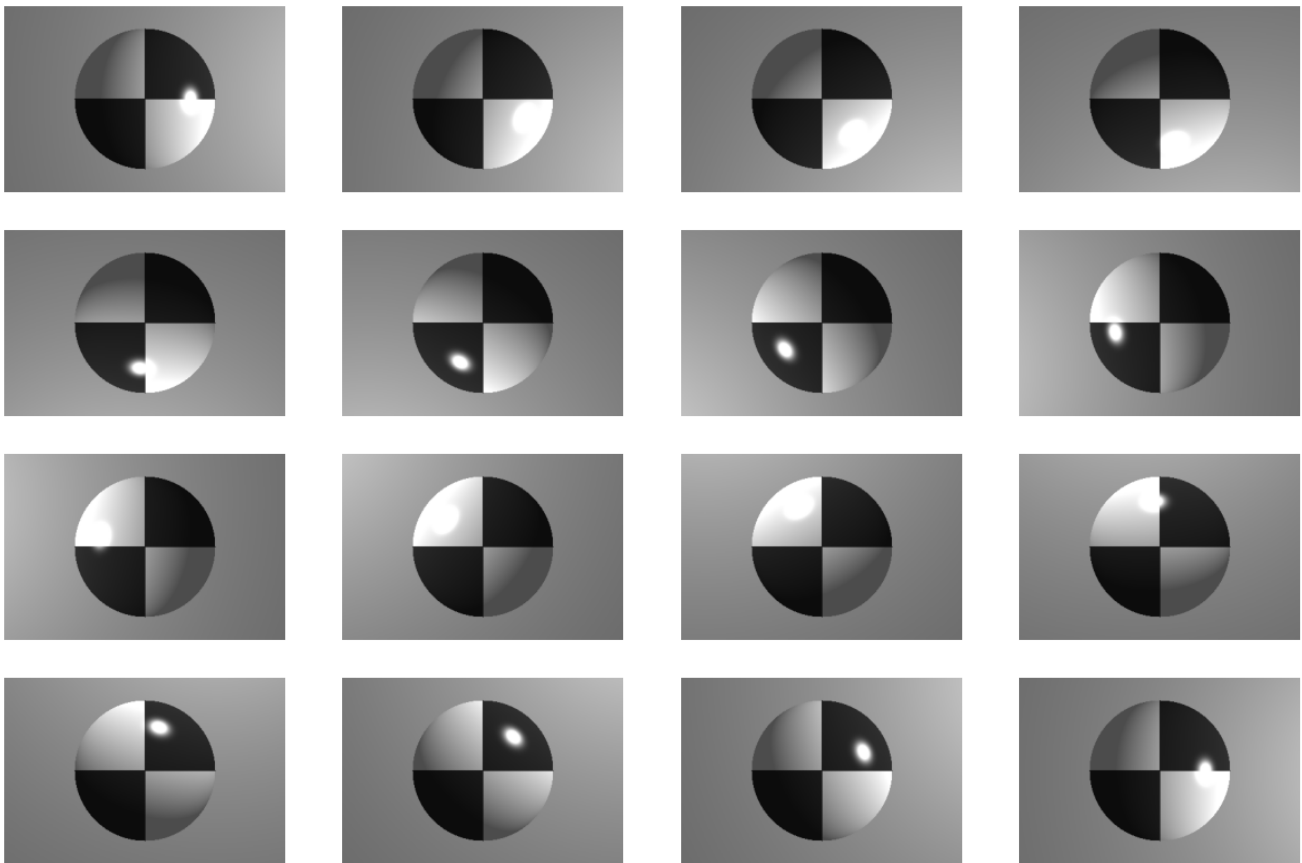


CS543 MP1 Q3

