Assignment 1: Photometric Stereo & Color

Davide Belli
11887532
davide.belli@student.uva.nl

Gabriele Cesa
11887524
gabriele.cesa@student.uva.nl

Introduction

In this first assignment, we are going to experiment and compare techniques for Photometric and Color analysis in digital images. In particular, we are going to estimate albedo and surface normals on different objects, both colored and gray-scaled. Afterwards, we are computing the integrability check to spot computational errors, and then generating height maps with column-major, row-major and average methods. The rest of the assignment focuses on color in images. First, we experiment and analyze different Color Spaces with their advantages and drawbacks. Then, we consider Intrinsic Image Decomposition with Reflectance and Shading components, trying to reconstruct and recolor the original image, and proposing different Decomposition components. Finally, we consider the Color Constancy problem which can be solved employing methods such Gray-World Algorithm, and we discuss different ways to tackle this task. All output images can be found in the compressed *outputs.zip* in each Task subdirectory.

1 Photometric Stereo

1.1 Estimating Albedo and Surface Normal

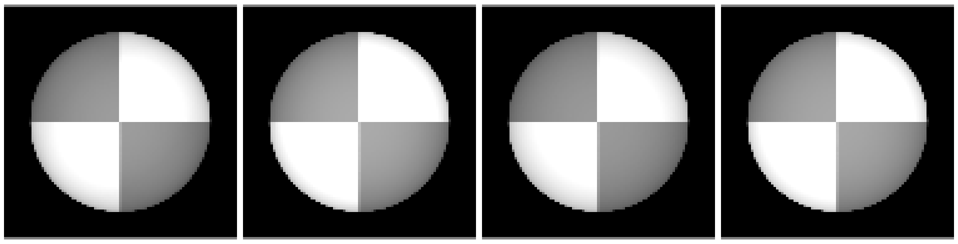

(a) 5 images (b) 25 images (c) 5 img + shadow trick (d) 25img + shadow trick

Figure 1: Albedo of Sphere Gray computed using 5 and 25 images and both with and without Shadow Trick

1.1.1

In Fig 1a and Fig 1c, the albedo compute using the 5 images from *SphereGray5* is shown (both with and without the Shadow Trick).

We expected the resulting albedo to be uniform since the reflectance ρ should be constant in the 2 color regions of the sphere. However, 5 images are not enough to obtain a uniform result and a shadow appears around the sphere.

1.1.2

Since the albedo and the normal are computed through Least Squares method, it is reasonable to use more than 3 images: since we have 3 unknowns (the 3-dimensional vector **g**), having only 3 constraints (i.e. 3 images) would make the system of equations completely determined.

Using more images, the resulting albedo becomes more uniform. Indeed, Fig 1b and Fig 1d show a smaller shadow around the sphere with respect to the albedo computed with only 5 images.

1.1.3

If a pixel is completely covered in shadow in one of the image, the resulting equation (from this image) may affect the least squares result. Indeed, in that case, the values of $I(x, y)$ does not depend on $\mathbf{g}(x, y)$ (since $I(x, y) = 0$). Therefore, it is an outlier with respect to the points laying on the least squares line.

The effect of these outliers is smaller as the number of images increases (i.e. with a large dataset), as many more points will lay on the line.

When the number of images is not big enough, there is a trick to deal with this problem. Multiplying both of the sides of the least squares equations for the value of the pixel, the equations corresponding to the shadowed images will degenerate to the identity $0 = 0$ and, so, will not affect the result.

As Fig 1 shows, the shadow trick seems to be very useful when using only 5 images. However, with the 25 images dataset it does not lead to relevant improvements.

1.2 Test of Integrability

1.2.1

Performing the test of integrability, the error is meaningful only on the edges of the sphere, i.e. where there is a sudden change in the image from the sphere surface to the black background. A possible explanation is that this high difference makes the discrete approximation of the derivatives faulty.

Using more images seems to reduce the number of points where the error is high. Particularly, we see less errors in the regions corresponding to the angles of the light sources. Ideally, having images with light sources from every angle would eliminate these errors.

