Pset 4 Solutions

Contents

Init

```
clear all
close all
clc
drawnow
```

Problem 1

```
load mdr_im
im=imr;
lim = log(im);
```

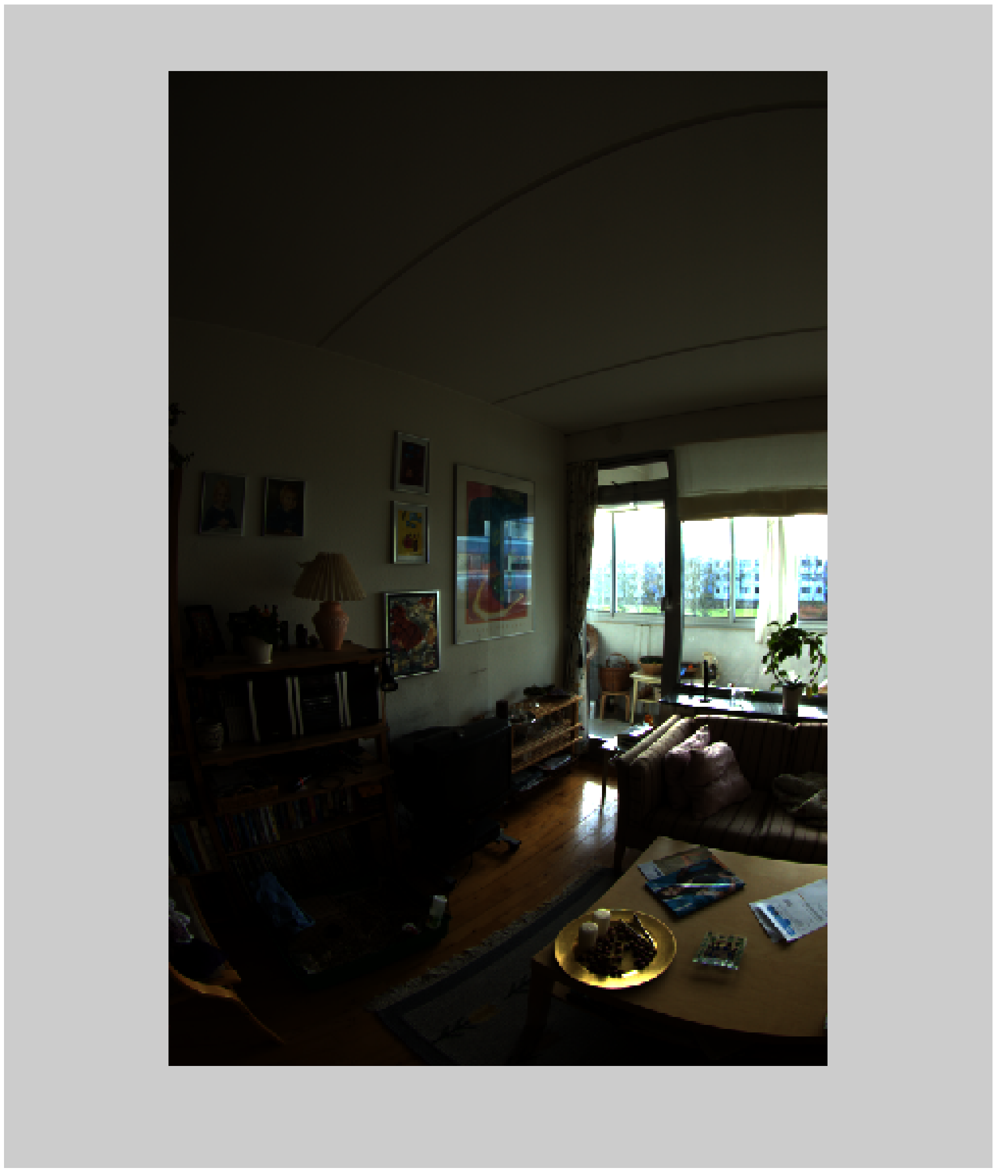
1a

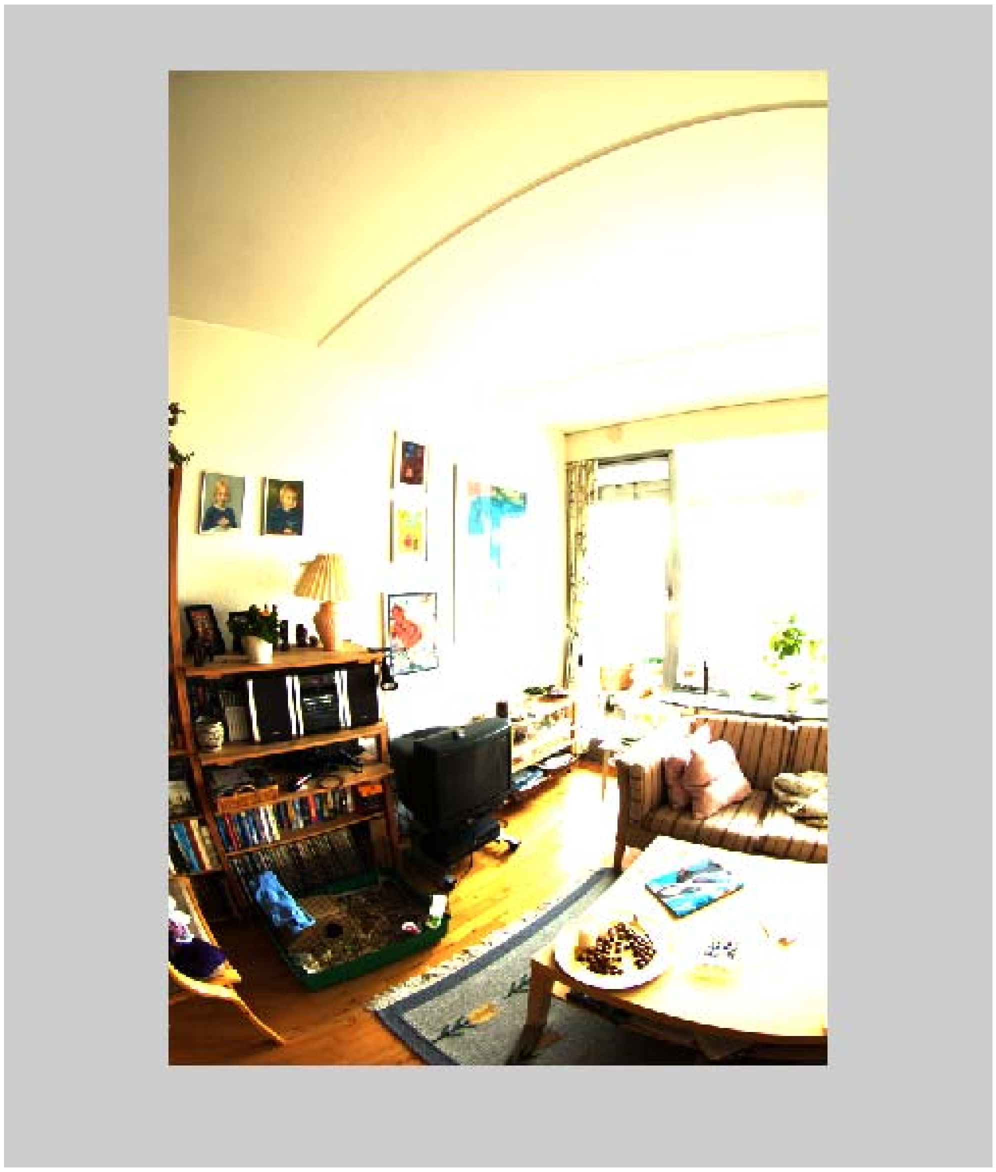
Here the luminance is much greater outside than inside (the sun is much more powerful than any lightbulb, and most of the light inside is an ambient reflection through the window anyway). This, of course, is