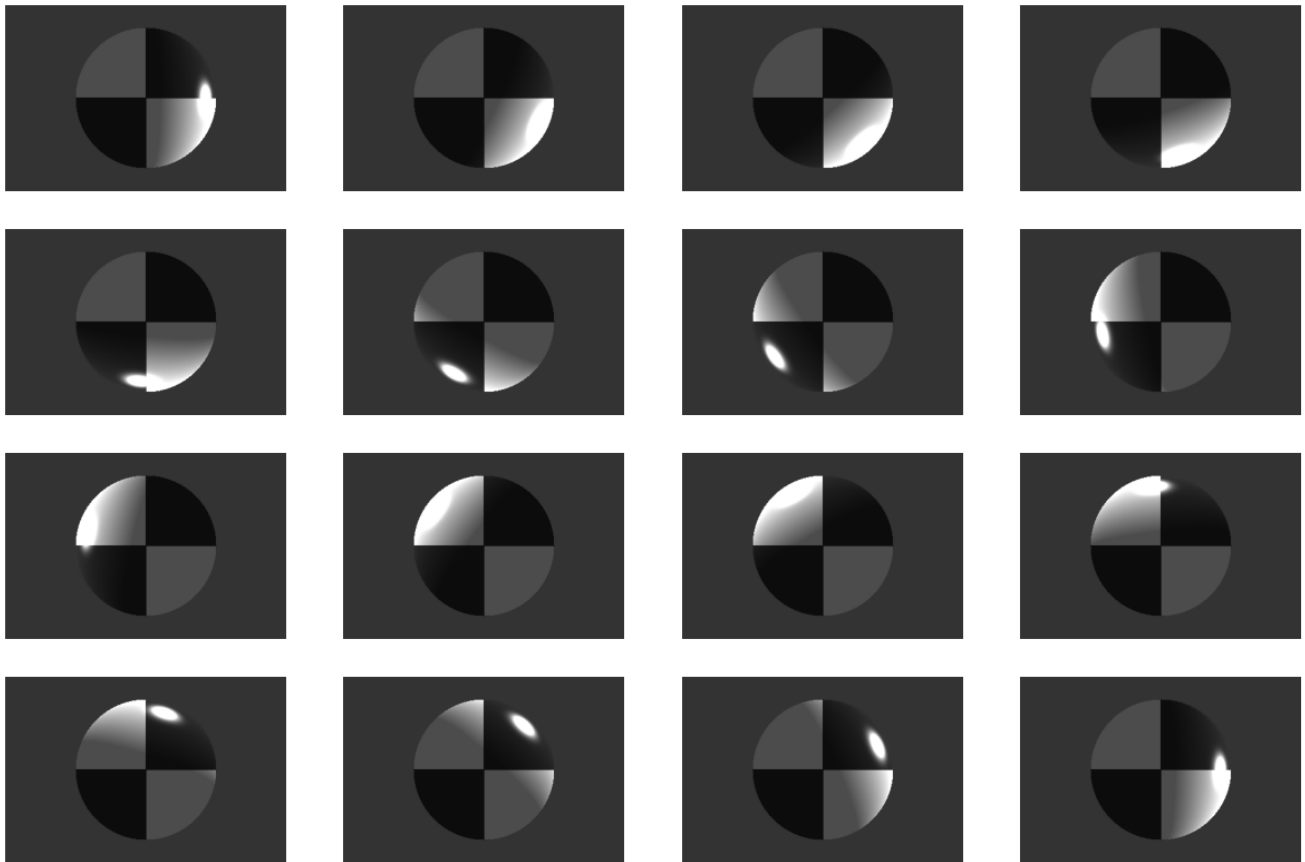
The results I generate:

The picture generated:

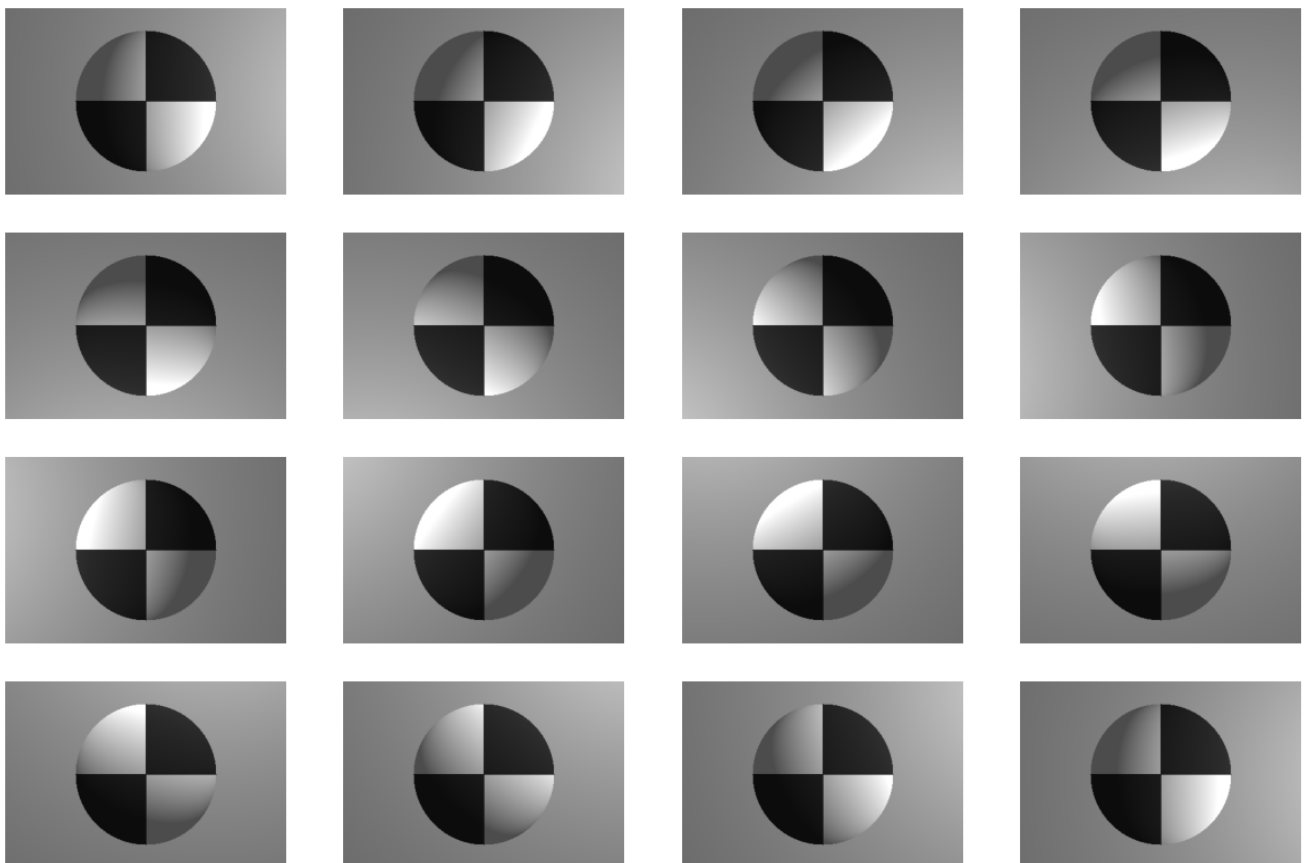
- The output of with specular and point light



