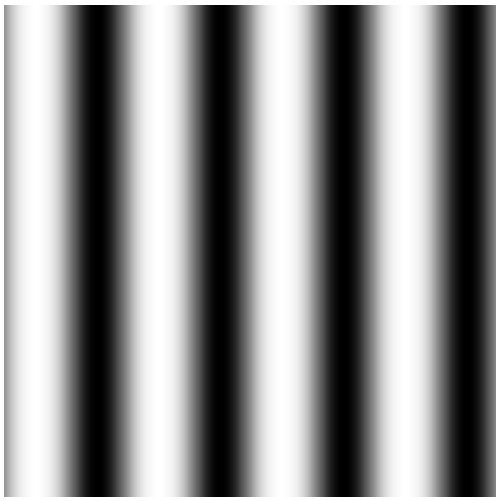
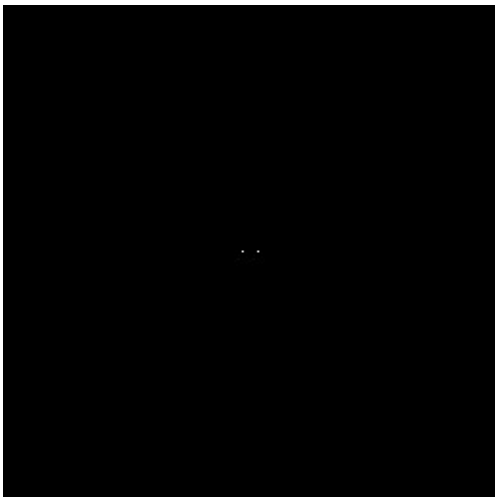
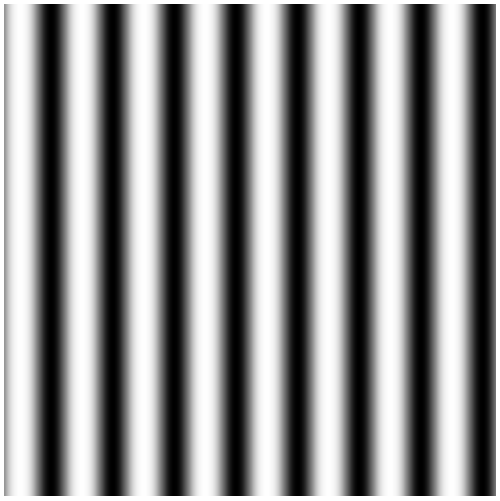
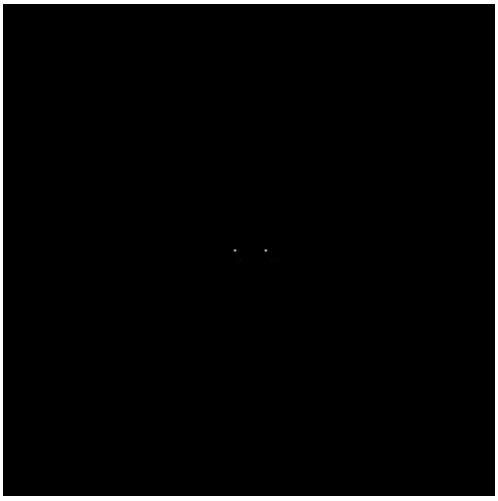
Michael Christensen
November 4, 2013
CS 450

