Computer Vision 1: Photometric Stereo

Gabriele Bani
11640758
gabriele.bani@student.uva.nl

Andrii Skliar
11636785
andrii.skliar@student.uva.nl

Introduction

In this homework we apply various algorithms to solve some of the basic problems in computer vision. In Section 1, we explore the photometric stereo algorithm for reconstructing scenes. We study the behavior of the algorithm when presented with different number of images, and try to understand how its behavior is related to the commplexity of the objects in the scene. We also analyze how the shadow trick can help in producing better results and when it is more useful. In section 2 we briefly study some popular color spaces, and give examples on their applications. In section 3, we study how using a few (restrictive) assumptions, we can model the problem of intrinsic image decomposition. We show how to reconstruct an image from its reflectance and shading components, and how changing the reflectance we can obtain different colors for the whole object. Finally, in section 4 we apply the Gray World algorithm to perform color correction, and discuss when it may fail, also briefly summarizing two more complex but more robust color constancy algorithm.

1 Photometric Stereo

1.1 Estimating Albedo and Surface Normal

1. We were expecting to see uniform albedo without any shadow, because illumination should be constant for both of the color regions of the sphere. However, that didn't happen, because to have that result we would need to have images with all possible light directions and 5 is clearly not enough. Due to this you can a bit of shadow closer to the border of the object in fig. 1.

2. In practice, as mentioned in Section 5.4 of [3], at least more than 3 images are used to make least squares solution appropriate. When we use 4 or more images, we get over-determined system of the local surface orientation and albedo (both having 3 degrees of freedom) and that allows us to use the residual of least squares solution in order to determine whether shadowing has occurred.

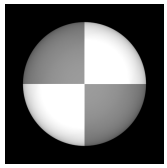
   However, as stated in [5] even three images can be enough. This also supports my point of understanding, that the number of images mostly depend on the complexity of the surface of the object and images that are being used for estimating albedo and surface normal. For example, if we need to estimate these values for flat object, one single image with no visible shadow might be enough. However, for flat objects there is no point in using photometric stereo. Therefore, let's assume that the object is not flat. In that case necessary number of images mostly depend on the complexity of the topology of the object and the material of the object. Let's say, object has a lot of dips and spikes. In that case, number of images needed to estimate albedo and surface normal will be very large, because we would need to consider images in which large number of sources of illumination will be used to have images with all possible shadows.

   So, in general, to have proper least squares solution, we need at least 4 images, but ideally the minimum number of images should be given by a function of the topology of the surface and the material of the object.

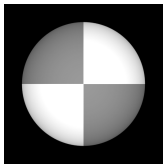
Despite that minimum number of images depends on the surface of the object, note that almost always the more images the better and this assumption is supported by the images that you can see in fig. 1.

3. Shadows play crucial role in photometric stereo as they allow us to correctly estimate albedo and surface normal. Shadows are basically our main workhorse, because photometric stereo is based on the fact that "the amount of light reflected by a surface is dependent on the orientation of the surface in relation to the light source and the observer" as mentioned in [1]. Therefore, because shadows are properties of the light source and the surface of the object being illuminated, they are used directly to reconstruct the surface.

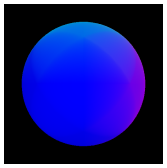
   However, the main issue with using shadows is that when we have only few images, one or the other light can shadow large areas of surface. The problem is that when we are trying to calculate value of the vector $g(x, y)$ at point (x, y) using least squares approach, it will try to account for the points which are shadowed in some images and not shadowed in the others, which means it tries to approximate the outliers. However, when we zero them out using shadow trick, they won't influence our solution anymore. This has a strong impact for a small number of images, (5 in our case), whereas for larger a large number of images (25 in our case), the large number of images will make the solution more robust to outliers and therefore the shadow trick will only make a slight difference. The results of applying the shadow trick can be seen in fig. 1.



