CS445: Computational Photography

Programming Project 4: Image-Based Lighting

Recovering HDR Radiance Maps

Load libraries and data

In [1]:

```
# jupyter extension that allows reloading functions from imports without clear
ing kernel :D
%load_ext autoreload
%autoreload 2
```

In [2]:

```
# System imports
from os import path
import math

# Third-Party Imports
import cv2
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import griddata

# modify to where you store your project data including utils
datadir = "C:/Users/jackt/Desktop/CS445/cs445lighting/"

# utilfn = datadir + "utils"
# !cp -r "$utilfn" .
# samplesfn = datadir + "samples"
# !cp -r "$samplesfn" .

# can change this to your output directory of choice
# !mkdir "images"
# !mkdir "images/outputs"

# import starter code
import utils
from utils.io import read_image, write_image, read_hdr_image, write_hdr_image
from utils.display import display_images_linear_rescale, rescale_images_linear
from utils.hdr_helpers import gsolve
from utils.hdr_helpers import get_equirectangular_image
from utils.bilateral_filter import bilateral_filter
```

Reading LDR images

You can use the provided samples or your own images. You get more points for using your own images, but it might help to get things working first with the provided samples.

In [3]:
```
# TODO: Replace this with your path and files

imdir = 'samples'
imfns = ['0024.jpg', '0060.jpg', '0120.jpg', '0205.jpg', '0553.jpg']
exposure_times = [1/24.0, 1/60.0, 1/120.0, 1/205.0, 1/553.0]

ldr_images = []
for f in np.arange(len(imfns)):
  im = read_image(imdir + '/' + imfns[f])
  if f==0:
    imsize = int((im.shape[0] + im.shape[1])/2) # set width/height of ball ima
ges
    ldr_images = np.zeros((len(imfns), imsize, imsize, 3))
  ldr_images[f] = cv2.resize(im, (imsize, imsize))

background_image_file = imdir + '/' + 'empty.jpg'
background_image = read_image(background_image_file)
```

Naive LDR merging

Compute the HDR image as average of irradiance estimates from LDR images

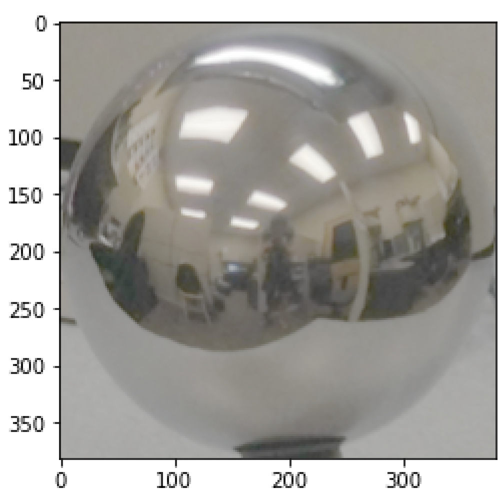
In [4]:
```
def make_hdr_naive(ldr_images: np.ndarray, exposures: list) -> (np.ndarray, np
.ndarray):
    '''
    Makes HDR image using multiple LDR images, and its corresponding exposure
 values.

    The steps to implement:
    1) Divide each image by its exposure time.
        - This will rescale images as if it has been exposed for 1 second.

    2) Return average of above images


    For further explanation, please refer to problem page for how to do it.

    Args:
        ldr_images(np.ndarray): N x H x W x 3  shaped numpy array representing
            N ldr images with width W, height H, and channel size of 3 (RGB)
        exposures(list): list of length N, representing exposures of each imag
es.
            Each exposure should correspond to LDR images' exposure value.
    Return:
        (np.ndarray): H x W x 3 shaped numpy array representing HDR image merg
ed using
            naive ldr merging implementation.
        (np.ndarray): N x H x W x 3  shaped numpy array represending log irrad
iances
            for each exposures

    '''
    N, H, W, C = ldr_images.shape
    # sanity check
    assert N == len(exposures)

    sum_image = np.zeros((H, W, C))
    irrad = np.zeros((N, H, W, C))

    for i in range(N):
        # process i-th image
        irrad[i] = ldr_images[i] / exposures[i]
        sum_image += irrad[i]

    hdr_image = sum_image / N
    log_irrad = np.log(irrad)

    return hdr_image, log_irrad
```

