# Computer Vision 1 Lab Assignment 1

Anne Chel id:10727477, Bobbie van Gorp id:11161108,
Marvin Lau id:12364282 and Milan Klaasman id:11431474

February 2019

## 1 Introduction

This document consists of the answers of the first lab assignment of the course Computer Vision 1. Containing the following four different sections of the assignment: Photometric Stereo; Color Spaces; Intrinsic Image Decomposition; and Color Constancy. It includes graphs, tables and code. The rest of the code can be found in the accompanied zip file.

## 2 Photometric Stereo

### 2.1 Estimating Albedo and Surface Normal

#### 2.1.1 Estimating Albedo and Surface Normal

It is expected to see a similar image as the original one in the albedo image as the surface is reconstructed with g, where the albedo is extracted from. The expectation is a more matte or clear surface as it excludes the illumination of the source and camera. The constructed albedo image from all Sphere5 images can be seen in Figure 1 in the middle. The image on the left is one of the input images used to construct the albedo image and is the image with the light source direction of 0.0 0.0. The image on the right in Figure 1 is also an albedo image, but no shadow trick was used in the process compared to the middle one.

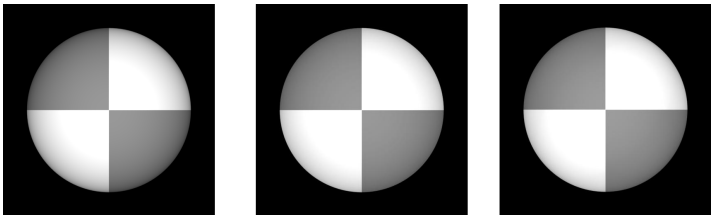


Figure 1: Sphere5 input image 0 0 on the left, constructed albedo image from all Sphere5 images with shadow trick in the middle and the right is albedo without shadow trick

From the albedo image in the middle, it can be observed that the expectation is satisfied to a certain extent when comparing to Sphere 0.0 0.0 image used for construction. The former has less shades and is more clear than the latter, especially at the white regions. However, a more noticeable difference was expected, where the albedo image would have been more clear than it is now. A possible explanation for this can be due to the limitation of having only 5 input images.

The general code that was used to estimate albedo and normal can be found in Algorithm 1, Appendix. For the following questions, this code was adjusted a bit to perform experiments with multiple images and with or without the shadow trick.

#### 2.1.2

The algorithm was ran with images from the Sphere25 folder and albedo and normal images were constructed in an incremental fashion. An overview of a portion of these albedo images can be seen in Figure 2. Starting from the left on the top row, it is constructed with 4 images and increments until 9 to the right. This continues on the second row, from 10 until 15 input images. The albedo image with 16 images and 25 images can be seen in Figure 3 on the left and in the middle respectively. A starting size of 4 images is chosen as usually the input uses larger than 3 images to get an appropriate least squares solution.

From these images, it can be observed that 4 images leads to a decent albedo image, but has a small misshape on the right side. With 5 images this improves a little, but with 6 images it suddenly get worse. So the quality of the input image

that is used in the construction is of importance as it might have caused the misshape. This misshape is corrected bit by bit with the increment of input images. At the albedo of 16 images, everything seems normal. Albedo images with input images in the range of 16 till 25 all have a correct shape and look similar. So from these results, it seems that a minimum of 16 images is needed average out any misshape and to correctly estimate the albedo.

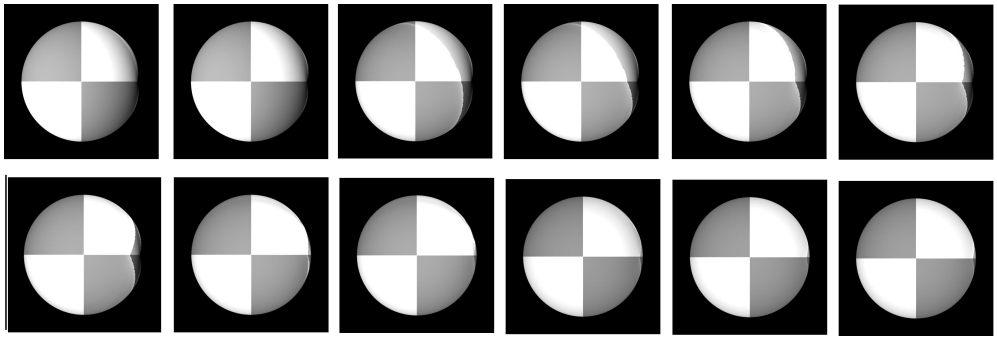


Figure 2: An overview of albedo images with a range from 4 until 15 for amount of input images of Sphere25 with shadow trick