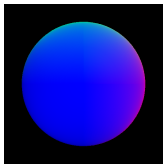
(a) Albedo for 5 images without shadow trick



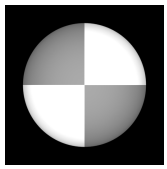
(b) Albedo for 5 images with shadow trick



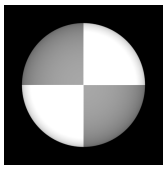
(c) Normals for 5 images without shadow trick



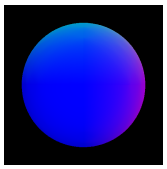
(d) Normals for 5 images with shadow trick



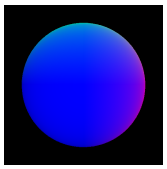
(e) Albedo for 25 images without shadow trick



(f) Albedo for 25 images without shadow trick



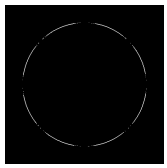
(g) Normals for 25 images without shadow trick



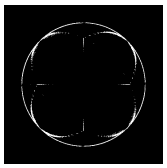
(h) Normals for 25 images with shadow trick

Figure 1: Albedos and normals for 5 and 25 images with and without shadow trick

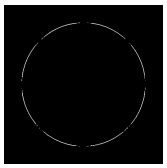
1.2 Test of Integrability



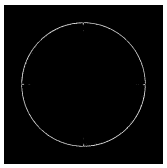
(a) Points, where SE > 0.005 for 5 images (in white)



(b) Points, where SE > 0.00005 for 5 images (in white)



(c) Points, where SE > 0.005 for 25 images (in white)



(d) Points, where SE > 0.00005 for 25 images (in white)

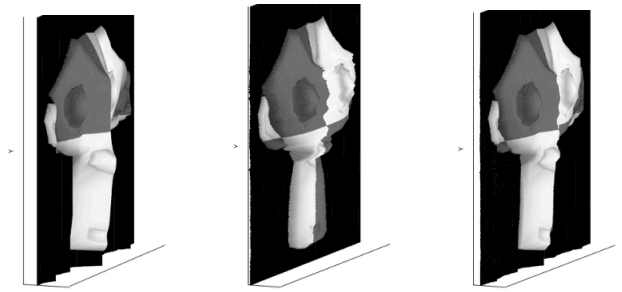
Figure 2: Points, where integrability condition is not satisfied

When we check if Schwarz integrability condition is satisfied, as we can see from fig. 2, the errors occur on the border of the figure, because that is a place where image suddenly changes from sphere surface to the black background. This happens because even despite being discretized in pixels, the color of the image of the sphere changes smoothly and that gives us more or less continuous function. However, at the border it is not continuous anymore and therefore we can assume that Clairaut's theorem is not satisfied and therefore integrability condition can't be satisfied as mixed derivatives are not symmetric.

When we use more images, the number of points where error occurs becomes smaller. This is particularly due to the fact that the estimated normal is closer to the true normal of the object, and therefore the normal changes more smoothly in all directions, which allows Schwarz integrability condition to be satisfied. So, ideally, if we could reconstruct normals perfectly (i.e using images with light source in all possible positions), there will be no errors except for the points on the border.

Taking into account that with a lower threshold it can be seen that error also occurs on the places where algorithm fails to correctly predict true normal of the point because of the low number of images and therefore bad shadowing. However, it makes sense to talk about the error only on the border, because it doesn't depend on the threshold. Even if we make threshold reasonably large, the error will still be there. This error also holds if you use more images. We have used images in fig. 2 to investigate the problem further and images tend to support our assumptions.

1.3 Shape by Integration



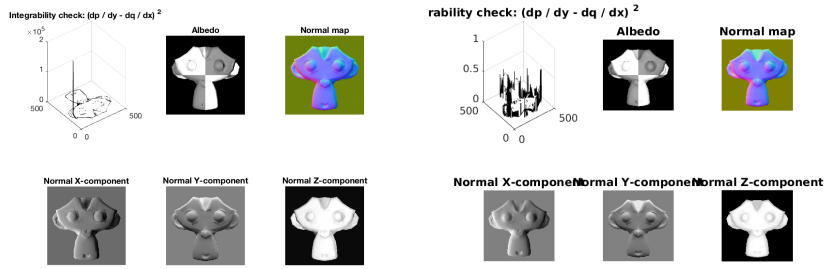
(a) Column-major order (b) Row-major order (c) Average order