1.3 Shape by Integration

1.3.1

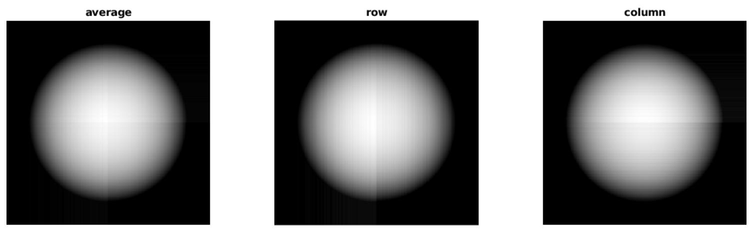
As Fig 2 shows, row path and column path lead to slightly different results. Particularly, when using only 5 images for the gray sphere (Fig 2a), the column path makes the surface appears squeezed from the top and bottom while the row path from the sides. Though it might not be emerge clearly by looking at these images side by side, the difference is noticeable when overlapping the figures (see output folder for higher resolution images).

A great difference can be seen using the Gray Monkeys images, too. In that case, it seems that the column path leads to a rather good surface. Conversely, the row path seems to not decrease enough the value after "building" the upper part of the head (which should have high values): indeed, both the nose/mouth and the background under the ears should have lower heights but keeps similar values as the pixels over them.

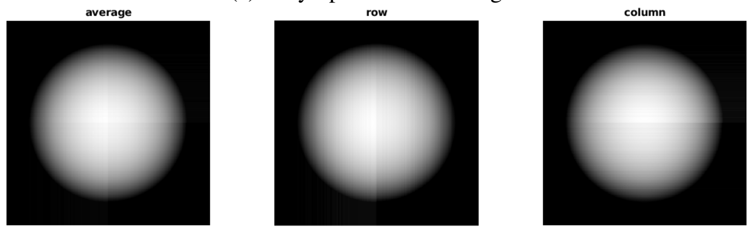
1.3.2

Using the average seems to improve results when using only 5 images for the gray sphere (Fig 2a), since the errors of the column and row paths compensate each others.

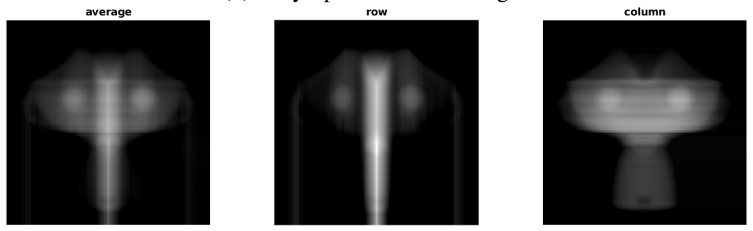
Even better results are achieved using the dataset with 25 images (Fig 2b). In that case, the differences between paths are smaller.



(a) Gray Sphere with 5 images



(b) Gray Sphere with 25 images



(c) Gray Monkey with 5 images

Figure 2: Height maps using row and columns path and their average (the gray-scale color represents the height of each pixel normalized with respect to the maximum height value among all the pixels in all the paths)

1.4 Experiments with Different Objects

1.4.1

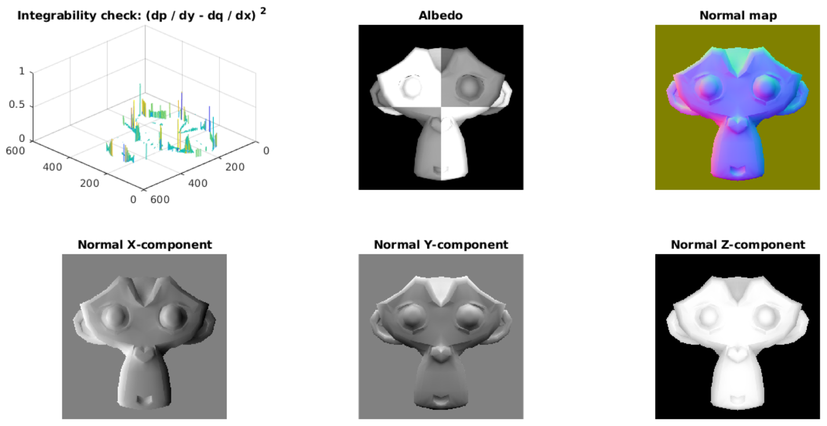


Figure 3: Results with the MonkeyGray dataset

The results with the MonkeyGray dataset can be seen at Fig 3. Though this dataset contains 121 images, the resulting albedo still contains many shadows corresponding to the shape of the monkey. A reason could be the fact that the shape of the monkey is much more complex: many parts of the surface are shadowed by other parts of the face (for example, the grooves around the eyes are almost always shadowed, either by the eyes or by the eyelids. Another example is the inner border of the ears).