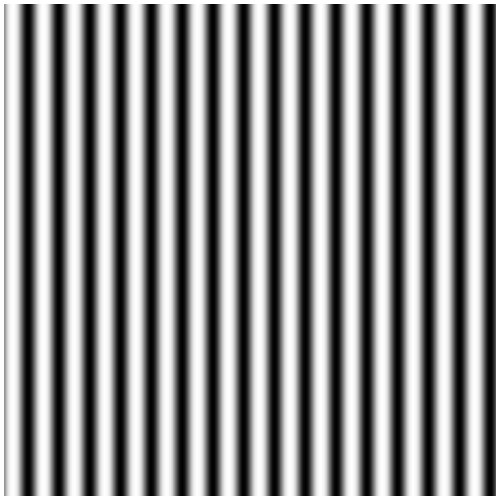
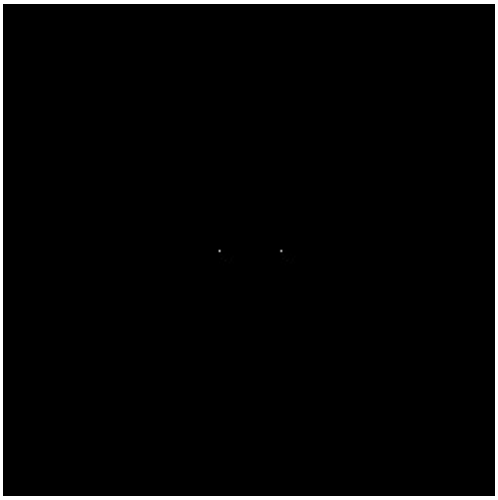
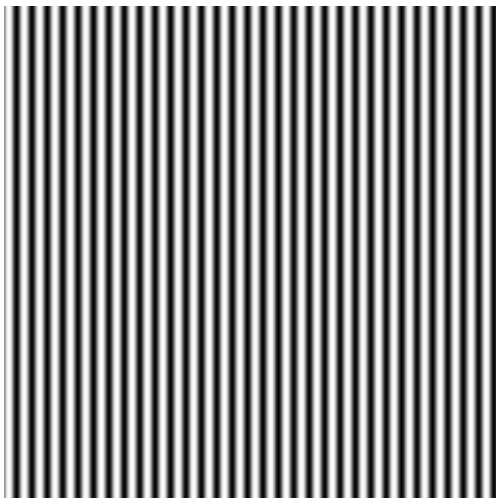
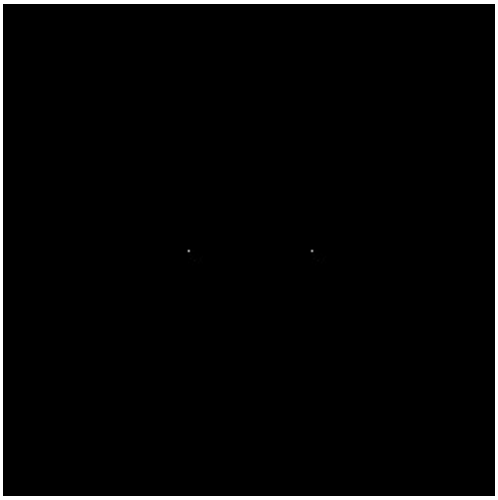
Homework #5

Programming Exercises:

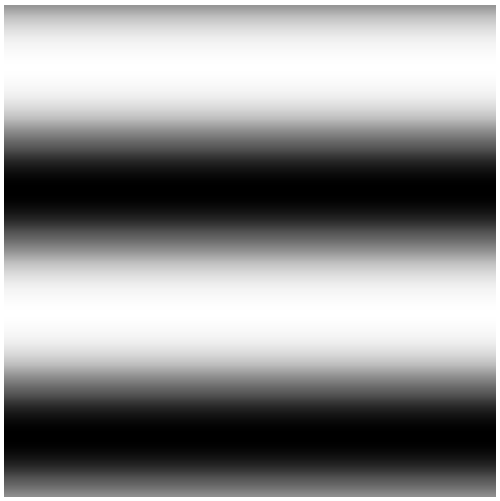
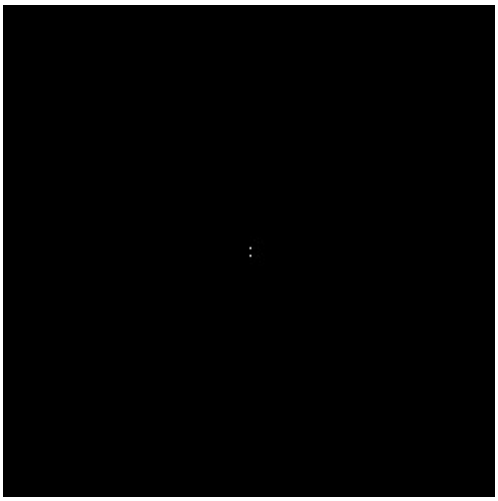
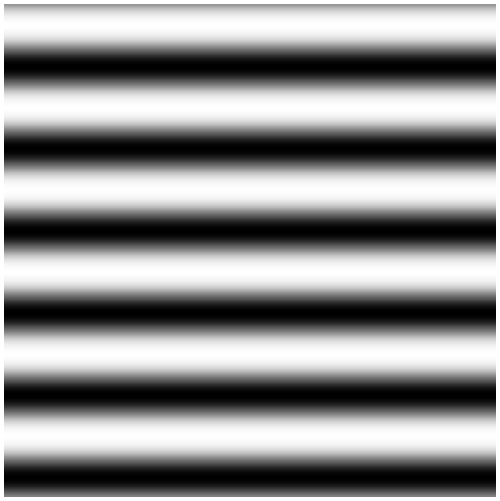
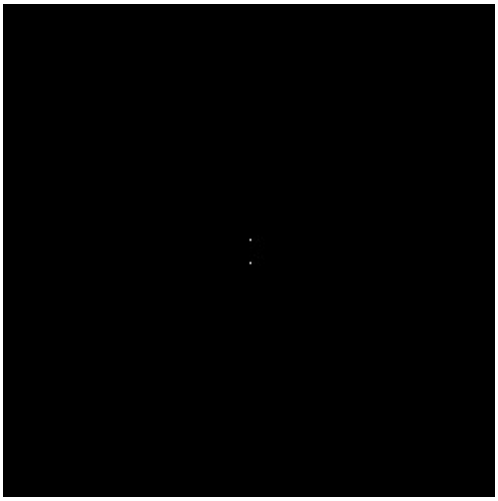
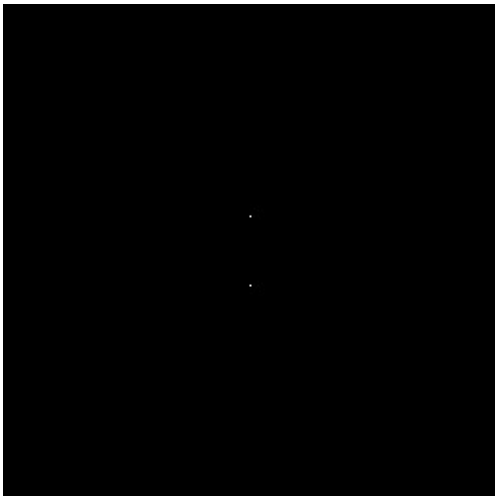
Part A: Simple Sines and Cosines

1. Image that is a sinusoid in one direction and constant in the other (f[x,y] = sin(2 pi s x / N) for various s values, and their magnitudes:

S = 4: 	Magnitude: 
S = 8: 	Magnitude: 

S = 16: 	
S = 32: 	