Figure 3: Column-major, row-major and average order height map for the grayscale monkey image

1. The difference between column-major and row-major order is the direction, that gets prioritized, as you can see in the image. This also means, that from different points of view different orders show different results (i.e. column-major order gives more realistic results better if you look from the bottom, but row-major order gives more realistic results if you look from the side).
2. As we can see from fig. 3, compared to previous two approaches, average over row-major and column-major approaches performs best, because it averages over both directions and thus smoothes out the final height map making it more similar to real height map. Also, when we use more images, final height map smoothes out as well, so for more images the order doesn't change the result significantly.

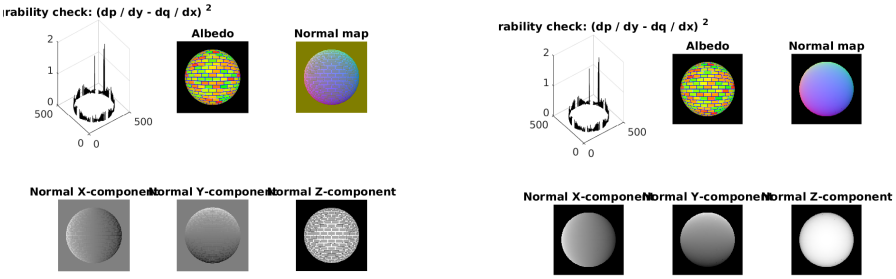
1.4 Experiments with different objects

1. The main error is the same as the error in first task of section 1.1, where we expected the albedo of our object to be constant due to the constant reflectance for both of the color regions of the object. However, because in this case surface of the Monkey is relatively complex, we can't get images with enough directions of the light that would account for all possible shadows and therefore we can still see shadows from eyes and ears in our albedo. If we use less images, as you can see in fig. 4, the result gets even worse, so the solution would be to use more images with sources of illumination located in different points to show all possible shadows that can occur for the given object.

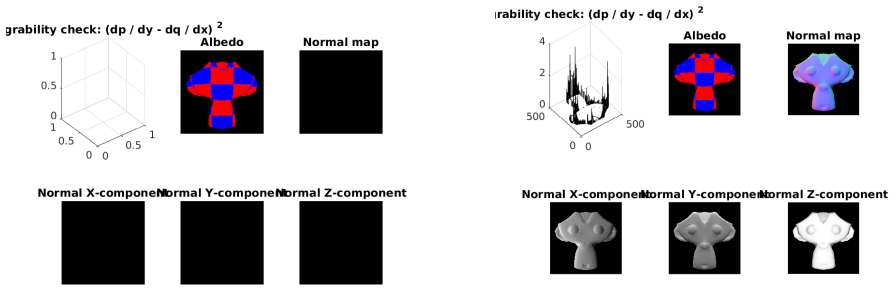


(a) 30 images used (b) All images used

Figure 4: GrayMonkey albedo and normals for different number of images



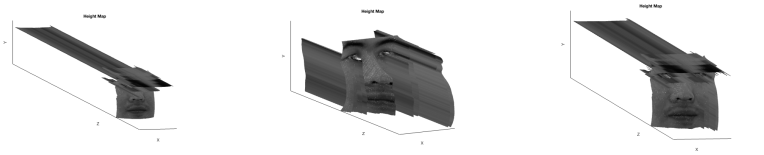
(a) ColorSphere without normalized normals (b) ColorSphere with normalized normals