Removing images seems to worsen the results as less data-points where the problematic pixels are lighted would be available.

1.4.2

To make our code work with 3-channel RGB inputs we assumed that the light source contains green, red and blue lights equally. Moreover, we know that the light source is always one (same for the 3 colors). As a result, every point on the surface should reflect the light in the same way, independently from the light color. For this reason, we computed albedo and normals vector for each channel and then we took the average of the resulting normal vectors to estimate the actual normal vectors. Conversely, we preserved each channel's albedo and used them as RGB channels for plotting the albedo.

When loading the images in the images stack we found a problem to be solved. In the grayscale case we were normalizing the values in the range [0, 1], and, similarly, we normalized each channel independently in the colored images. However, in the MonkeyColor dataset, the Green channel is zero everywhere making the normalization faulty (dividing each value by zero). Therefore, we kept the Green channel to zero.

Another problem was that the average of three normal vectors is not necessarily a normal vector. As a result, when plotting the normal components, the pixel colors were not meaningful. Normalizing the average vectors solved this problem.

Finally, the problem with completely black pixels explained before could still arise. When the cause of a black pixel is a shadow, the shadow trick can be used again, applying it to to each channel. Anyway, since the dataset provided was large enough, the shadow trick was not necessary. However, when a pixel is completely black independently of the position of the light (as in the case of the background or the whole green channel in the Color Monkey dataset), the shadow trick is useless. For example, let's consider the Color Monkey dataset. When a pixel is always completely black in a channel, its albedo and the components of the normal (for this channel) are set to zero. Therefore, when we estimate the normals vector through the average of the three channels, the resulting vectors for a pixel will be $\overline{\boldsymbol{n}} = \frac{\boldsymbol{n_r}+\boldsymbol{n_g}+\boldsymbol{n_b}}{\boldsymbol{3}} = \frac{\boldsymbol{n_r}+\boldsymbol{n_b}}{\boldsymbol{2}}$, i.e. the resulting vector will be proportional to the mean of only the red and blue channels. If, then, we normalize this vector, the result will not be affected anymore by the green channel.

1.4.3

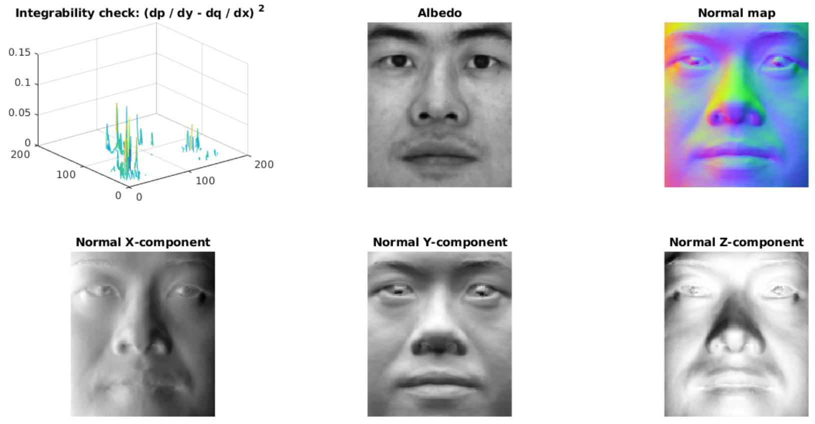


Figure 4: Results with the Faces dataset without the shadow trick

In Fig 5 the results using the Yale Face Database are shown.