a very common problem, at least until the sun sets, but our eyes are so good at quickly accommodating that we do not typically notice it.

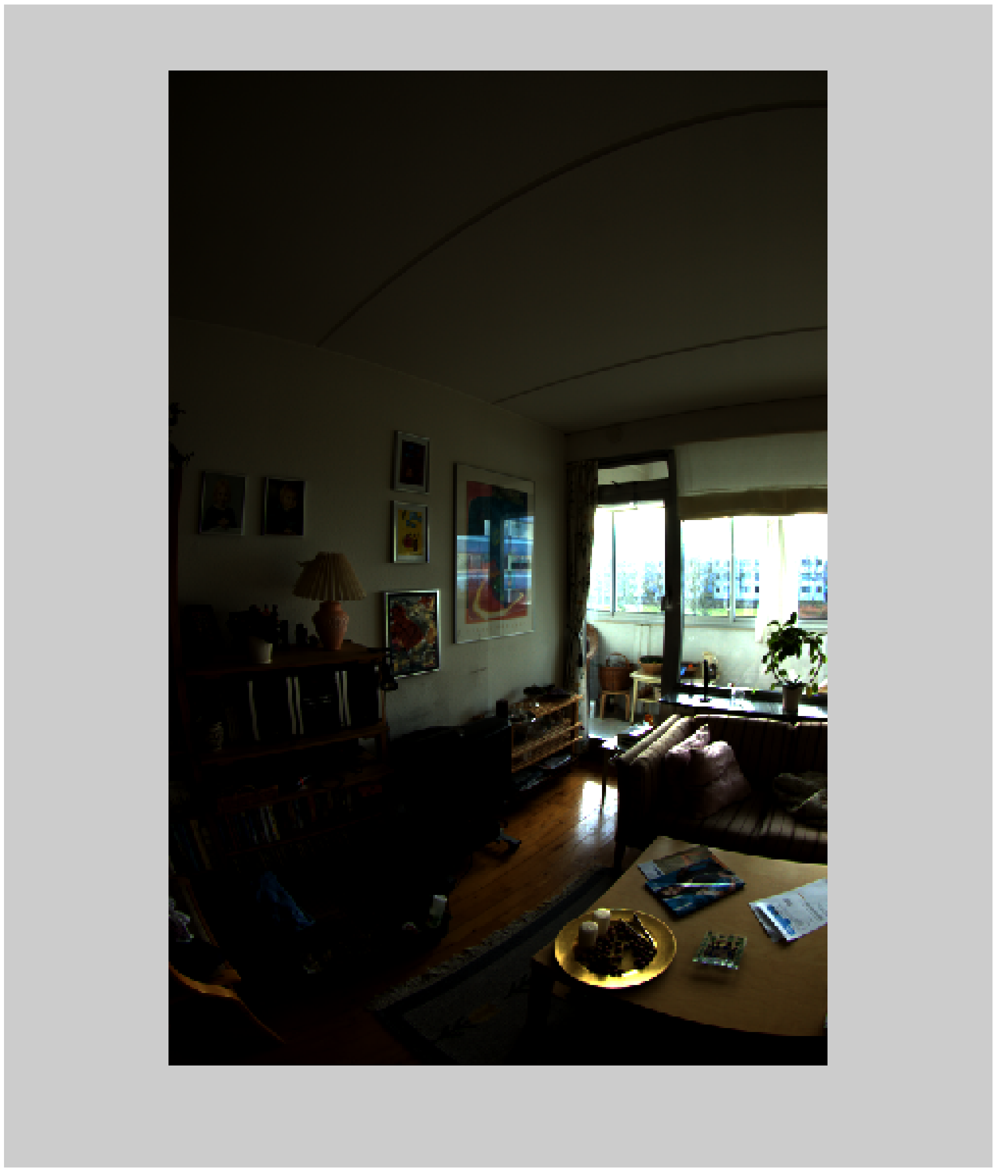
```
figure();
viewim(im);
figure();
viewim(im, [0 .5]);
figure();
viewim(im, [.2 .97]);
drawnow
```

```
Warning: Image is too big to fit on screen; displaying at 67%
```

```
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
```