(c) ColorMonkey with incorrect loader (d) ColorMonkey with normalized normals

Figure 5: ColorSphere and ColorMonkey with and without normalized normals

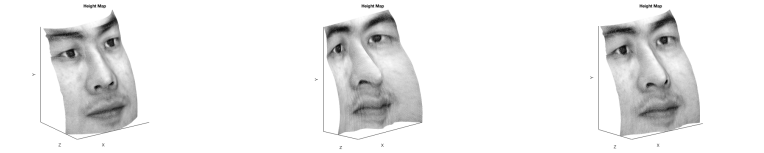
2. We have changed our code to construct albedo and normals in the following way: albedo is calculated separately over each of the three channels and is then combined into a single RGB image; normals are also calculated for each of the channels as well, but then they are averaged and normalized again. There might be several possible issues with that - if we don't normalize our final normals, we get white lines in the places where color changes from one to another (as you can see in fig. 5a). However, normalization fixes that issue and it can be clearly seen from fig. 5a. Another issue is the one with zero pixel, which happens due to the loading process because we normalize images in $image\_stack$ and when min and max value in the channel are the same, we get NaN values for the whole channel and that leads to zeros in the final albedo and normals. This issue can be seen if you compare fig. 5c where it has not been fixed and fig. 5d, where loader has been fixed.
3. From the images fig. 6 we can see, that shadow trick makes height map of the face images very distorted for any order of integration.



(a) Face heightmap with column-major order, shadow trick

(b) Face heightmap with row-major order, shadow trick

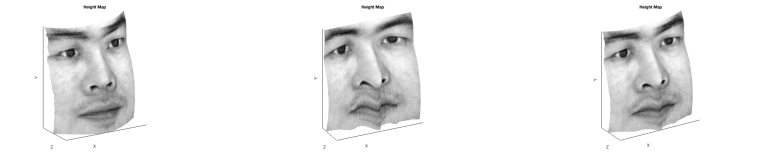
(c) Face heightmap with average-major order, shadow trick



(d) Face heightmap with column-major order, no shadow trick

(e) Face heightmap with row-major order, no shadow trick

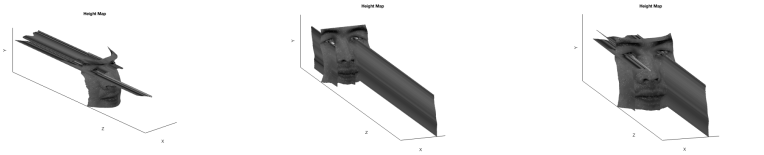
(f) Face heightmap with average-major order, no shadow trick



(g) Face heightmap with column-major order, no shadow trick, filtered images

(h) Face heightmap with row-major order, no shadow trick, filtered images

(i) Face heightmap with average-major order, no shadow trick, filtered images



(j) Face heightmap with column-major order, shadow trick, filtered images

(k) Face heightmap with row-major order, shadow trick, filtered images

(l) Face heightmap with average-major order, shadow trick, filtered images

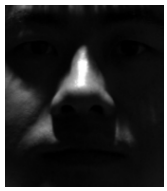
Figure 6: Face heightmap for different configurations

This happens due to multiple violations of shape-from-shading method, all of which can be seen in fig. 7. First of all, some of the images are simply corrupted and have weird lines of light all over the image. Also, some of the facial points don't have Lambertian reflectance (i.e. nose) and therefore do not reflect surface diffusely. Moreover, we can see that some images contain shading from objects, which are not in the image at all (i.e. from hair). Also, in some of the images eyes of the person are moving and therefore this violated the constraint that object should stay constant within all the images and only lighting conditions change.

However, our main finding is that when light source is positioned behind the head (absolute value of its azimuth is larger than $90°$), photos get very weird, because even though light source is behind the head, some parts of the face are still illuminated. This is especially weird if you look at how light changes from images with azimuth $95°$, where the whole image is completely dark to images with azimuth $120°$ and $130°$, where there some parts of the face are illuminated again. Also, we can assume that human face is more or less symmetric and therefore symmetric light sources (i.e. with azimuths $130°$ and -$130°$) would

illuminate face in a similar way. However, if you compare images with 130° and -130°; 120° and -120°, you can clearly see that this assumption doesn't hold. That led us to three possible options why these photos may be wrong: either there is a special box for photo shoot, which reflects some of the light back on the face or light source used changes from photo to photo, or there is no point light source and global illumination is used. Clearly, any of these would violate assumptions for shape-from-shading method.