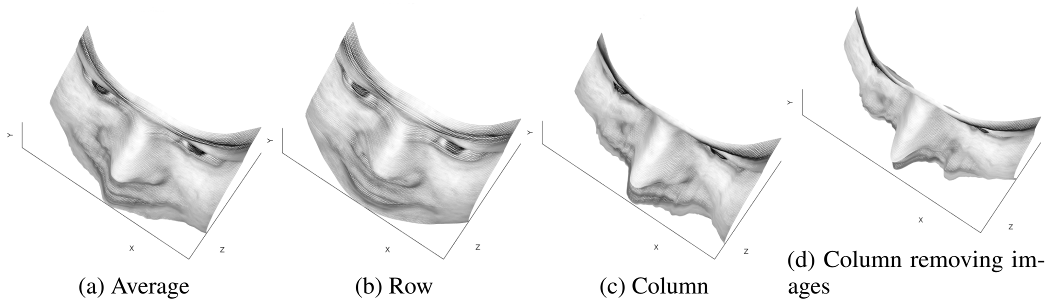


Figure 5: Height Map of the Yale Face Database without the shadow trick

Figs 5a, 5b and 5c show the height maps obtained with the different integration paths without the shadow trick. Notably, the row path leads to a "swollen" face with respect to the column path. Indeed, in the first case the face seems to have a more uniform height, whereas in the second the nose and the left cheek (on the right side of the image) have higher values than the rest of the image and the face results slightly deformed.

Enabling the shadow trick leads to even worse results with faces completely deformed. The problem is that some part of the image (like the pupils or the eyebrows) are almost always black, independently from the position of the light (since they are black). Since the shadow trick "removes" the contribution of images where a pixel is completely black from the least squares algorithm, in these regions the systems of equations will probably be undetermined.

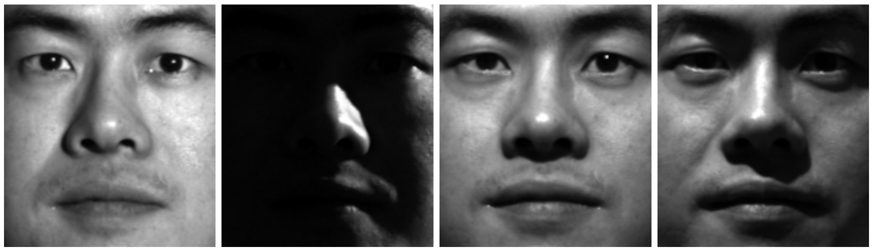
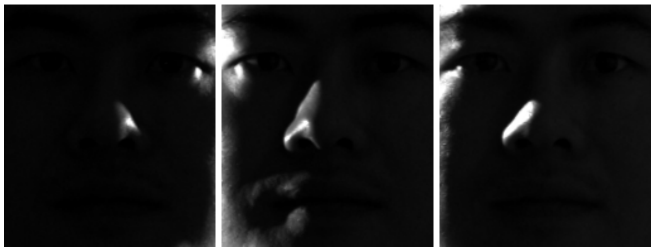


Figure 6: Examples of images from the Yale Face Dataset containing specular highlights

However, even without the shadow trick, the shape-from-shading method is not very accurate (see again the height maps without the shadow trick). Indeed, the shape-from-shading methods assume the surface to be a Lambertian surface illuminated by a distant point source, i.e. only the diffuse reflection is taken into account. This means that specular highlights coming from specular reflections violate this assumption and can't be modeled. However, in real images like the ones in the Yale Face Dataset, this effect is present. The result is that sometimes this highlights are interpreted as part of the albedo of the image. Indeed, their effect can be seen in the computed albedo shown in Fig 4 on the nose or on the cheekbones. Some examples of images from the dataset showing this effect can be seen in Fig 6.



(a) Azimuth: -120° Elevation: 0° (b) Azimuth: 120° Elevation: 0° (c) Azimuth: 130° Elevation: 20°

Figure 7: Examples of images from the Yale Face Dataset with light coming from behind

Finally, we found some of the photo where take from behind the head. Looking at the azimuth coordinates, some of the images where taken from an angle grater than $90°$ or lower than $-90°$, which means the camera was beyond the sides of the head. However, in some of these images the face was still illuminated: though some light could still reach the end of the corresponding side of the face, the whole face still is slightly lighted. This means that this light probably did not reach the face directly but after reflections (for example, reflecting against the walls or other objects around). Some examples are shown in Fig 7. Removing all those images whose absolute azimuth was greater than $90°$ led to slightly better results, particularly for the height map computed through the column path (compare Fig 5c with Fig 5d).