#### 2.1.3

The shadow trick uses a diagonal matrix that is created from the image vector and multiplies the original equation $i(x, y) = Vg(x, y)$ on both sides. It has the effect of zeroing the contributions of the regions that are in the shadow. Since it makes use of a diagonal matrix, it is particular interesting with the multiplication with the matrix on the right side as the resulting matrix will contain some zeros due to the diagonal matrix having zeros on the off-diagonal. The right side that contains the surface g and illumination property V may contain points that contribute to shadows and are then zeroed out as elements on the off diagonal are zero.
Albedo images with shadow trick were already presented in previous sections such as the Sphere5 albedo image in Figure 1 in the middle. On the right of it, The albedo image without shadow trick can be seen. The difference is very small, but the albedo image with shadow trick does have a bit more shadow than the albedo image without shadow trick in both the white and black regions of the circle.

In the case of 25 images of Sphere25, the albedo image with shadow trick can be seen in 3 in the middle and to the right of it without shadow trick. The difference is also small, but the image without shadow trick is slightly brighter than with shadow trick, mainly in the black regions.

The impact of the shadow trick should be that it negates the contribution of shadow regions, if they are present in the image, making the image more clear. Using the shadow trick requires more overhead, but on the scale of these 2 cases it is not much. On images with less shadows, it may have less effect or might not be worth the trade off between gain and overhead in terms of scalability.

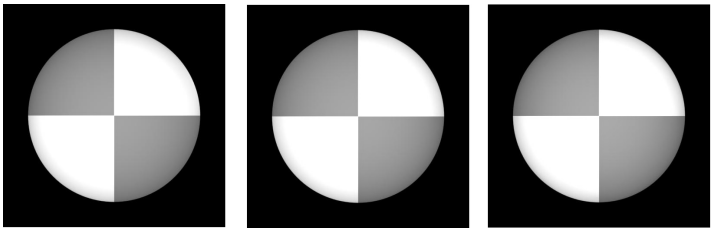


Figure 3: 16 input images from Sphere25 on the left, 25 input images from Sphere 25 in the middle and on the right again 25 input images but without shadow trick

On the other hand, comparing the normal map and normal components with and without shadow trick, seen in Figure 4a and 4b respectively, it can be observed that the latter contains some elliptical lines at the images of normals. The normal map and components plots with shadow trick does not have these lines. So the shadow trick can help smoothing the normal components by negating the contribution of shadows.

### 2.2 Test of Integrability

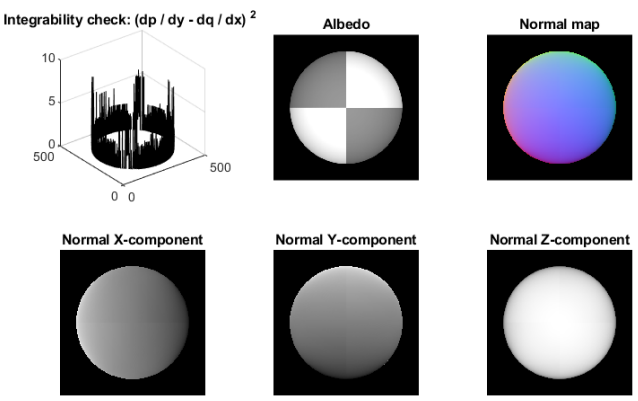
The general code to compute the derivatives can be seen in Algorithm 2, Appendix. Second order derivatives were computed with backward difference.