We tried filtering out all the problematic images and that actually led to better results, which you can see in fig. 6. However, with shadow trick heightmap was still bad and the reason might be that some of the points are black (i.e. eybrows, pupils) in all the images and that makes shadow trick fail.



(a) Shading from hair; Azimuth:0° Elevation:90°

(b) Specular highlights on the nose and cheeks

(c) Corrupted image

(d) Corrupted image

(e) Movement of eyes

(f) Movement of eyes

(g) Azimuth:95° Elevation:0°

(h) Azimuth:-95° Elevation:0°

(i) Azimuth:120° Elevation:0°

(j) Azimuth:-120° Elevation:0°

(k) Azimuth:130° Elevation:20°

(l) Azimuth:-130° Elevation:20°

Figure 7: Shape-from-shadow violations in faces dataset

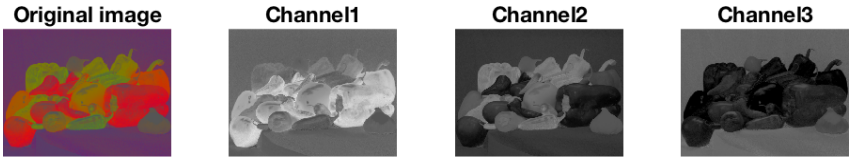
2 Color Spaces

2.1 RGB color model

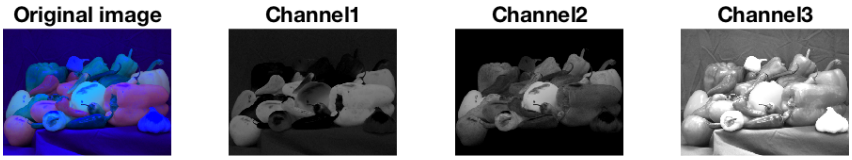
The RGB model is used for different reasons. The main one is that it tries to encode information about color in a way which is biologically plausible, since the eyes perceive color using red, green and blue cones. Also, we use the RGB color model as basis our our digital cameras because the R, G and B values can be calculated easily by using integrals of spectral response function for every sensor over wavelenghts. In formulas for channel c, we have $c = \int L(\lambda)S_c(\lambda)d\lambda$, where $L(\lambda)$ is the incoming spectrum of light and the function S_c is the spectral sensivity of the sensor for channel c.

2.2 Color Space Properties

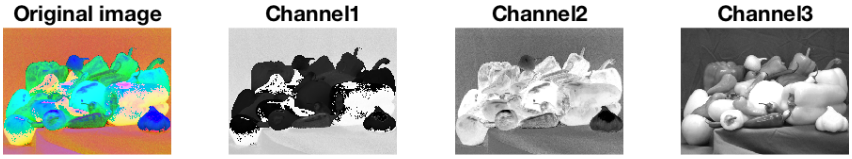
1. The opponent color space (oRGB) is mainly used in computer graphics application. The reason for using it is that like HSV, it allows to easily operate on colors, and also allows a



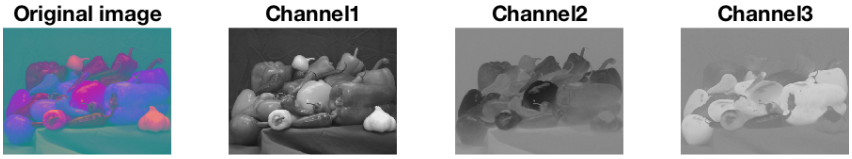
(a) RGB



(b) Opponent color space



(c) HSV



(d) YCbCr

(e) Average (f) Luminosity (g) Lightness

Figure 8: Different color space visualizations

range of other operations. Furthermore, oRGB allows for more computationally efficient calculations than HSV and since it has two color channels, it can support efficient compression with minimal loss of information similarly to the color space CIE L*a*b*.