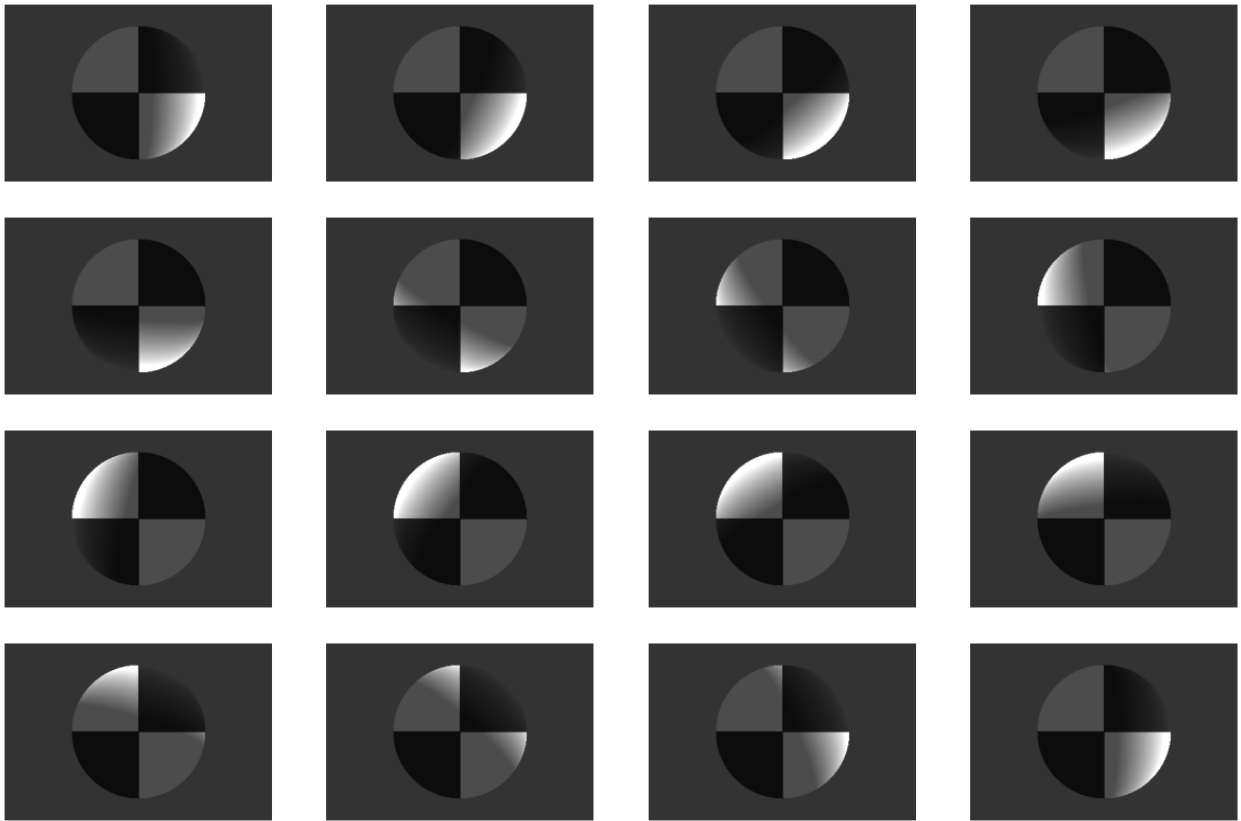
- The output of with specular and directional light



- The output of without specular and point light



- The output of without specular and directional light



My algorithm:

- For n , it is the N given by the argument :
- For v_i ,
 - in directional light situation, v_i is `directional_light_dirn` with normalization
 - in point light situation,
 - we need to calculate the vector from `point_light_loc` to the location on the surface.
 - For the location on the surface, we need

```
x = x - cx
y = y - cy
X = x / f * Z
Y = y / f * Z
```

to calculate the X and Y

So we have the X, Y, Z for all the points on the surface

- Then, normalize v_i .

- For v_r ,
 - v_r is the direction from camera to the location, so we use the (0,0,0) - location on the surface to get v_r .
 - Then, normalize v_r .
- For s_i is the reflection point of, the v_i is the incident vector, N is the normal vector, so we need to calculate the s_i as the reflective vector.
 - The angle between incident vector and normal vector is the same as the angle between reflective vector and the normal vector
 - Suppose AO is incident vector and OB is reflective vector, OP is normal vector.

$$\begin{aligned}
 OB &= AB - AO, \\
 AB &= 2(AO + OP) \\
 OB &= AO + 2OP \\
 OA' &= (OA \cdot N) * N \\
 \\
 \implies OB &= AO - 2(AO \cdot N) * N
 \end{aligned}$$

The code:

```

# specular exponent
k_e = 50

def render(Z, N, A, S,
           point_light_loc, point_light_strength,
           directional_light_dirn, directional_light_strength,
           ambient_light, k_e):
    # To render the images you will need the camera parameters, you can assume
    # the following parameters.
    #(cx, cy) denote the center of the image (point
    # where the optical axis intersects with the image, f is the focal length.
    # These parameters along with the depth image will be useful for you to
    # estimate the 3D points on the surface of the sphere for computing the
    # angles between the different directions.
    # N.shape #(256, 384, 3)
    # Z.shape #(256, 384)
    # vi: directional_light_dirn.shape: (1, 3)
    # point_light_loc = [[0, -10, 2]]
    h, w = A.shape
    cx, cy = w / 2, h / 2
    f = 128.