1b

logR = logL - logI

1c Estimate Illumination

We can (fairly) safely assume that illumination will change gradually across an image, and that any quickly changing features are more likely due to reflectance changes. The Retinex theory solves for reflectance in this way by running an edge-detection algorithm on the image, and then reconstructing the image based on the found edges. This blurs across regions which don't have a clearly defined edge structure (ie, low spatial frequencies). A much simpler, but cruder, way of doing the same thing is by Gaussian blurring the image - high spatial frequencies will be attenuated, and we will be left with an image that approximates the illumination (as opposed to the reflectance, which is the output of Retinex).

1d

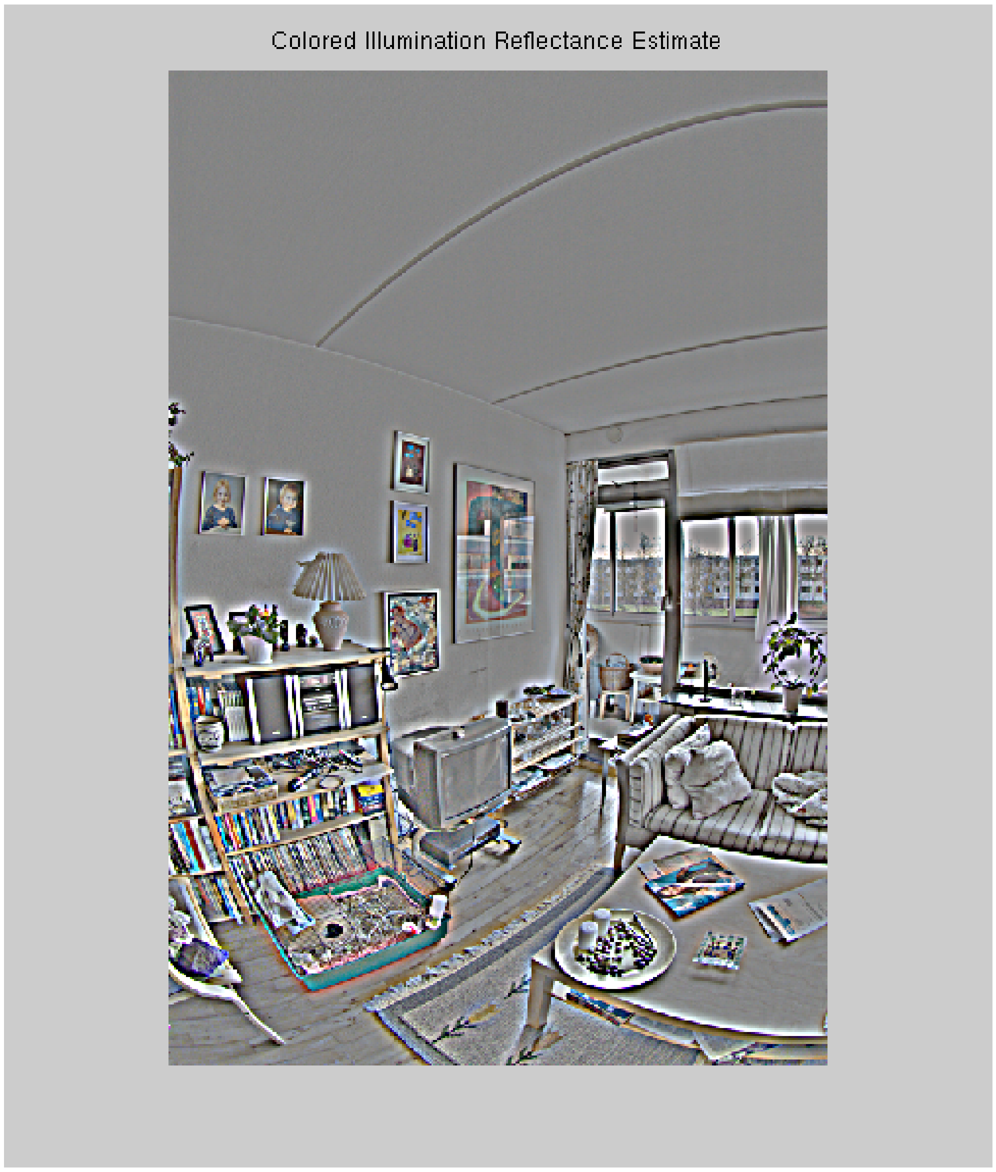
Gaussian bluring gives us the illumination, as described above. However, we run the risk of interpreting any fast changes in illumination as reflectance changes, and this may give us artifacts.