In [5]:
```
def display_hdr_image(im_hdr):
    '''
    Maps the HDR intensities into a 0 to 1 range and then displays.
    Three suggestions to try:
      (1) Take log and then linearly map to 0 to 1 range (see display.py for e
xample)
      (2) img_out = im_hdr / (1 + im_hdr)
      (3) HDR display code in a python package
    '''
    im_out = np.log(im_hdr)
    im_out = rescale_images_linear(im_out)
    #im_out = np.divide(im_hdr, 1+im_hdr)
    plt.imshow(im_out)
    plt.show()
```

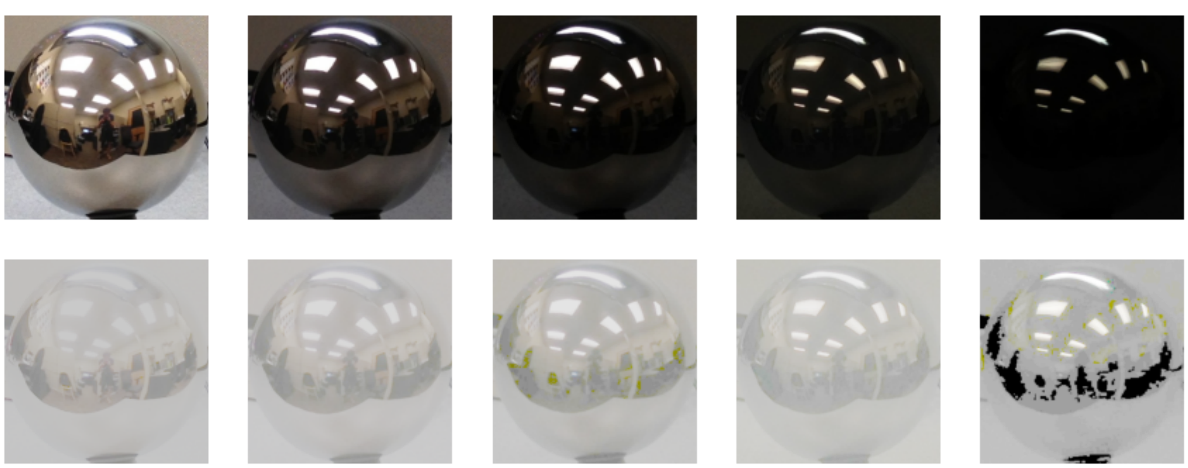
In [6]:
```python
# get HDR image, log irradiance
naive_hdr_image, naive_log_irradiances = make_hdr_naive(ldr_images, exposure_t
imes)

# write HDR image to directory
write_hdr_image(naive_hdr_image, 'images/outputs/naive_hdr.hdr')

# display HDR image
print('HDR Image')
display_hdr_image(naive_hdr_image)

# display original images (code provided in utils.display)
display_images_linear_rescale(ldr_images)

print('irradiances')

# display log irradiance image (code provided in utils.display)
display_images_linear_rescale(naive_log_irradiances)
```

```
c:\python\python37\lib\site-packages\ipykernel_launcher.py:39: RuntimeWarnin
g: divide by zero encountered in log
```

```
HDR Image
```




```
irradiances
```