2. The normalized RGB loses information about light intensity because of the division by $S = R + G + B$. One effect of this is that white and black will become the same grey color. Also, variations of light of the same color will be the same, so for example darker blue can be the same as a lighter blue. Normalized RGB can thus be used when we do not want our algorithm to be sensitive to excessive light or shade.

3. One advantage in using HSV over RGB is that it separates the image intensity from the color information. Some operations such as histogram equalization can be better done when having a separate channel for intensity. It is also much easier for example to filter colors since they are represented with a single continuous variable and not as a mixture of three channels.

4. yCrCb represents luminance information mainly with the Y channel, while the other two channels carry information about color. This can be exploited in video transmission to reduce the bandwidth used with minimal loss in image quality by subsampling only the color channels.

5. The greyscale color space is used when color is not relevant for the task, for example in some cases of edge detection or feature detection. Also, it can used to simplify computations and algorithms, because in greyscale an image can be represented simply as a 2D matrix.

   The greyscale converting methods that we used are all very simple but achieve different results.

   The lightness method creates the image by averaging between the two most prominent colors, and as effect tends to reduce contrast.

   The average method is the most simple, and computes the final image as an average of the three RGB channels. This method suffers however from the fact that shades of gray are not represented in a similar way to how humans perceive luminosity.

   On the opposite, the luminosity method uses a weighted average of the three channels, giving more importance to green, and less to red, and much less to blue. This is done to imitate the way humans perceive light and in fact usually produces the most visually plausible results.

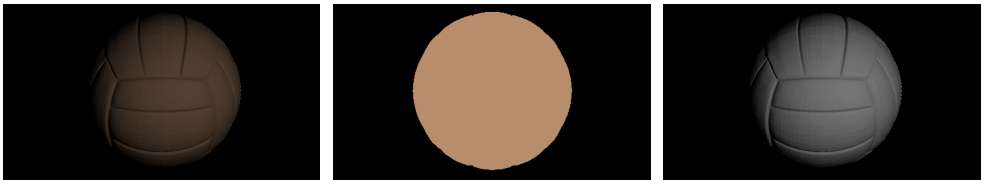
2.3 More on color spaces

The Lab (or L*a*b*) color space defines colors using three dimensions: L, a and b. L stands for lightness, and represents white to black colors. An L value of zero represents white color, while 100 represents black one, and gray colors are in between. The a channel goes from the cyan color at -100, to a magenta color at 100, while the b channel goes from blue at -100 to yellow at 100.

The Lab color space has been created from the CIE 1931 XYZ space with the objective of being more perceptually uniform, so that a change of the same value in the color value should produce a change of about the same visual importance.

One property of the Lab colors is that they are absolute, so they are device independent, which allows to communicate colors across devices. So, it can be used in all applications that involve transmitting a color between two different devices. One example can be matching paint colors to printed media.

3 Intrinsic Image Decomposition



(a) Original image (b) Reflectance component (c) shading component

1. Apart from reflectance and shading, an image can be for instance decomposed into structure and texture. Although the goal of this decomposition is hard to formulate explicitly, the idea is that an image can be thought of as being composed of a structural part, which corresponds to the main large objects in the image, and a texture part, which represents the finegrained details of smaller scale. These details are often considered to be periodical or have an oscillatory nature.
2. The main reason why almost all image decomposition datasets are composed of synthetic images is that it is very difficult to capture real world photos with known ground truth reflectance and illumination. By using synthetic images, we know those ground truths, and can properly train and evaluate different intrinsic image decomposition algorithms.
3. To reconstruct the original image we can multiply its reflectance and shading. The result obtained using the data of *ball.png* can be seen in section 3.

3.1 Recoloring

1. The normalized true material color is $(0.7216, 0.5529, 0.4235)$, which corresponds to the unnormalized color $(184, 141, 108)$. To extract it, since we know that the color is uniform, we take the maximum value for every channel.

(a) From left to right: Original image, magenta recolored image, green recolored image

Figure 9: Recoloring

2. To recolor the ball image, we set to zero the color channels that are not needed. In other words, we just keep the G channel for the green recoloring and the R channel for the magenta recoloring. The recolored images can be seen in fig. 9.
3. The reason is that although the color is uniform, the reconstruction of the image consists in a multiplication with the shading, which is clearly non uniform. This causes the resulting images not to display the pure color and being non uniform.