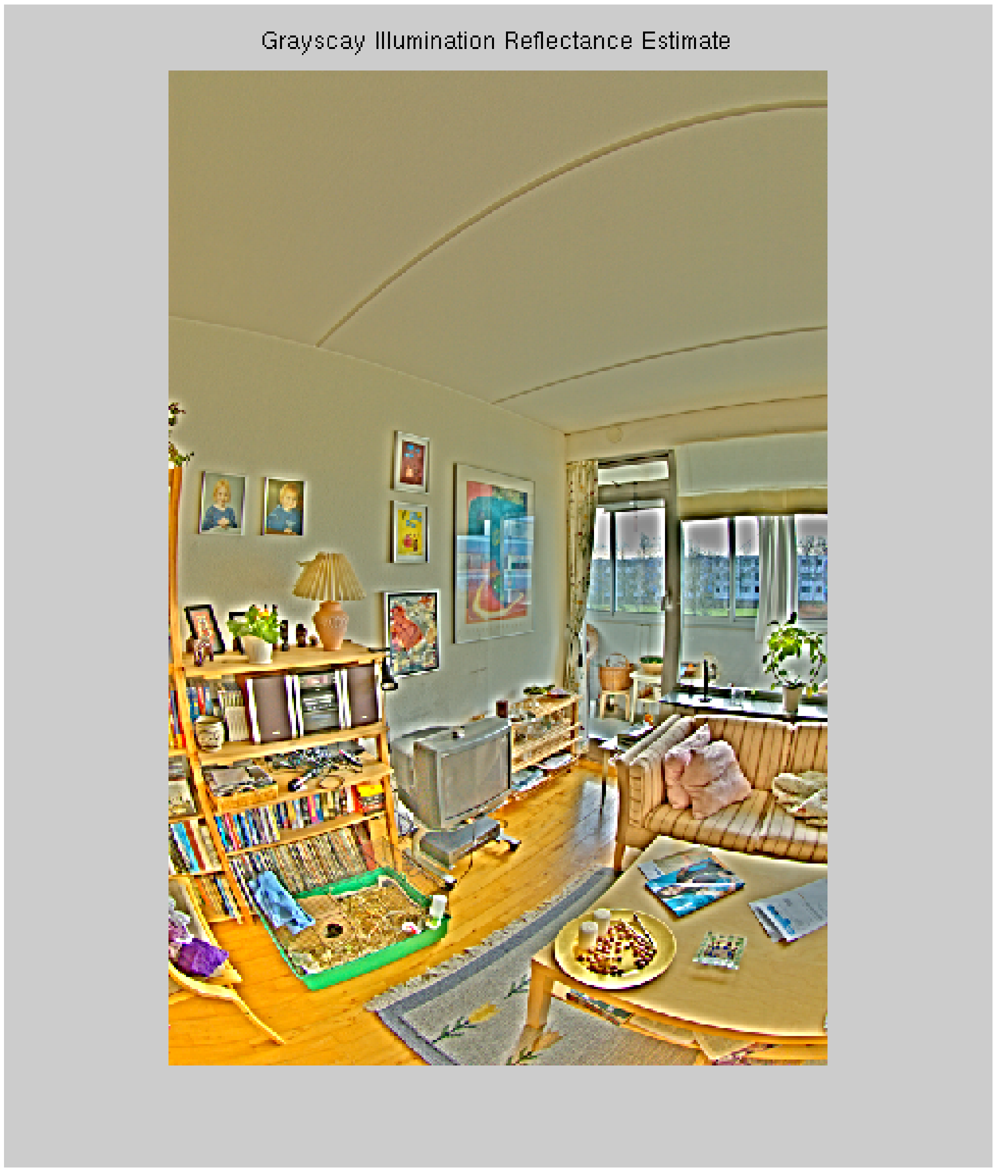
The light in this scene appears fairly monochromatic, in that it doesn't vary in color as much as intensity over the image, so I would estimate illumination as a grayscale. However, doing it in color can be more fun, so I'll do this too!

At this stage, it is very easy to see the artifacts. They are exactly where we would predict - at illumination edges (which are interpreted as reflectance edges) such as around the windowsill, where there is a very fast transition from inside ~low light to outside bright light. Also, this method basically gives a halo effect around any large change, like the pictures and table edge.

```
illum = filterim(lim, 4);
figure
viewim(lim - illum);
title('Colored Illumination Reflectance Estimate')
figure
viewim(lim - repmat((mean(illum, 3)), [1 1 3]));
title('Grayscay Illumination Reflectance Estimate') ;
```

```
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
```


Colored Illumination Reflectance Estimate



1e,f

I found the best scales to use were exponentially growing with a power of 2, weighted over the size of the filter. This produces the most natural and subtle results to my eye, and I prefer this to a truly 'optimized' image with all illumination discounted (but with more artifacts).