2 Color Spaces

2.1 RGB Color Model

RGB is the most used color model in digital systems for two main reasons. First, this model is based on the biological way in which human sight works. Cone cells, one of the two types of light-sensitive receptors, are responsible of generating electrical signales that are interpreted in the brain to allow color vision. There are three types of cones which are most sensitive to wavelengths corresponding to red, green and blue colors. By representing digital images as a mixture of this 3 colors, we can therefore simulate human sight. The second reason is that it is convenient to generate colors in an additive way. Pixels in cameras and photographies are composed by three different colored lights. By setting them to different intensity values in the range from 0 to 255, over 16 millions of color combination can be produced. This additive color space results in white light when all of the pixel lights are at maximum intensity and that is why RGB is an additive color model. In general, digital cameras capture the RGB images by representing the objective as a matrix of colored points and breaking them up in the three main colors. Every pixel in the image is therefore stored in 3 bytes of information.

2.2 Color Space Conversion

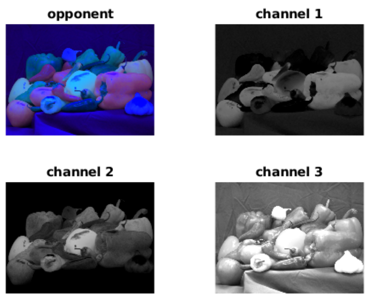
Here we plot images converted in different Color Spaces: Opponent Color Space (Fig. 8a), Normalized RGB (Fig. 8b), HSV Color Space (Fig. 8c), YCbCr Color Space (Fig. 8d), Greyscale Color Spaces (Fig. 9).

2.3 Color Spaces Properties

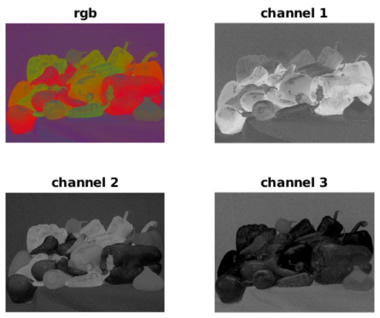
Opponent Color Space Opponent color space is based on Hering's Opponent Process Color Theory. This theory explains how it is natural to describe images as combination of four main colors: Green, Red, Yellow, Blue. Also, we don't have experience of colors that are both yellow and blue or red and green, while we have colors half-way between other combinations (such as orange or purple). Those two *opposite* colors result in axis on which we can perceive images' colors. As a result, the Opponent Color Space is built over three channels: one luma (white and black) and two chroma (four main colors on the specified opposite axis). This representation is suitable to compute, select and manipulate colors in digital images, and allows qualitative metrics such as cool-to-warm classification. This is the task in which OCS outperforms HSV, which is still designed for computer graphics, but not optimized for color transfer. [1]

Normalized RGB Normalized RGB consists in dividing the values in the three channels by the sum over those values. As a result, the colors are represented by the proportion of red, green and blue, and no more by their actual intensity. This method aims to remove distortions in regions of the images caused by shadows or lights. In particular, over-exposed region of the picture will have higher values for RGB channel, and the opposite is true for parts in shadow. After applying the normalization, colors in objects will appear to be much more homogeneusly, lacking in depth. On the otherside, all region in which the RGB channels had the same values (for example black and white parts) will become indistiguishable, and often times colors will no longer appear natural.