Weighted LDR merging

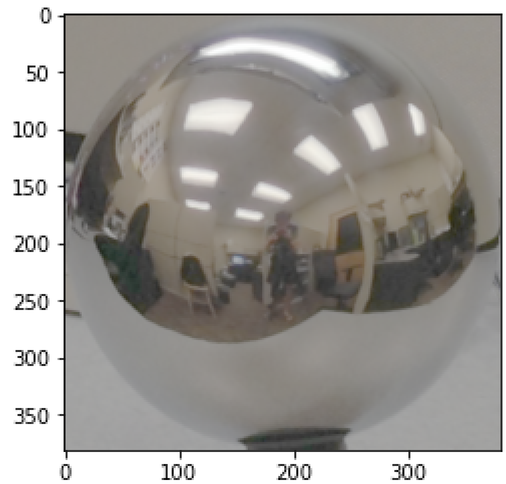
Compute HDR image as a weighted average of irradiance estimates from LDR images, where weight is based on pixel intensity so that very low/high intensities get less weight

In [7]:
```python
def make_hdr_weighted(ldr_images: np.ndarray, exposure_times: list) -> (np.nda
rray, np.ndarray):
    '''
    Makes HDR image using multiple LDR images, and its corresponding exposure
 values.

    The steps to implement:
    1) compute weights for images with based on intensities for each exposures
        - This can be a binary mask to exclude low / high intensity values

    2) Divide each images by its exposure time.
        - This will rescale images as if it has been exposed for 1 second.

    3) Return weighted average of above images


    Args:
        ldr_images(np.ndarray): N x H x W x 3 shaped numpy array representing
            N ldr images with width W, height H, and channel size of 3 (RGB)
        exposure_times(list): list of length N, representing exposures of each
images.
            Each exposure should correspond to LDR images' exposure value.
    Return:
        (np.ndarray): H x W x 3 shaped numpy array representing HDR image merg
ed without
            under - over exposed regions

    '''
    N, H, W, C = ldr_images.shape
    # sanity check
    assert N == len(exposure_times)

    sum_image = np.zeros((H, W, C))
    weights   = np.zeros((H, W, C))
    irrad = np.zeros((N, H, W, C))

    wgt = lambda z : float(128-abs(z-128)) if z < 256 and z > 0 else 0.01

    for i in range(N):
        # process i-th image
        irrad[i] = ldr_images[i] / exposure_times[i]

        for h in range(H):
            for w in range(W):
                for c in range(C):
                    temp = wgt(ldr_images[i,h,w,c])
                    weights[h,w,c] += temp
                    sum_image[h,w,c] += irrad[i,h,w,c] * temp

    hdr_image = sum_image / weights
    log_irrad = np.log(irrad)

    return hdr_image, log_irrad
```

In [8]:

```
# get HDR image, log irradiance
weighted_hdr_image, weighted_log_irradiances = make_hdr_weighted(ldr_images, e
xposure_times)

# write HDR image to directory
write_hdr_image(weighted_hdr_image, 'images/outputs/weighted_hdr.hdr')

# display HDR image
display_hdr_image(weighted_hdr_image)


display_images_linear_rescale(weighted_log_irradiances)
```

```
c:\python\python37\lib\site-packages\ipykernel_launcher.py:47: RuntimeWarnin
g: divide by zero encountered in log
```



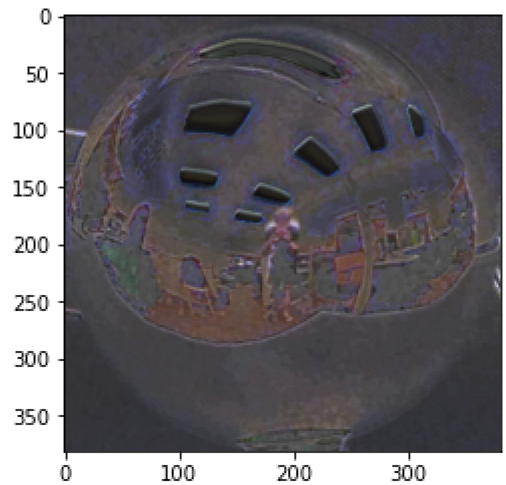
Display of difference between naive and weighted for your own inspection

Where does the weighting make a big difference increasing or decreasing the irradiance estimate? Think about why.