NB: the 'flattest' solution will likely be one using a very small scale, and will have the most artifacts.

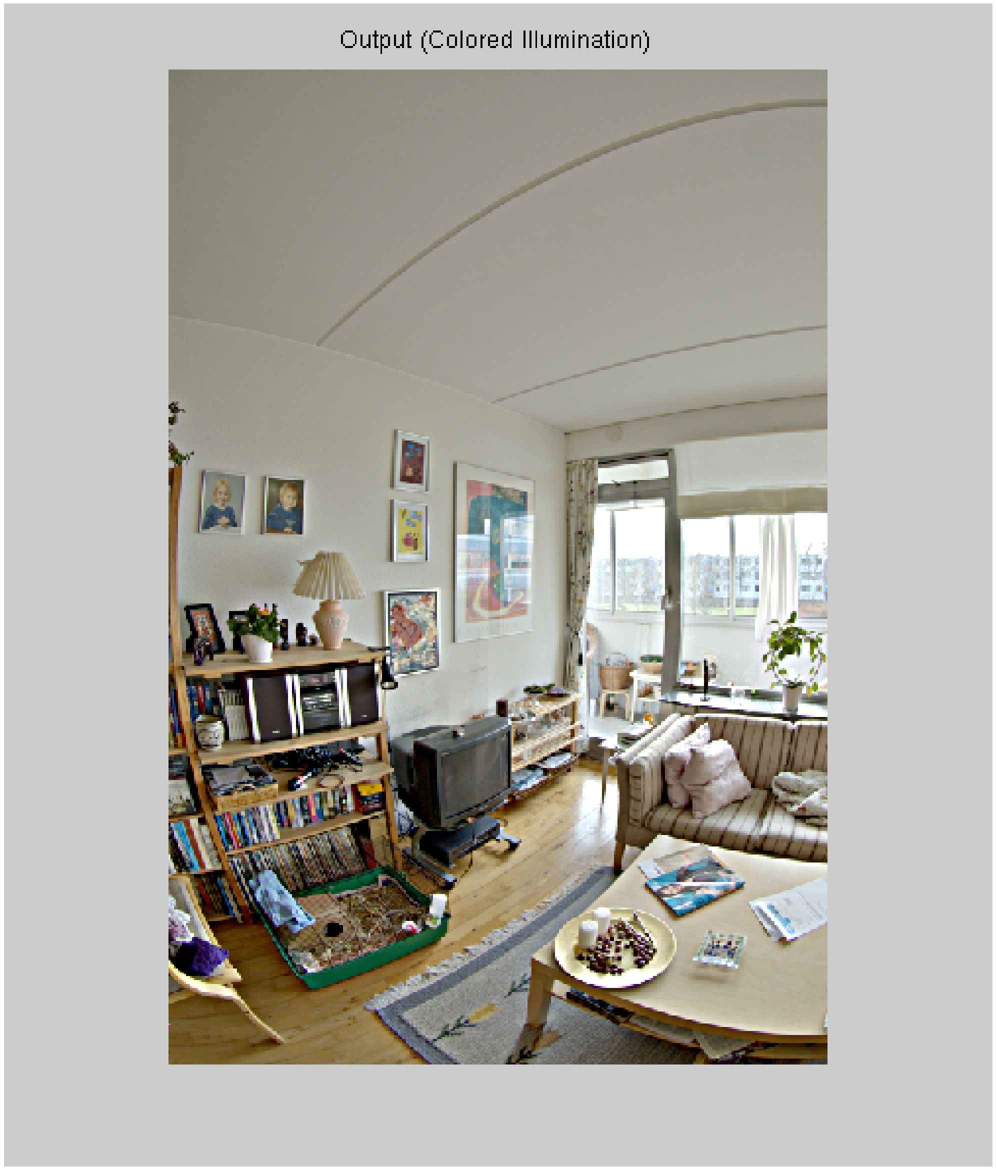
```
scales = 2.^[6:-.25:2];

o_colored = 0;
o_gray = 0;
i_colored = 0;
```

```
i_gray = 0;

for s = scales
    ilum = filterim(lim, s);
    i_colored = i_colored+ilum/s;
    i_gray = i_gray+mean(ilum, 3)/s;

    o_colored = o_colored + (lim-ilum)/s;
    o_gray = o_gray + (lim-repmat(mean(ilum, 3), [1 1 3]))/s;
end

o_colored = o_colored + lim;
o_gray = o_gray + lim;

figure
viewim(o_gray);
title('Output (Gray Illumination)');
figure
viewim(i_gray)
title('Gray Illumination')

figure
viewim(o_colored)
title('Output (Colored Illumination)')
figure
viewim(i_colored)
title('Colored Illumination');

figure
viewim(lim)
title('Log Luminance (Original Image)')
```