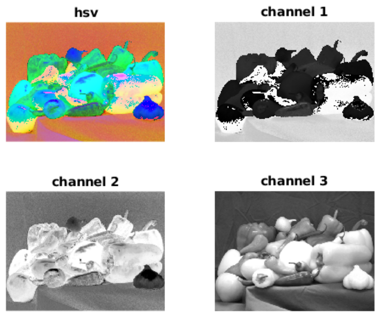
[1] `https://graphics.stanford.edu/~boulos/papers/orgb_sig.pdf`



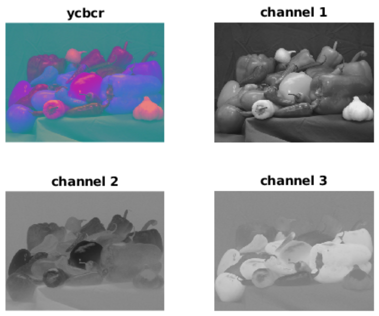
(a) OCS and its 3 channels



(b) rgb and its 3 channels



(c) HSV and its 3 channels



(d) YCbCr and its 3 channels

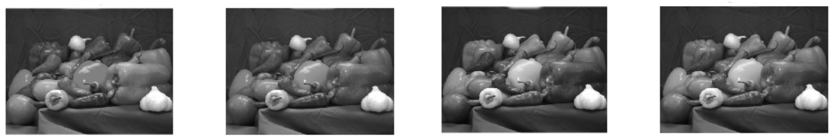


Figure 9: Grayscale techniques: Lightness, Average, Luminosity, MATLAB rgb2gray

HSV Color Space This Color Space was initially designed for computer graphics purposes. Instead of representing images based on three main colors, HSV focuses on three different aspects: Hue (the color), Saturation and Value (Brightness). This representation is widely used in color selection tool in various software, but also employed for more complex tasks such as image segmentation and object recognition. Disadvantages of this model include the fact that hue and lightness might be confounded. For example, we may have blue and yellow parts of an image having the same brightness value, while the actual perception results in the latter been much brighter than the former.

YCbCr Color Space In this model, the Y is the luminance component, while Cb and Cr are respectively the blue-difference and red-difference chroma components. This Color Space is particularly useful to separate the luminance signal from the chroma. The separation can be used to compress information (Chroma Subsampling), by reducing the chromatic aspect of an image, leveraging on the fact that human sight is more sensible to black and white differences.

Greyscale Color Space This color space is the only one considered in this task that do not include information about colors, reducing the number of channels to one and greatly compressing the memory usage. There are different ways to achieve this result, for example by averaging over channels (average method), by taking half of the difference between the higher and lower channel (lightness method), or by considering a linear combination which roughly reflects the proportion of cones per color in human eyes (luminosity method and built-in MATLAB function). As explained in Chroma Subsampling compression, this idea leverage on the fact that vision is more stimulated by light-shadow information (captured by rods) than by colors. Colors may also be not very meaningful in tasks such as edge detection or textual representation. The obvious disadvantage is the loss of relevant information about colors.

2.4 More on Color Spaces

The **CMYK Color Space** is a subtractive model particularly used in color printing. Opposite to additive models, CMYK describes how to mask colors on a lighter background (such as common printing paper), to generate colored images. CMYK stands for the printing colors Cyan, Magent and Yellow, while the last letter represents the Key of the Black key plate. This Color Space can be opposed to the ones used to represent computer images. This is because digital pictures are generated by lights specifically colored to result in different colors, starting from a black screen, while subtractive models start from a white background and subtract colors to generate other ones.

3 Intrinsic Image Decomposition

3.1 Other Intrinsic Components

An alternative way to the one implemented for decomposing an image in components is to use the method proposed by Evangelopoulos and Marangos [2]. Their idea is to represent images as a combination of a smooth *Cartoon* function containing geometric information (edges, contours, large-scale features and illumination effects), and an oscillating *Texture* function capturing small-scale features. The main advantage of this component representation, they say, is that it improves performances in classification and clustering tasks. Another approach proposed by Janner et al. [3] aims at using a Rendered Intrinsics Network to predict reflectance, shape, and lighting components. This kind of network model has the advantage of allowing the use of self-supervised learning to train over unlabeled data.

3.2 Synthetic Images

There are two reasons that can explain why synthetic images are preferred in literature for Intrinsic Image Decomposition tasks. First of all, by digitally creating images, researchers can easily generate large amount of data with specific configurations of light sources and possibly using objects that are not available or practical in the real world. The second, and probably most important aspect, is that real-world data generation is susceptible to measurement errors and noise (e.g. ambient illumination, object movement). On the other side, by using synthetic images, researchers can avoid this issues and produce highly accurate images where ground-truth values for reflectance, shading and other components are defined a priori when generating data.

