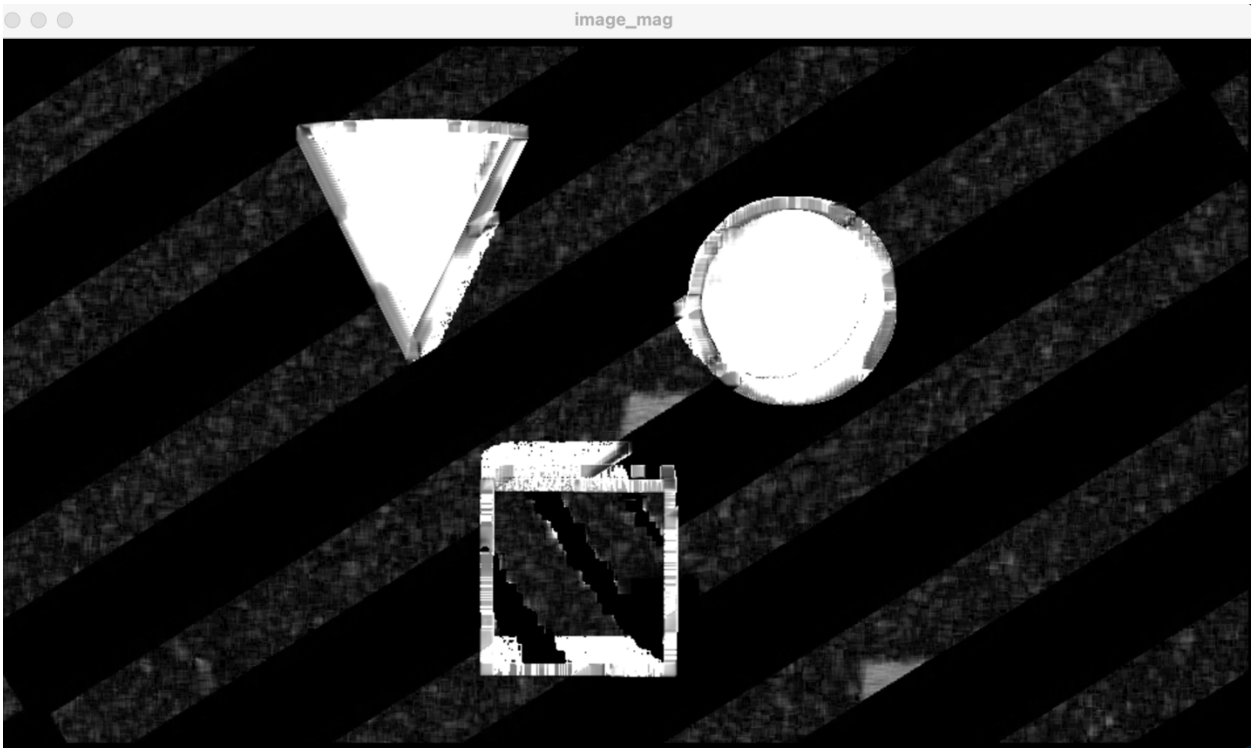
This arrow visualization had the arrows pointing in every which way no matter how large or small the interval for the arrows I chose was.