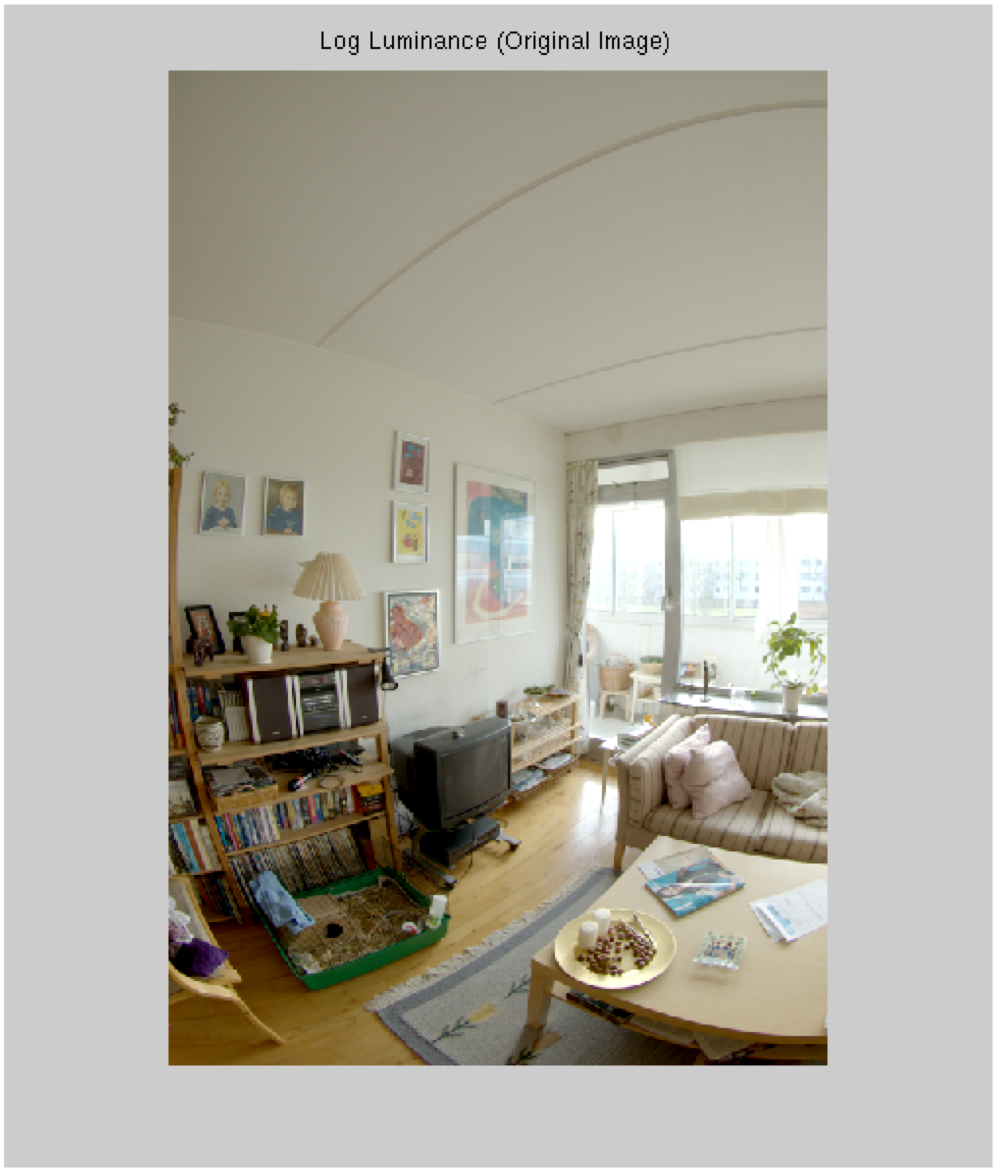
```
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
```

Output (Gray Illumination)