```

```

#     print(Z.shape)
# Ambient Term
I = A * ambient_light

#Diffuse Term
#point_light_strength * [dot product of v_i (for point light) and n] +
directional_light_strength * [dot product of v_i (for directional light) and n]

#take care vi and n as shape (256, 384, 3)
N = np.array(N)
n1, n2, n3 = N.shape
directional_light_dirn = np.array(directional_light_dirn)
directional_light_dirn = directional_light_dirn - np.zeros(N.shape)
for i in range(n1):
    for j in range(n2):
        directional_light_dirn[i][j] = directional_light_dirn[i][j]/
np.linalg.norm(directional_light_dirn[i][j])

#     directional_light_dirn = directional_light_dirn/
np.linalg.norm(directional_light_dirn)
#D for directional_light
Li_vi_dot_n_directional = directional_light_strength[0] *
helper(directional_light_dirn, N)
D_directional_light = np.multiply(A, Li_vi_dot_n_directional)

#calculate vi for point light
point_light_loc = np.array(point_light_loc)
point_light_dirn = np.zeros((n1,n2,n3))

x, y = np.meshgrid(np.arange(w), np.arange(h))
x = x - cx
y = y - cy
X = x / f * Z
Y = y / f * Z

for i in range(n1):
    for j in range(n2):
        point_light_dirn[i][j] = point_light_loc - np.array([X[i][j],Y[i]
[j],Z[i][j]])
        point_light_dirn[i][j] = point_light_dirn[i][j]/
np.linalg.norm(point_light_dirn[i][j])

Li_vi_dot_n_point = point_light_strength[0] * helper(point_light_dirn, N)
D_point_light = np.multiply(A, Li_vi_dot_n_point)

```

```

D = D_directional_light + D_point_light

# Specular Term
vr = np.zeros((n1,n2,n3))
for i in range(n1):
    for j in range(n2):
        vr[i][j] = -np.array([X[i][j],Y[i][j],Z[i][j]])
        vr[i][j] = vr[i][j]/ np.linalg.norm(vr[i][j])

#calculate si
directional_light_si = np.zeros(directional_light_dirn.shape)
for i in range(n1):
    for j in range(n2):
        directional_light_si[i][j] = -directional_light_dirn[i][j] - 2 *
(np.dot(-directional_light_dirn[i][j], N[i][j]))* N[i][j]

point_light_si = np.zeros(point_light_dirn.shape)
for i in range(n1):
    for j in range(n2):
        point_light_si[i][j] = -point_light_dirn[i][j] - 2 *(np.dot(-
point_light_dirn[i][j], N[i][j]))* N[i][j]

vr_si_directional = helper(vr,directional_light_si)
vr_si_point = helper(vr,point_light_si)

S_directional = S * directional_light_strength[0] * vr_si_directional**k_e
S_point = S * point_light_strength[0] * vr_si_point**k_e

S = S_directional + S_point

I = I + D + S

I = np.minimum(I, 1)*255
I = I.astype(np.uint8)
I = np.repeat(I[:, :, np.newaxis], 3, axis=2)

return I

def helper(directional_light_dirn, N):
    n1, n2, n3 = N.shape
    ans = np.zeros((n1,n2))
    for i in range(n1):
        for j in range(n2):

```

```
        tmp = directional_light_dirn[i][j][0]* N[i][j][0] +  
directional_light_dirn[i][j][1]* N[i][j][1] + directional_light_dirn[i][j][2]*  
N[i][j][2]  
        if tmp > 0:  
            ans[i][j] = tmp  
    return ans
```