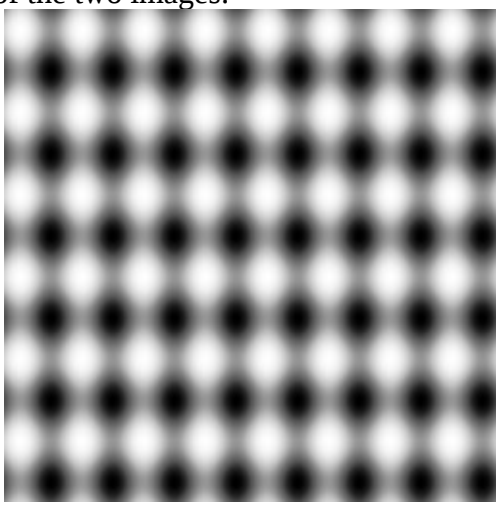
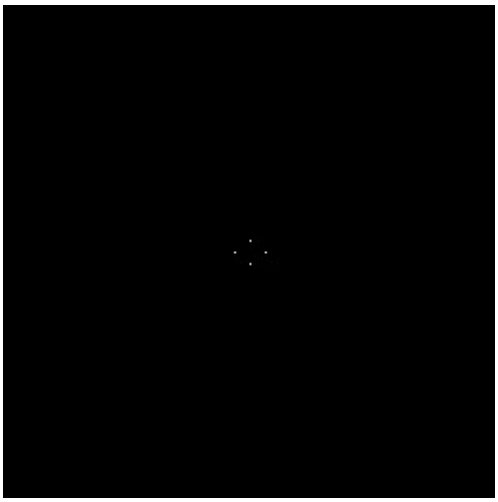
2. Same as 1, but swapping the two directions, for various s values:

S = 2: 	Magnitude: 
S = 6: 	Magnitude: 
S = 18 	Magnitude: 

Part B: Addition

1. Add two images from Part A together (I added f[x,y] = sin(2*pi*8*x/N) and f[x,y] = sin(2*pi*6*y/N))

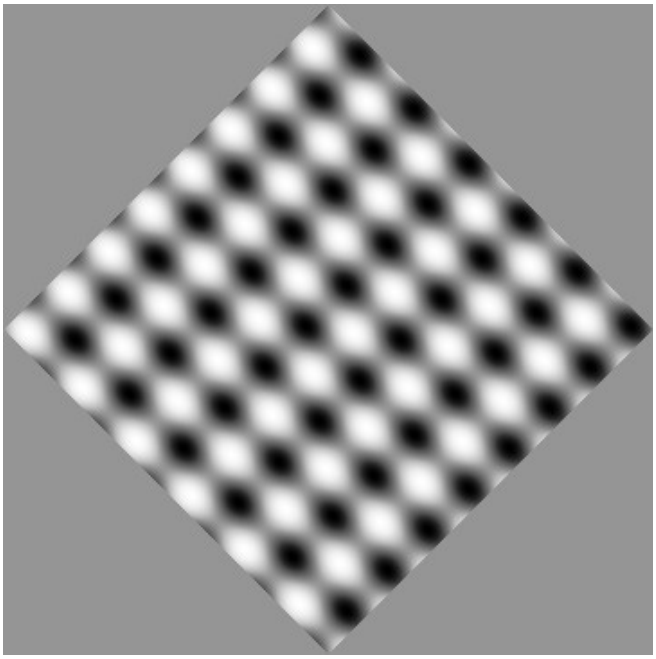
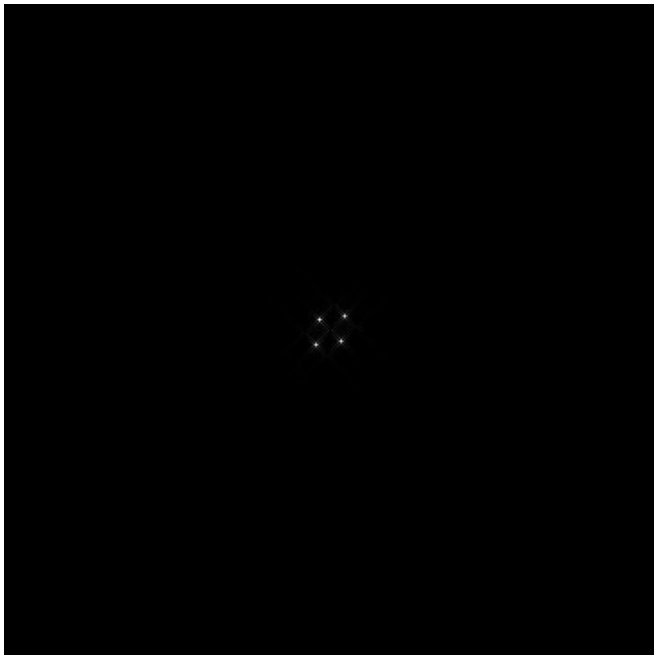
2.