4 Color Constancy

1. The results of the gray world algorithm can be seen in fig. 10, which clearly shows that grey world algorithm removes the reddish color.

Figure 10: Original image (on the left) and corrected using Gray-world algorithm (on the right)

2. The gray world algorithm assumes that the scene is a neutral gray on average. This can hold when the colors are well distributed in the scene, and the average reflected color is assumed to be the color of the light. The assumption does not hold when colors are not well distributed, for example when many similar colors are present. This can be the case in natural images, in which there are typically just one or two dominant colors (such as blue for the sky or sea, and green for grass). This is the case in the image shown in fig. 10, where we can clearly see that the dominant colors are blue and green. The result of the gray world algorithm is visibly unsatisfactory because of the violation of the grey world assumption.
3. The ACE algorithm [4] is inspired by the adaptation mechanisms of the human visual system, and is in fact able to adapt to widely varying lighting conditions. The algorithm takes into the account the spatial distribution of color information as can be seen from the formulas below. The first step of the algorithm is called chromatic/spatial adjustment, and produces an image R, in which every pixel is recomputed according to the image content. The image R_c, is calculated for every channel c.

$$R_c(p) = \sum_{j \in S, j \neq p} \frac{r(I_c(p) - I_c(j))}{d(p,j)}$$

where d is a distance function (such as euclidean or Manhattan) used to weigh the amount of global or local contribution. The function r instead is typically taken to be a step function

or a saturated linear function, and is meant to imitate the brain lateral inhibition mechanism of neurons. S is a subset of the image, and can be chosen to account more or less about local or global information. The second step of the algorithm is simply a projection of the intermediate images R_c in the range $[0, 255]$, and is typically chosen to be linear.

Another algorithm that can be used for color constancy is the gamut mapping algorithm [2], which is a pixel based color constancy algorithm. This uses the assumption that in real world images, only a limited number of colors can be observed for a given illuminant. This means that any unexpected variations of the colors of an image is caused by a deviation in the color of the light source. The algorithm relies on a canonical gamut, which is the set of colors that can occur under a given illuminat, and is learned from a training set. When presented with a new input image, its input gamut is constructed. Once the input gamut is obtained, this is used together with the canonical gamut to find a set of feasible mappings that map the input gamut into the canonical gamut. Then, one mapping is selected from the set of feasible mappings using an estimator. The selected mapping is applied to the input image to estimate the unknown light source.

5 Conclusion

In this homework, we have tried to solve some important computer vision problems using various algorithms, and argued their strenghts and their applicability in the real world. Using the Photometric stereo algorithm, we managed to estimate surface normal and albedo of objects. We have argued why although only three images of the same objects under different illumination are theoretically needed, a large number of images can be needed for complex objects. We have also explored how the shadow trick can help generating better results. We have understood how different color spaces work, and when they are more useful since they can represent some of the light information in a more explicit way. Images can be decomposed in many components, and we explored how to combine shading and reflectance to manipulate images, also briefly describing other possible image decompositions. We were also able to succesfully apply the Gray World algorithm for color correction, and argued why in practice it can fail in the real world, briefly analyzing two more complex alternatives.

References

[1] Wikipedia contributors. Photometric stereo — wikipedia, the free encyclopedia, 2017. [Online; accessed 19-February-2018].

[2] Graham Finlayson and Steven Hordley. Selection for gamut mapping colour constancy. *Image and Vision Computing*, 17(8):597 – 604, 1999.

[3] David A. Forsyth and Jean Ponce. *Computer Vision A Modern Approach*. Prentice Hall, 2003.

[4] Carlo Gatta, Alessandro Rizzi, and Daniele Marini. Ace: An automatic color equalization algorithm. 01 2002.

[5] C. Hernández, G. Vogiatzis, and R. Cipolla. Overcoming shadows in 3-source photometric stereo. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 33:419–426, 09 2010.