Gray Illumination

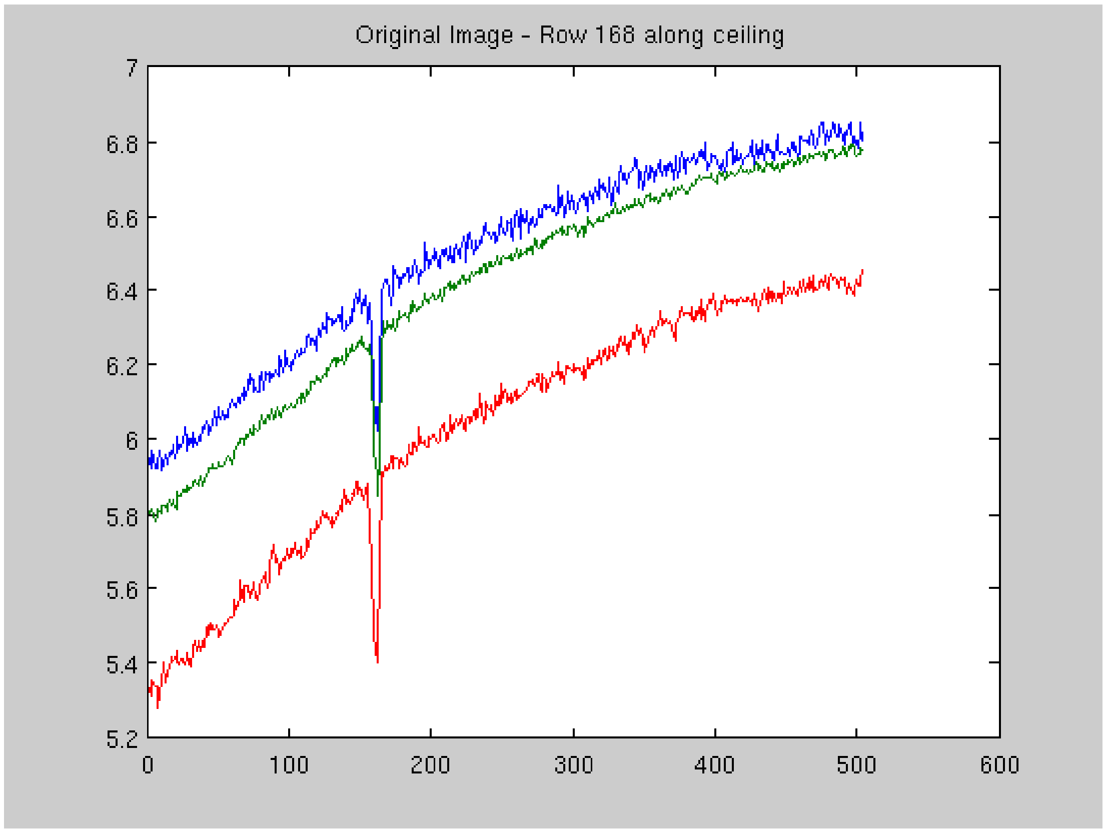

Output (Colored Illumination)

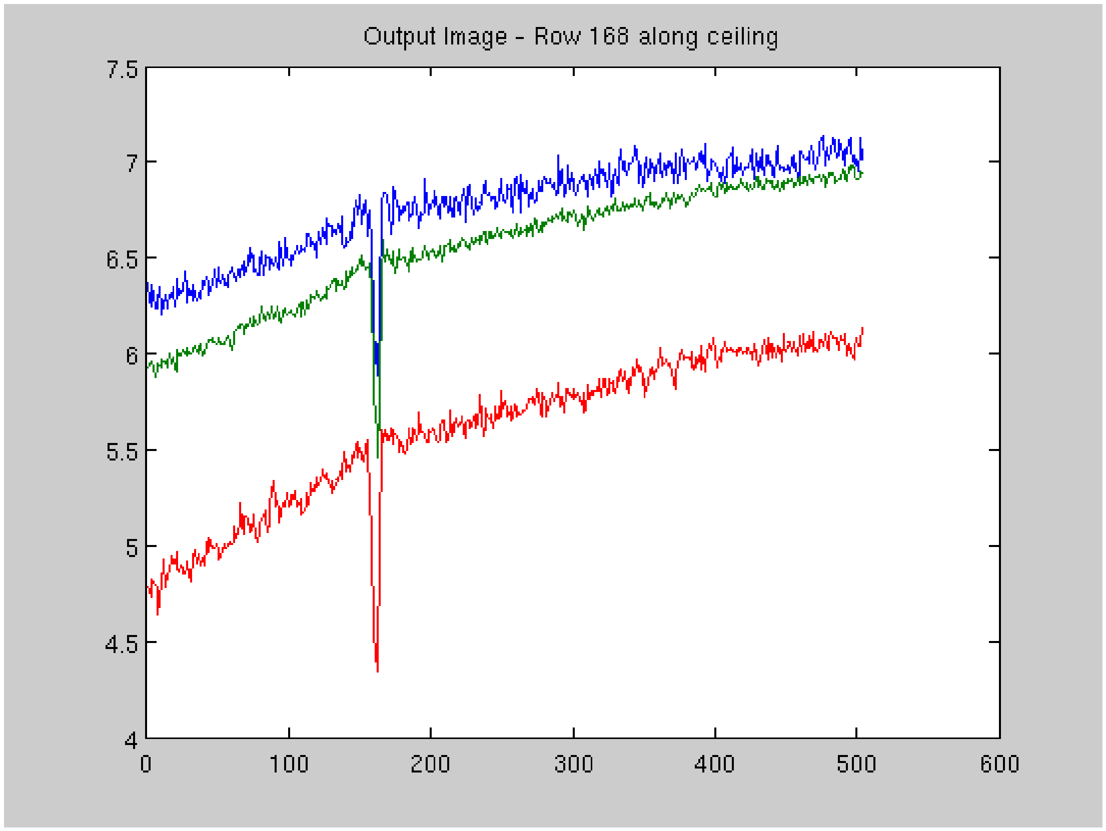
Colored Illumination



Plots the line profiles. Still not perfect, but this does a decent job of flattening the profile even though I did not optimize for this.

```
figure
plot(1:size(lim, 2), squeeze(lim(168, :, :)));
title('Original Image - Row 168 along ceiling')
figure
plot(1:size(lim, 2), squeeze(o_gray(168, :, :)));
title('Output Image - Row 168 along ceiling')
```


Original Image - Row 168 along ceiling
7
6.8
6.6
6.4
6.2
6
5.8
5.6
5.4
5.2
0
100
200
300
400
500
600


Output Image - Row 168 along ceiling
7.5
7
6.5
6
5.5
5
4.5
4
0
100
200
300
400
500
600