Sum of the two images:	Magnitude of fft of this sum:
	

Explanation: The white colored areas in the magnitude image correspond to the +/- s values in the u and v directions (8 and 6), basically the combination of the individual corresponding magnitude images. The sum of the two images has 8 patterns going horizontally along the u-axis and 6 going vertically along the v-axis, as should be expected since that corresponds to the frequencies in the individual images.

Part C: Rotation:

1 and 2. Rotate image produced from part B, plot magnitude of its Fourier Transform

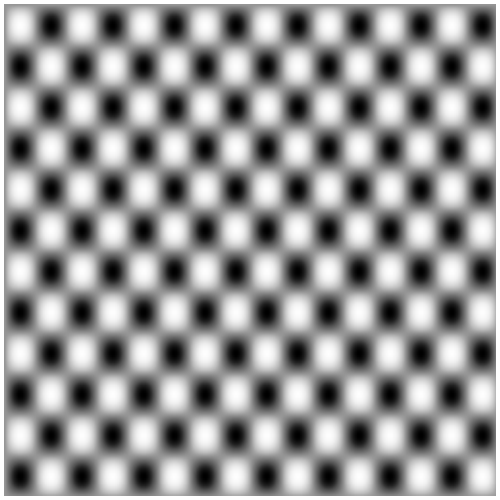
	

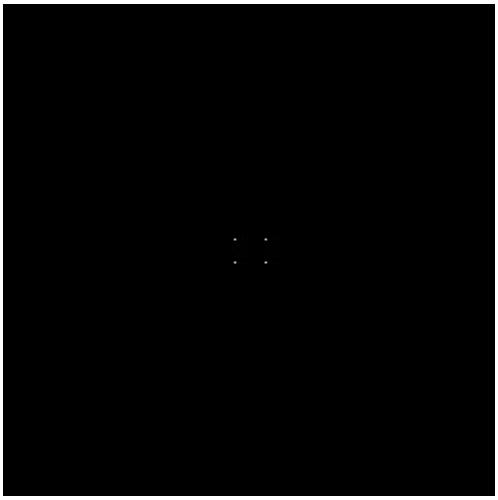
Explanation: Since the Fourier Transform is linear, it makes sense that the image above on the right

(the magnitude of the Fourier Transform of the rotated image) looks just as if you had simply rotated the magnitude of the Fourier Transform of the original image. That is, rot(mag(F(original))) = mag(F(rot(original))).

Part D: Multiplication:

1. Same as B, but multiplying instead of adding the two sinusoids.





Explanation: By multiplying the two images, we multiply the two element wise. I think that we get the clear checkerboard pattern here, as opposed to part b, because multiplying takes into account edges properly. In we had an array A = [[-2, -0.5, 0, 0.5, 2], …] and B = [[0, 0, 0, 0, 0], …], (pretend that A is like the sinusoid changing in in the x-direction, and B the y-direction), adding them producing a new array where each element in a summation of the both, and therefore where B was once a zero and unchanging, the new element shows a change in that corresponding position. Multiplication, on the other hand, preserves the rows/columns (depending on which general sinusoid we are using) that are 0, since anything multiplied by zero is zero.