3.3 Recoloring

We found the real color (in RGB) of the ball is [184, 141, 108].

In Fig. 10 we show how we can rebuild the original image by composing reflectance and shading intrinsics, along with recolored image generated using different colors for reflectance. The reconstructed images do not appear as pure green and magenta because the modified reflectance (green or magenta) is combined with the shading. The reconstruction results in an object that is no more uniform, as the green and magenta colors appear lighter or darker in different regions depending on the angulation with respect of the light source and covering from different parts of the ball (e.g. inside folds).

[2]Georgios Evangelopoulos, Petros Maragos, "Texture modulation-constrained image decomposition", Image Processing 2008. ICIP 2008. 15th IEEE International Conference on, pp. 793-796, 2008, ISSN 1522-4880.

[3]arXiv:1711.03678

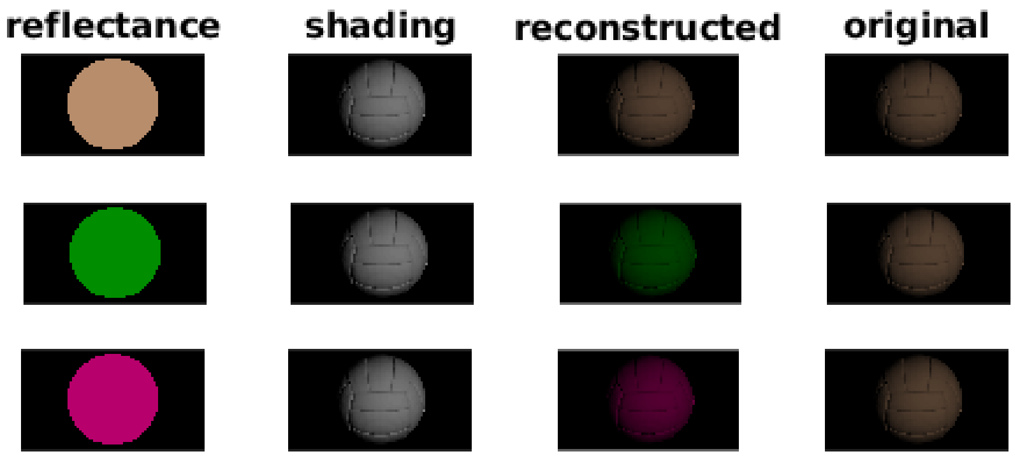


Figure 10: Splitting image into intrinsics and reconstructing it from them.

4 Color Constancy

4.1 Gray-World

4.1.1

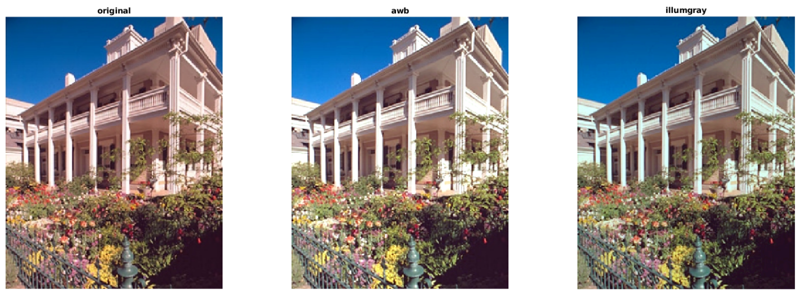


Figure 11: Gray-World algorithm (awb) and illumingray for color constancy

In Fig. 11 we show the results of applying Gray-World Algorithm onto the tested image. We also plot the image elaborated by MATLAB built-inf function *illumingray*.