For the test of integrability, thresholds of 0.01 and 0.005 were chosen for measurements of amount of outliers as those are reasonable threshold, but still have some distance apart from each other. Test cases include the Sphere 5 case and multiple input images of Sphere25. The results of the measured amount of outliers can be seen in Table 1. First, it can be observed that the amount of outliers is greater when the threshold is 0.005 than when it is 0.01. This is logical as a smaller threshold means it is more strict in what it is considered an outlier. Second, when looking at different input sizes for Sphere25, the amount of outliers is in general lower when less input images are used as 7 has the lowest amount of outliers. With 11 and 16 images, the amount increases and with 25 images the amount decreases. A possible explanation might be related to the misshape that was observed in section 2.1.2. Outliers can be interpreted as edges when the second derivatives are plotted. Some second derivatives plots for 5 Sphere5 case and input size 7,11,16,25 with Sphere25 respectively can be found in the appendix. Due to misshapes, some edges might not be properly detected as in the case of 7 images with a misshape. With more images, the shape corrects itself as mentioned before as edges get sharper, which leads to greater edge detection and outliers. Having enough images, such as in the 25 case, edges can be more refined and decreases the amount of outliers.

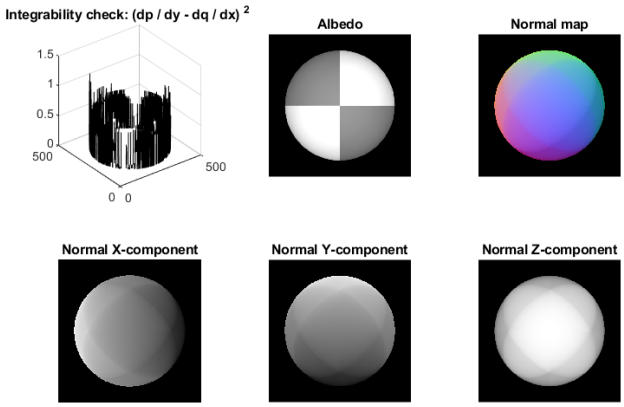
In addition, surf plots of the differences in integrability check for Sphere5 with and without shadow trick can be seen at the top left in Figure 4a and 4b. From this plot it can be observd that outliers lie at the edges. Furthermore, it can be observed that the use of the shadow trick leads to greater difference in second order partial derivatives. The shadow trick may remove shadows, which causes a higher contrast for edges to be detected, leading to greater outliers.

| Case | Threshold | Amount of outliers |
|---|---|---|
| 5 Sphere5 images | 0.01 | 2901 |
| 5 Sphere5 images | 0.005 | 3641 |
| 7 Sphere25 images | 0.01 | 1539 |
| 7 Sphere25 images | 0.005 | 1765 |
| 11 Sphere25 images | 0.01 | 2291 |
| 11 Sphere25 images | 0.005 | 2911 |
| 16 Sphere25 images | 0.01 | 2666 |
| 16 Sphere25 images | 0.005 | 3346 |
| 25 Sphere25 images | 0.01 | 2233 |
| 25 Sphere25 images | 0.005 | 2783 |

Table 1: Outlier results of test of integrability




(a) An overview of results with all Sphere5 images with shadow trick




(b) An overview of results with all Sphere5 images without shadow trick

Figure 4

.

## 2.3 Shape by Integration

### 2.3.1

Height maps and 2 views of direction of normals are plotted in Figure 5 for each different path using all the images for Sphere5. The plots from top to bottom are for the column, row and average path respectively. From the height maps, there are very small noticeable difference. The column height map has a slight "cut" of an edge on the left side and the

row height map has it at the bottom. The average height map has it both positions, but is corrected a bit as it takes the average. Looking at the direction of the normals, there is a slight difference in direction between the column and row major order path. For the row path, the normals diverges quicker at the top of the sphere compared to the column path. It can be interpreted as a "peak" that is sticking out a bit, whereas with the column it is more of round shape according to the direction of the surface normals that are perpendicular to the surface.