In [9]:
```
# display difference between naive and weighted

log_diff_im = np.log(weighted_hdr_image)-np.log(naive_hdr_image)
print('Min ratio = ', np.exp(log_diff_im).min(), '  Max ratio = ', np.exp(log_
diff_im).max())
plt.figure()
plt.imshow(rescale_images_linear(log_diff_im))
```

```
Min ratio =  0.7302035087611243   Max ratio =  2.8387186691182267
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x22d652a29b0>
```



LDR merging with camera response function estimation

Compute HDR after calibrating the photometric reponses to obtain more accurate irradiance estimates from each image

Some suggestions on using `gsolve`:

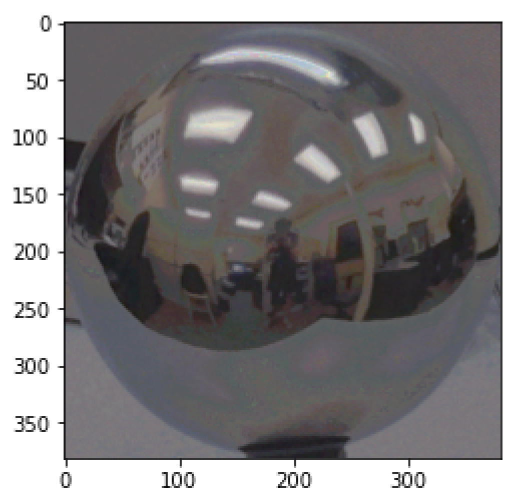
- When providing input to gsolve, don't use all available pixels, otherwise you will likely run out of memory / have very slow run times. To overcome, just randomly sample a set of pixels (1000 or so can suffice), but make sure all pixel locations are the same for each exposure.
- The weighting function w should be implemented using Eq. 4 from the paper (this is the same function that can be used for the previous LDR merging method).
- Try different lambda values for recovering *g*. Try lambda=1 initially, then solve for *g* and plot it. It should be smooth and continuously increasing. If lambda is too small, g will be bumpy.
- Refer to Eq. 6 in the paper for using g and combining all of your exposures into a final image. Note that this produces log irradiance values, so make sure to exponentiate the result and save irradiance in linear scale.

In [10]:
```
def make_hdr_estimation(ldr_images: np.ndarray, exposure_times: list, lm)-> (n
p.ndarray, np.ndarray):
    '''
    Makes HDR image using multiple LDR images, and its corresponding exposure
 values.
    Please refer to problem notebook for how to do it.

    **IMPORTANT**
    The gsolve operations should be ran with:
        Z: int64 array of shape N x P, where N = number of images, P = number
 of pixels
        B: float32 array of shape N, log shutter times
        l: lambda; float to control amount of smoothing
        w: function that maps from float intensity to weight
    The steps to implement:
    1) Create random points to sample (from mirror ball region)
    2) For each exposures, compute g values using samples
    3) Recover HDR image using g values


    Args:
        ldr_images(np.ndarray): N x H x W x 3 shaped numpy array representing
            N ldr images with width W, height H, and channel size of 3 (RGB)
        exposures(list): list of length N, representing exposures of each imag
es.
            Each exposure should correspond to LDR images' exposure value.
        lm (scalar): the smoothing parameter
    Return:
        (np.ndarray): H x W x 3 shaped numpy array representing HDR image merg
ed using
            gsolve
        (np.ndarray): N x H x W x 3 shaped numpy array represending log irradi
ances
            for each exposures
        (np.ndarray): 3 x 256 shaped numpy array represending g values of each
pixel intensities
            at each channels (used for plotting)
    '''

    N, H, W, C = ldr_images.shape
    # sanity check
    assert N == len(exposure_times)

    # 32*32 grid of pixels selected, P=1024
    grid_size = 10
    Z = np.zeros((N, grid_size * grid_size, C), dtype=np.int64)
    B = np.log(exposure_times, dtype=np.float32)
    w = lambda z : float(128-abs(z-128))

    ldr_big = ldr_images * 255

    flat = lambda a, b : a * grid_size + b
    delta = min(H, W) // grid_size

    # prepare Z before gsolve
    for c in range(C):
```

```
        for i in range(grid_size):
            for j in range(grid_size):
                Z[:,flat(i, j),c] = ldr_big[:, i*delta, j*delta, c]

    # solve for g
    g = np.zeros((C, 256))

    g[0], lE = gsolve(Z[:,:,0], B, lm, w)
    g[1], lE = gsolve(Z[:,:,1], B, lm, w)
    g[2], lE = gsolve(Z[:,:,2], B, lm, w)

    # recover HDR using g values
    im_out = np.zeros((H, W, C))
    log_irrad = np.zeros((N, H, W, C))

    for c in range(C):
        for i in range(H):
            for j in range(W):
                num = 0.0
                den = 0.0
                for k in range(N):
                    z = int(ldr_big[k,i,j,c])
                    log_irrad[k,i,j,c] = (g[c,z] - B[k])
                    num += w(z)*(g[c,z] - B[k])
                    den += w(z)
                im_out[i,j,c] = num/den
    return np.exp(im_out), log_irrad, g
```