4.1.2

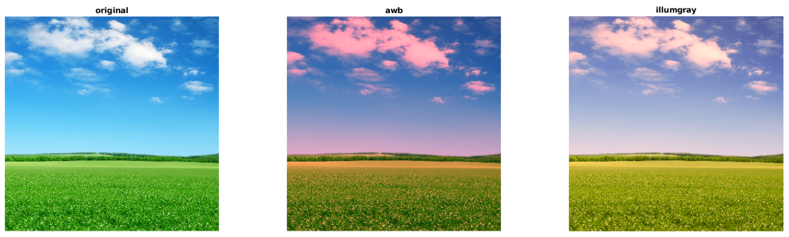
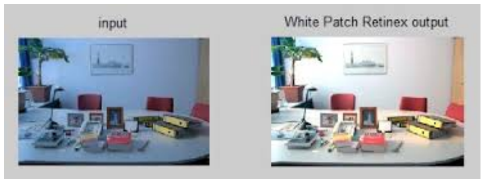


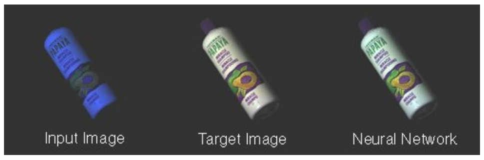
Figure 12: Failure of Gray-World when one main color is missing in the picture.

The main assumption behind Gray-World Algorithm is that the average color in a scene is gray. In pictures that satisfy this assumption, colored lights will result in a corrupted average color tending to the one emitted by the light source and ambient light. As a result, applying Gray-World will shift pixel values towards the average gray tonality [128,128,128], replacing the corrupted colored illumination with a white light source. The problem appears in scenes where the assumption is not respected. For example, in Fig. 12, the only colors present are Blue and Green, with little to no presence of Red. By applying Gray-World Algorithm, the tonality of the image is shifted towards Red to balance the average color, resulting in an unexpected and unnatural output image. Another case in which this algorithm may fail is when the input image is not static. For example, let's consider the case in which we have two images depicting the same scene, with one of them including one extra object (e.g. a person appears in the scene). In this situation, applying Gray-World may produce inconsistent output images, because the average color in the second one is different from the first one as biased by the introduction of a new colored object. The shift in the general color will vary in the two pictures, leading to an unnatural change of illumination in the same scene.

4.1.3



(a) White Patch Retinex Algorithm for Color Constancy



(b) Applying Neural Network Methods for Color Constancy.

White Patch Retinex This algorithm is based on retinex algorithm proposed by Land and McCann[4]. It considers each color channel of the image, taking the highest value as the white representation for that channel. To normalize the image, pixels are scaled according to the highest values for each channel. Performances of White Patch Algorithm can be see in Fig. 13a.

Neural Network Methods Neural Networks can also be employed to estimate the color of the illuminant and consequently correct the image. For example, Cardei, Funt and Barnard[5] used a percepton with two hidden layers, taking as input the binarized chromaticity histogram of the RGBs in the picture. The output layer returns chromaticity values *r* and *g* of the illuminant. Actual performances of Neural Network methods can be seen in Fig. 13b

Conclusion

To sum up, in this assignment we experimented with various algorithm for image manipulation, photometric modeling and color analysis using MATLAB. For each task and method discussed we learned advantages and drawbacks. By having a set of photos from the same scene taken with different light source position, it is possible to estimate the Albedo and Surface Normal of the object. This can be used to build 3D models of objects, as well as extrapolating information about the true color of surfaces, without illumination bias. Some expedients should be used to achieve consistent performances in photometric analysis. For example, it is important to have a sufficient amount of images related to the complexity of the object (anyway, at least 3) and to filter them in order to satisfy the assumptions on the whole dataset. The shadow trick can also be used to improve results (especially with few images). Changes in Color Space can be used to represent different characteristic of the image or separate features like saturation and brightness from color representations such as RGB. Intrinsic Decomposition is useful to separate feature components of light source and objects, which in the final image appear combined. As a result of applying decomposition algorithms, we are able to examine and modify those components, reconstructing modified images in the end. Color Constancy is not an easy task to tackle. Basic methods like Gray-World have strong assumptions which are often not fulfilled by real pictures. Nevertheless, many other algorithms have been proposed in literature, also including Bayesian and Neural Network approaches.

[4]Edwin H. Land, John, and J. Mccann. Lightness and retinex theory. Journal of the Optical Society of America, pages 1–11, 1971.

[5]B. V. Funt, V. C. Cardei, K. Barnard, Method of estimating chromaticity of illumination using neural networks, 1999.