Part E: Magnitude and Phase:

Figure 1: Inverse fft of (Magnitude(B(u,v)), Phase(G(u,v))):

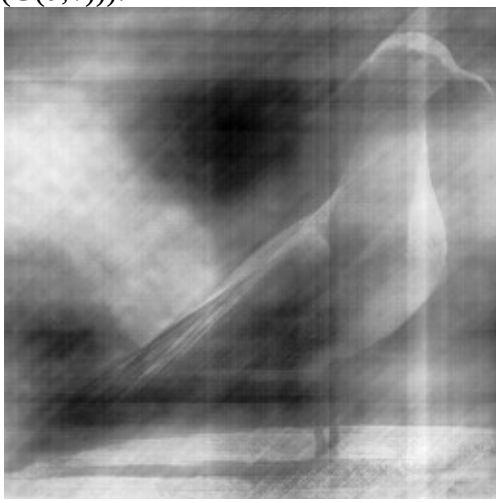


Figure 2: Inverse fft of (Magnitude(G(u,v)), Phase(B(u,v))):

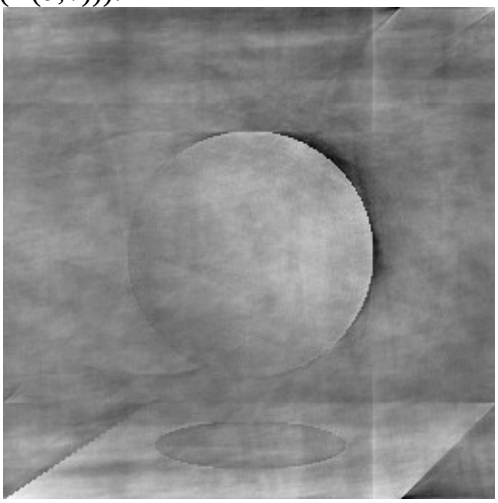


Figure 1 looks most like the gull (it gets this contribution from the Phase(G(u,v))) and Figure 2 looks most like the Ball (which again gets this from the contribution of the Phase(B(u,v))).

Challenges:
My main challenge was with the last question in determining what was meant by create a pair (I received the necessary information from Dr. Farrell and then it made sense). The only other challenge I have is consistent across all these projects-- not knowing all the 'secrets'/shortcuts built-in to numpy, scipy, PIL, matplotlib.pyplot, etc. I mainly refer to being able to read in an image and only specific channels, which I worked around not knowing how to do by creating arrays that were copies of the images read in but just 1 dimension smaller.

It took about 3 hours to do this assignment.