In [11]:

```
lm = 5
# get HDR image, log irradiance
calib_hdr_image, calib_log_irradiances, g = make_hdr_estimation(ldr_images, ex
posure_times, lm)

# write HDR image to directory
write_hdr_image(calib_hdr_image, 'images/outputs/calib_hdr.hdr')

# display HDR image
display_hdr_image(calib_hdr_image)
```



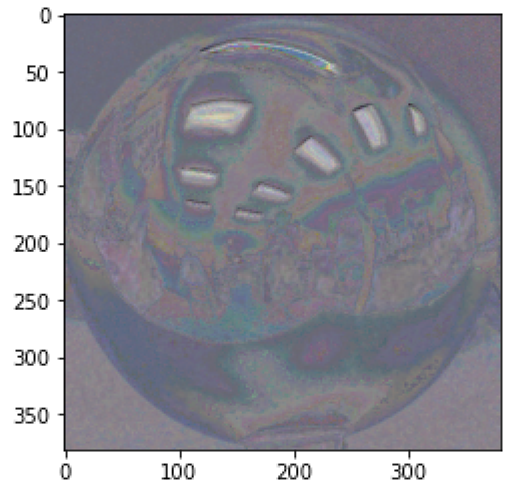
The following code displays your results. You can copy the resulting images and plots directly into your report where appropriate.

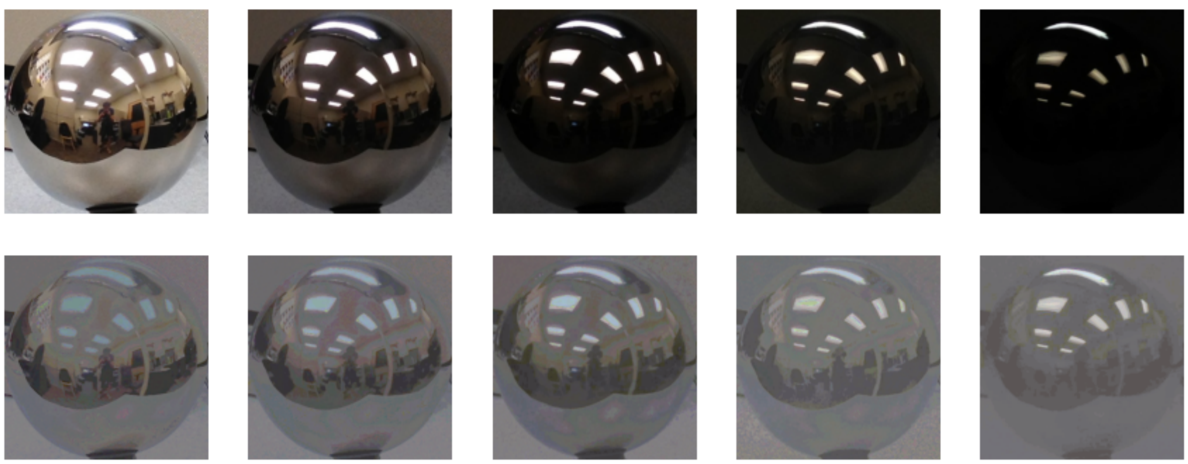
In [12]:

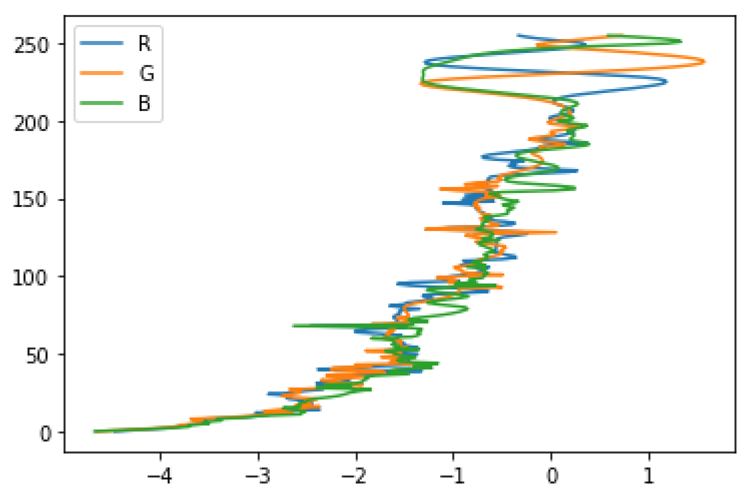
```
# display difference between calibrated and weighted
log_diff_im = np.log(calib_hdr_image/calib_hdr_image.mean())-np.log(weighted_h
dr_image/weighted_hdr_image.mean())
print('Min ratio = ', np.exp(log_diff_im).min(), '  Max ratio = ', np.exp(log_
diff_im).max())
plt.figure()
plt.imshow(rescale_images_linear(log_diff_im))

# display original images (code provided in utils.display)
display_images_linear_rescale(ldr_images)

# display log irradiance image (code provided in utils.display)
display_images_linear_rescale(calib_log_irradiances)

# plot g vs intensity, and then plot intensity vs g
N, NG = g.shape
labels = ['R', 'G', 'B']
plt.figure()
for n in range(N):
    plt.plot(g[n], range(NG), label=labels[n])
plt.gca().legend(('R', 'G', 'B'))

plt.figure()
for n in range(N):
    plt.plot(range(NG), g[n], label=labels[n])
plt.gca().legend(('R', 'G', 'B'))
```

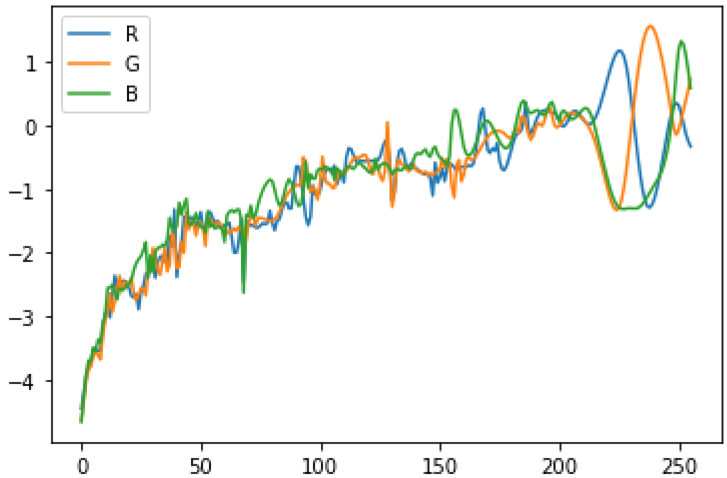
```
Min ratio =  0.14679170045133771   Max ratio =  5.636165534019351
```

```
Out[12]: <matplotlib.legend.Legend at 0x22d6ae0ba20>
```