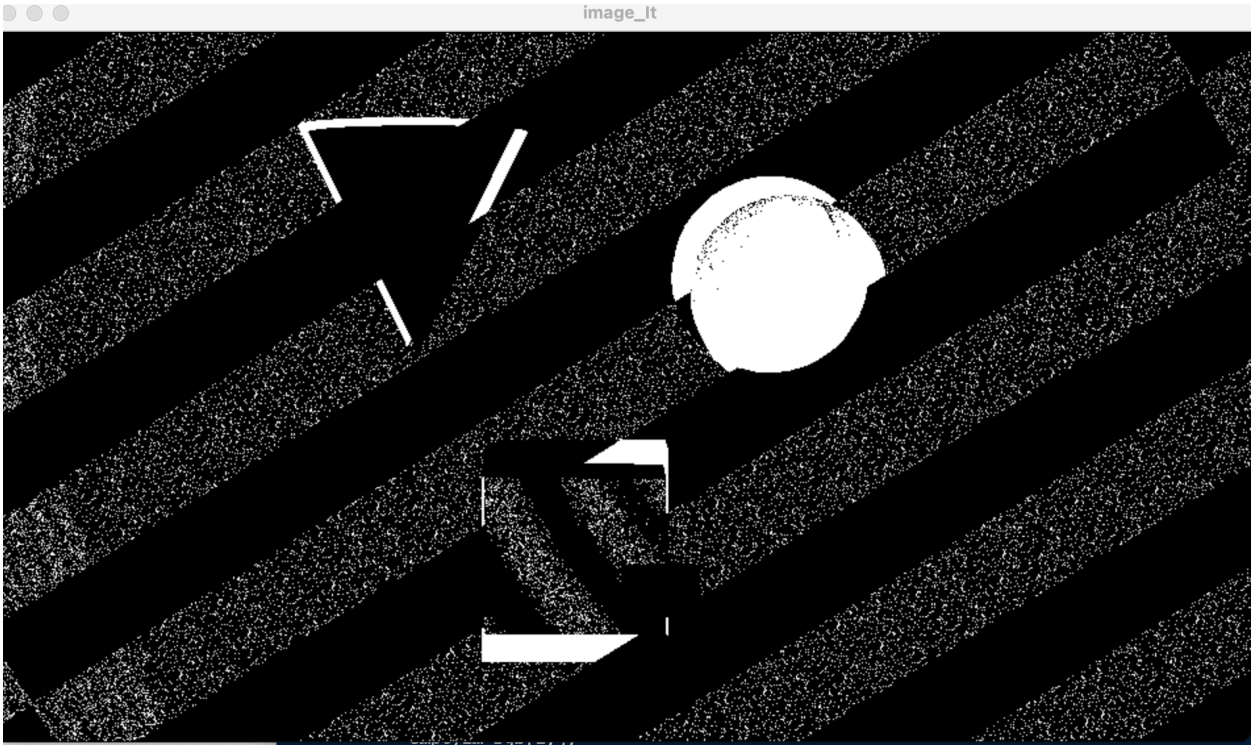

image_mag

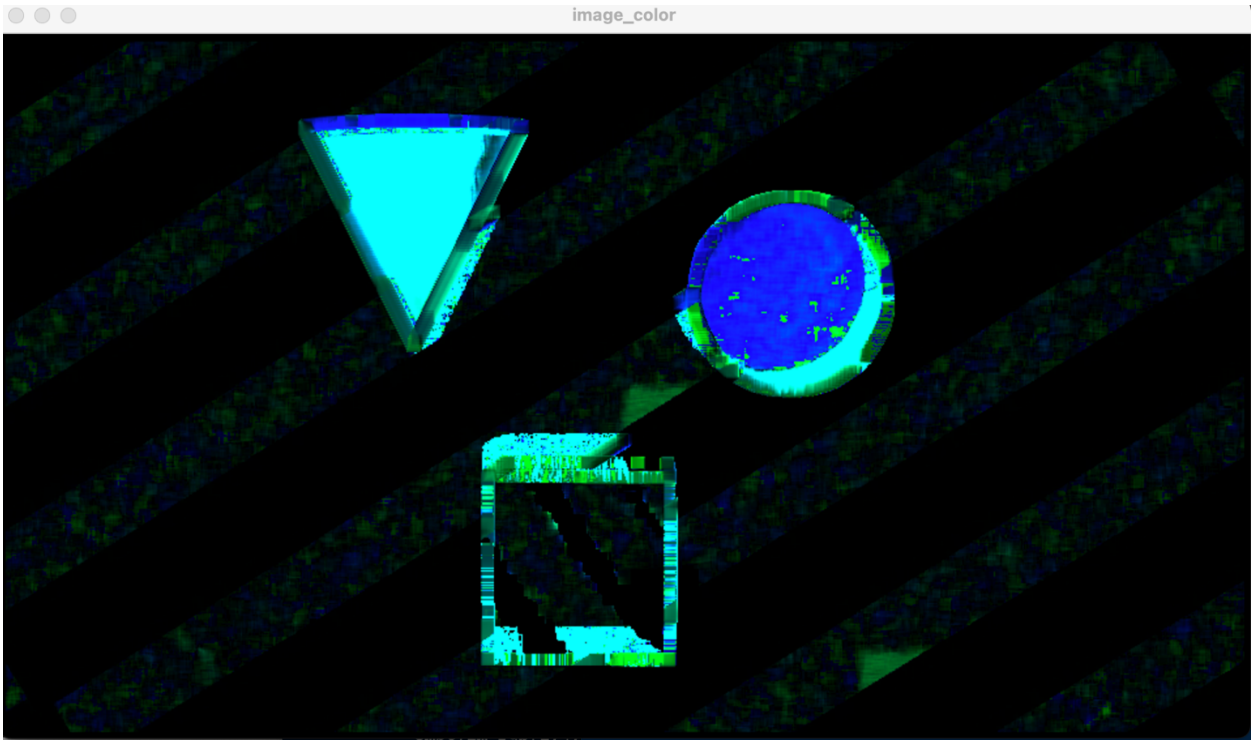

image_lt

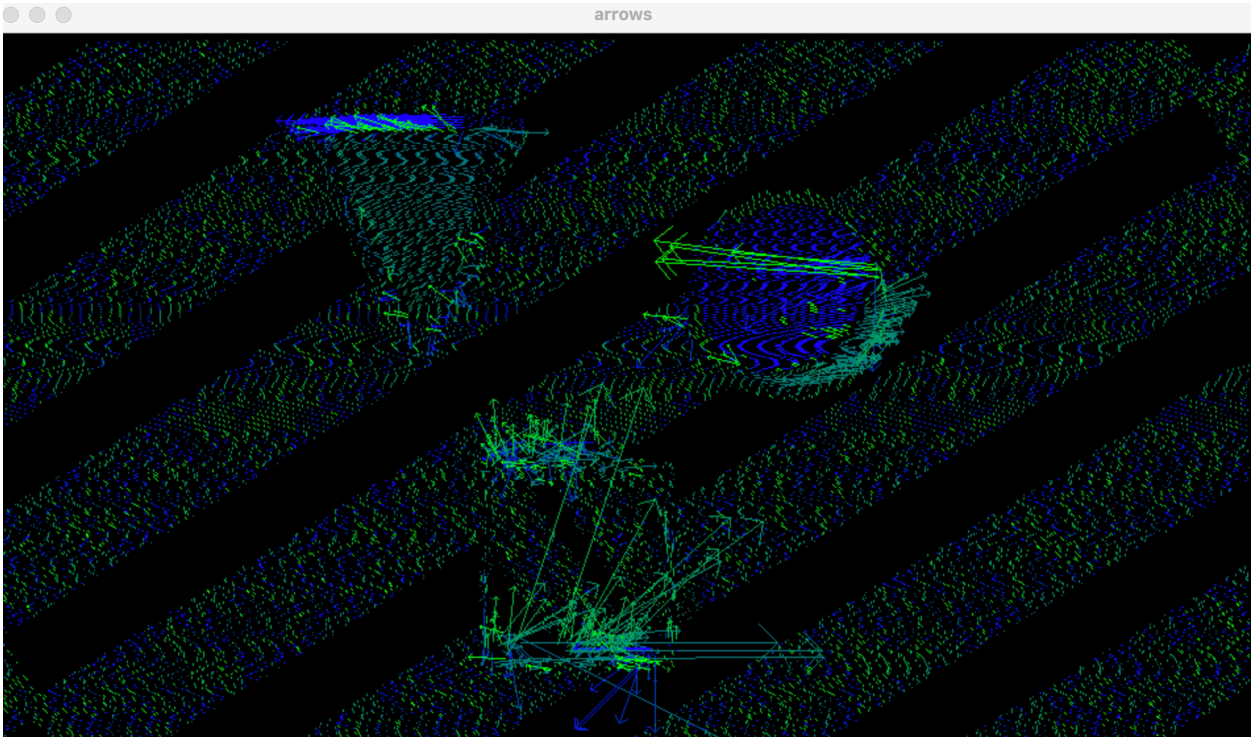
Solid Shape Images:




image_mag


image_lt





For this arrow image also, the arrows as shown above are moving all over the place showing some movement but in odd directions. This also was a case where the

placement of arrows no matter how large or small the interval the movement or flow was not shown quite as well as I would’ve hoped.