Code:
```
import numpy as np
import matplotlib.pyplot as pl
import matplotlib.cm as cm
import pickle as pk
import scipy.ndimage.interpolation as sy

#I will be using the built-in fft2/ifft2 functions

def getSinx(s):
    N = 256
    sine = np.zeros((N, N))
    for y in range(N):
        for x in range(N):
            sine[y][x] = np.sin(2 * np.pi * s * x / N)
    return sine

def getSiny(s):
    N = 256
    sine = np.zeros((N, N))
    for y in range(N):
        for x in range(N):
            sine[y][x] = np.sin(2 * np.pi * s * y / N)
    return sine

def parta():
    #1:Make a two-dim image that is a sinusoid in one direction
    #f[x,y] = sin(2 * pi * s * x / N)
    #produces vertical lines
    svals = (4, 8, 16, 32)
    for s in svals:
        sine = getSinx(s)
        fft2sine = np.fft.fft2(sine)
        shifted = np.fft.fftshift(fft2sine)

        pl.imsave('f[x,y] = sin(2 pi %d x div N)' % s, sine, cmap=cm.Greys_r)
        pl.imsave('Magnitude with s = %d' % s, np.abs(shifted), cmap=cm.Greys_r)

    #2: Swap the two directions
    #f[x,y] = sing(2 * pi * s * y / N)
    #produces horizontal lines
```

```
    svals = (2, 6, 18)
    for s in svals:
        sine = getSiny(s)
        fft2sine = np.fft.fft2(sine)
        shifted = np.fft.fftshift(fft2sine)

        pl.imsave('f[x,y] = sin(2 pi %d x div N)' % s, sine, cmap=cm.Greys_r)
        pl.imsave('Magnitude with s = %d' % s, np.abs(shifted), cmap=cm.Greys_r)

def partb():
    sine1 = getSinx(8)
    sine2 = getSiny(6)

    sum = np.add(sine1, sine2)
    fft2sine = np.fft.fft2(sum)
    shifted = np.fft.fftshift(fft2sine)

    pl.imsave('Sum of two images, sin(x) with s = 8 and sin(y) with s = 6', sum,
cmap=cm.Greys_r)
    pl.imsave('Magnitude of two images, sin(x) and sin(y)', np.abs(shifted),
cmap=cm.Greys_r)


def partc():
    sine1 = getSinx(8)
    sine2 = getSiny(6)

    sum = np.add(sine1, sine2)
    rotation = sy.rotate(sum, 45.0)
    pl.imsave('Sum of two images, rotated', rotation, cmap=cm.Greys_r)

    fft2sine = np.fft.fft2(rotation)
    shifted = np.fft.fftshift(fft2sine)
    pl.imsave('Magnitude of fft of rotated sum of two images', np.abs(shifted),
cmap=cm.Greys_r)

def partd():
    sine1 = getSinx(8)
    sine2 = getSiny(6)

    mult = np.multiply(sine1, sine2)
    fft2sine = np.fft.fft2(mult)
    shifted = np.fft.fftshift(fft2sine)
    pl.imsave('Multiplication of two images, sin(x) with s = 8 and sin(y) with s=
6', mult, cmap=cm.Greys_r)
    pl.imsave('Magnitude of fft of multiplication of two images', np.abs(shifted),
cmap=cm.Greys_r)

def parte():
    ball3d = pl.imread('ball.png') #b(x,y)
    gull3d = pl.imread('gull.png') #g(x,y)
    #To verify I read it in correctly
    #pl.imshow(ball)
    #pl.imshow(gull)
    #pl.show()

    #There is most assuredly an easier way to do this, but time is of the essence
and I don't want to study man pages right now
```

```
    ball = np.zeros((ball3d.shape[0], ball3d.shape[1]), dtype=np.double)
    gull = np.zeros((gull3d.shape[0], gull3d.shape[1]), dtype=np.double)
    for a in range(ball.shape[0]):
        for b in range(ball.shape[1]):
            ball[a][b] = ball3d[a][b][0]
            gull[a][b] = gull3d[a][b][0]

    fftBall = np.fft.fft2(ball) #B(u,v)
    fftGull = np.fft.fft2(gull) #G(u,v)

    magBall = np.abs(fftBall)
    magGull = np.abs(fftGull)
    phaseBall = np.zeros(fftBall.shape)
    phaseGull = np.zeros(fftGull.shape)
    for a in range(fftBall.shape[0]):
        for b in range(fftBall.shape[1]):
            phaseBall[a][b] = np.angle(fftBall[a][b])
            phaseGull[a][b] = np.angle(fftGull[a][b])

    magBphaseG = np.zeros((magBall.shape[0], magBall.shape[1]), dtype=complex)
    magGphaseB = np.zeros((magGull.shape[0], magGull.shape[1]), dtype=complex)

    for x in range(magBphaseG.shape[0]):
        for y in range(magBphaseG.shape[1]):
            magBphaseG[x][y] = np.complex(magBall[x][y]*np.cos(phaseGull[x][y]),
magBall[x][y]*np.sin(phaseGull[x][y]))
            magGphaseB[x][y] = np.complex(magGull[x][y]*np.cos(phaseBall[x][y]),
magGull[x][y]*np.sin(phaseBall[x][y]))
    ifft01 = np.fft.ifft2(magBphaseG)
    ifft02 = np.fft.ifft2(magGphaseB)
    pl.imsave('IFFT of (Mag(B(u,v)),Phase(G(u,v))', ifft01, cmap=cm.Greys_r)
    pl.imsave('IFFT of (Mag(G(u,v)),Phase(B(u,v))', ifft02, cmap=cm.Greys_r)

def main():
    parta() #simple sines and cosines
      partb() #addition
      partc() #rotation
      partd() #multiplication
      parte() #magnitude and phase

if __name__ == "__main__":
    main()
```