R
G
B
1
0
-1
-2
-3
-4
0
50
100
150
200
250

In [13]:
```
def weighted_log_error(ldr_images, hdr_image, log_irradiances):
  # computes weighted RMS error of log irradiances for each image compared to
 final log irradiance
  N, H, W, C = ldr_images.shape
  w = 1-abs(ldr_images - 0.5)*2
  err = 0
  for n in np.arange(N):
    err += np.sqrt(np.multiply(w[n], (log_irradiances[n]-np.log(hdr_image))**2
).sum()/w[n].sum())/N
  return err


# compare solutions
err = weighted_log_error(ldr_images, naive_hdr_image, naive_log_irradiances)
print('naive:  \tlog range = ', round(np.log(naive_hdr_image).max() - np.log(n
aive_hdr_image).min(),3), '\tavg RMS error = ', round(err,3))
err = weighted_log_error(ldr_images, weighted_hdr_image, naive_log_irradiances
)
print('weighted:\tlog range = ', round(np.log(weighted_hdr_image).max() - np.l
og(weighted_hdr_image).min(),3), '\tavg RMS error = ', round(err,3))
err = weighted_log_error(ldr_images, calib_hdr_image, calib_log_irradiances)
print('calibrated:\tlog range = ', round(np.log(calib_hdr_image).max() - np.lo
g(calib_hdr_image).min(),3), '\tavg RMS error = ', round(err,3))

# display log hdr images (code provided in utils.display)
display_images_linear_rescale(np.log(np.stack((naive_hdr_image/naive_hdr_image
.mean(), weighted_hdr_image/weighted_hdr_image.mean(), calib_hdr_image/calib_h
dr_image.mean()), axis=0)))
```

```
naive:          log range =  6.462      avg RMS error =  0.324
weighted:       log range =  5.966      avg RMS error =  0.332
calibrated:     log range =  7.32       avg RMS error =  0.301
```

Panoramic transformations

Compute the equirectangular image from the mirrorball image

In [ ]:

```
def panoramic_transform(hdr_image):
    '''
    Given HDR mirror ball image,

    Expects mirror ball image to have center of the ball at center of the imag
e, and
    width and height of the image to be equal.

    Steps to implement:
    1) Compute N image of normal vectors of mirror ball
    2) Compute R image of reflection vectors of mirror ball
    3) Map reflection vectors into spherical coordinates
    4) Interpolate spherical coordinate values into equirectangular grid.

    Steps 3 and 4 are implemented for you with get_equirectangular_image

    '''
    H, W, C = hdr_image.shape
    assert H == W
    assert C == 3

    # TO DO: compute N and R

    # R = V - 2 * dot(V,N) * N



    plt.imshow((N+1)/2)
    plt.show()
    plt.imshow((R+1)/2)
    plt.show()

    equirectangular_image = get_equirectangular_image(R, hdr_image)
    return equirectangular_image
```

In [ ]:

```
hdr_mirrorball_image = read_hdr_image('images/outputs/calib_hdr.hdr')
eq_image = panoramic_transform(hdr_mirrorball_image)

write_hdr_image(eq_image, 'images/outputs/equirectangular.hdr')

plt.figure(figsize=(15,15))
display_hdr_image(eq_image)
```

Rendering synthetic objects into photographs

Use Blender to render the scene with and with objects and obtain the mask image. The code below should then load the images and create the final composite.

In [ ]:

```
# Read the images that you produced using Blender.  Modify names as needed.
O = read_image('images/proj4_objects.png')
E = read_image('images/proj4_empty.png')
M = read_image('images/proj4_mask.png')
M = M > 0.5
I = background_image
I = cv2.resize(I, (M.shape[1], M.shape[0]))
```

In [ ]:

```
# TO DO: compute final composite
result = []

plt.figure(figsize=(20,20))
plt.imshow(result)
plt.show()

write_image(result, 'images/outputs/final_composite.png')
```

Bells & Whistles (Extra Points)

Additional Image-Based Lighting Result

Other panoramic transformations

Photographer/tripod removal

Local tonemapping operator

In [ ]:

```
```