1g

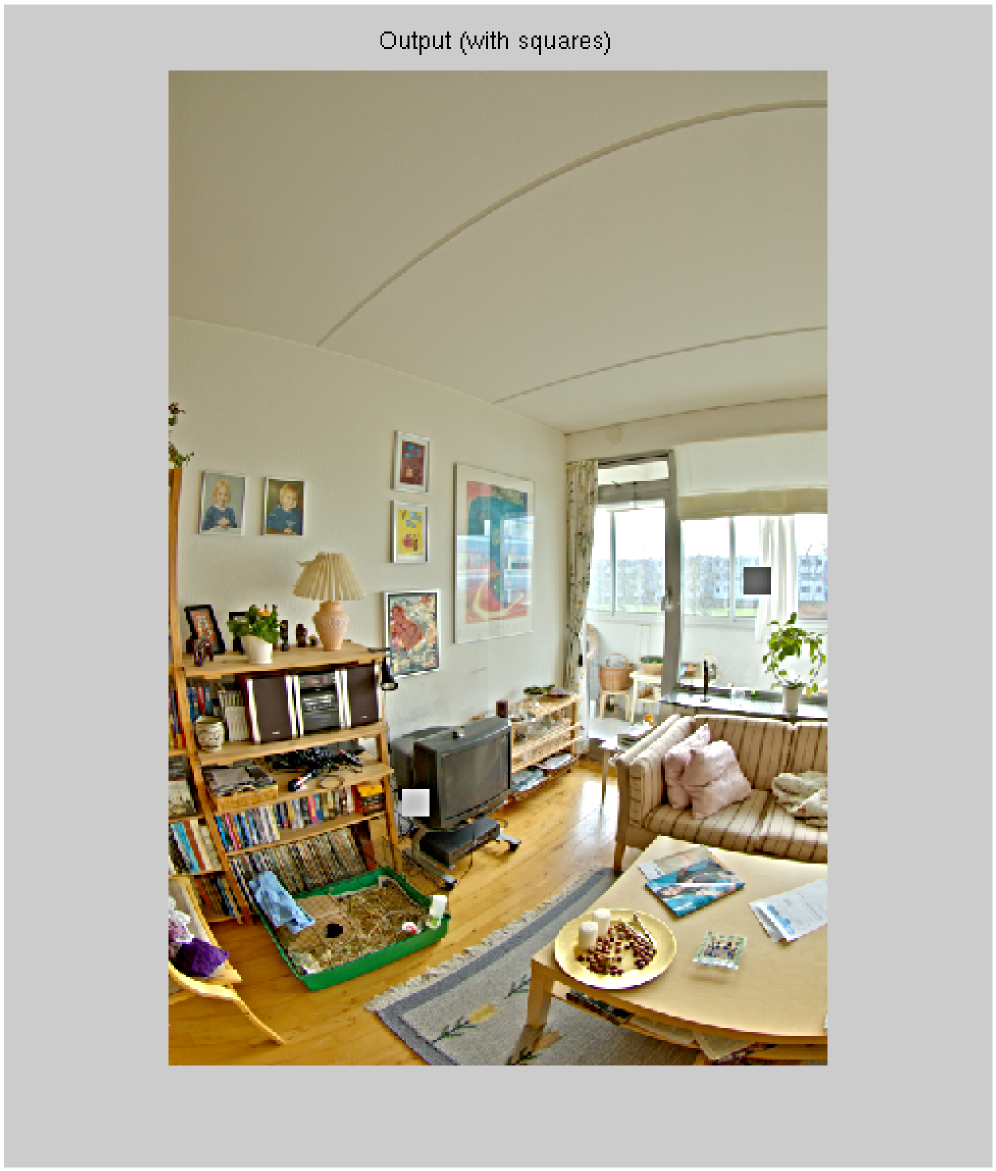
We have essentially solved 2 problems in 1 - by removing the illumination, we have compressed the entire dynamic range, and now the image displays well on our monitors. Of course, we only estimated the illumination, so there will be artifacts where our assumptions used to estimate the illumination are not true. As mentioned above, the assumption that reflectance changes more rapidly than illumination is not true in the region of the window sill, where a very sharp edge is formed with a lot of light on one side, and interior lighting on the other. This produces very ugly dark halos around the window sill, since the algorithm interprets these as very dark objects in the outdoor light.

1h

The squares should no longer be the same luminance, much like the checkerboard illusion(s).

```
scales = 2.^[6:-.25:2];

o_gray = 0;
lim(550:570, 180:200, :) = 6;
lim(380:400, 440:460, :) = 6;
for s = scales
    ilum = filterim(lim, s);
    i_gray = i_gray+mean(ilum, 3)/s;
    o_gray = o_gray + (lim-repmat(mean(ilum, 3), [1 1 3]))/s;
end
o_gray = o_gray + lim;

figure
viewim(o_gray);
title('Output (with squares)');
```

```
Warning: Image is too big to fit on screen; displaying at 67%
```



1i

This should work well on your picture, but will be noisier. The more subtle the manipulation (more scales, varying smoothly) the better. It usually is not as striking however since your picture has a much lower dynamic range.

1j

In theory, classic Retinex will reduce the halo artifacts since it explicitly detects edges. However, it's much harder to implement.

1k

We have simultaneously reduced the dynamic range of light across this scene, and removed a good deal of the illumination from the luminance.