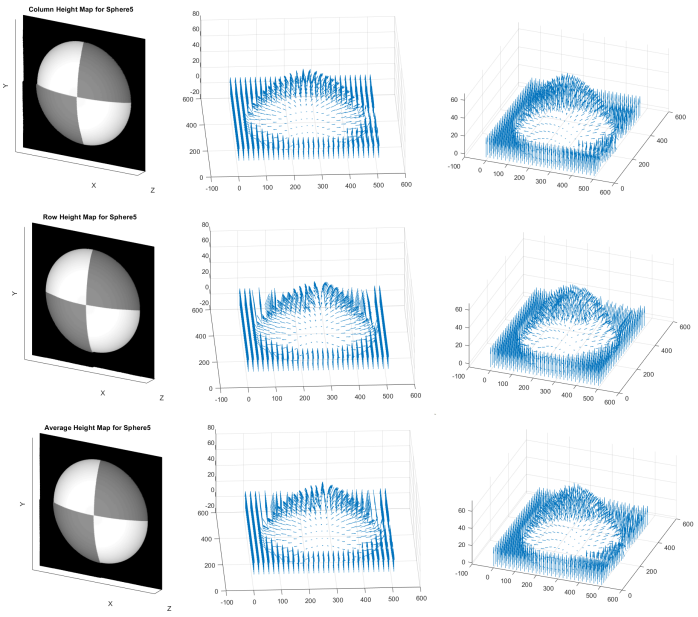



Figure 5: An overview of plots of height maps and normals directions with column major order at the top, row major path on second line and the average path at the bottom for Sphere5

#### 2.3.2

The results of the average path can be seen at the bottom of 5. It negates the "peak" or "divergence" effect a bit in the normal plots of the row path as it takes the average. Next to that, there is no further improvement. When looking at different input sizes such as 7,11,16 and 25 again for Sphere25, it does play a role for the construction of the surface and the normals. These plots can be found in the appendix from Figure 19 to 22. With 7 images, the surface construction is missing large part and also for the normals. With 11 input images, this improves, but only with 16 it is complete, which also corresponds to minimum amount of input images. With 25 images it is even more refined as the "cuts" from earlier have been fixed and the whole surface construction and normals seems fine.

### 2.4 Experiments with different objects

#### 2.4.1

Plot results of running the algorithm with all 121 images of MonkeyGray can be seen in Figure 6 with shadow trick and column major order for the height map. The amount of outliers were measured under the threshold of 0.005 for multiple input sizes for MonkeyGray. Those results can be seen in Table 2. Overall, the MonkeyGray images has more outliers than the Sphere5 and Sphere25 as seen in Table 1. Outliers are present mainly on the edges or corners of the object. Since MonkeyGray has a greater surface area and more edges, the amount of outliers increases compared to the Sphere case. With different input sizes, the amount of outliers changes. It takes on a similar pattern as in the Sphere5 case discussed in section 2.2. More images lead to construction edges, which increases the amount of outliers. Up to a certain amount of input images, the edges get more refined are decreases the amount of outliers as in the case when all 121 MonkeyGray images are used.

#### 2.4.2

With 3 channels, we can load images from each channel and compute the albedo and normal individually for each channel. The albedos can then be recombined into a 3D matrix to combine the three channels. For the normals, the average is taken to give a well balanced representation for the normal components. However, for the height map, the normals of each channel are used to compute the derivatives, which in turn are used to create a height map for each channel. These three

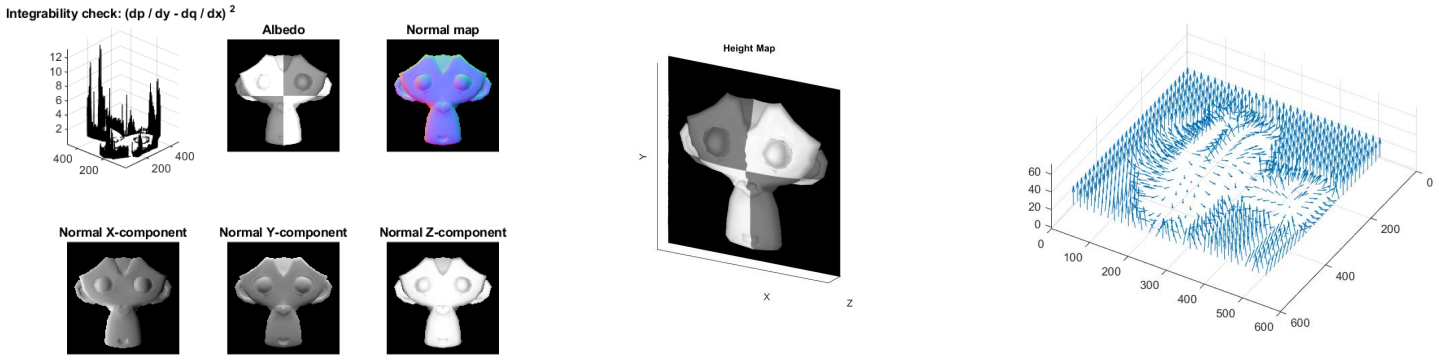
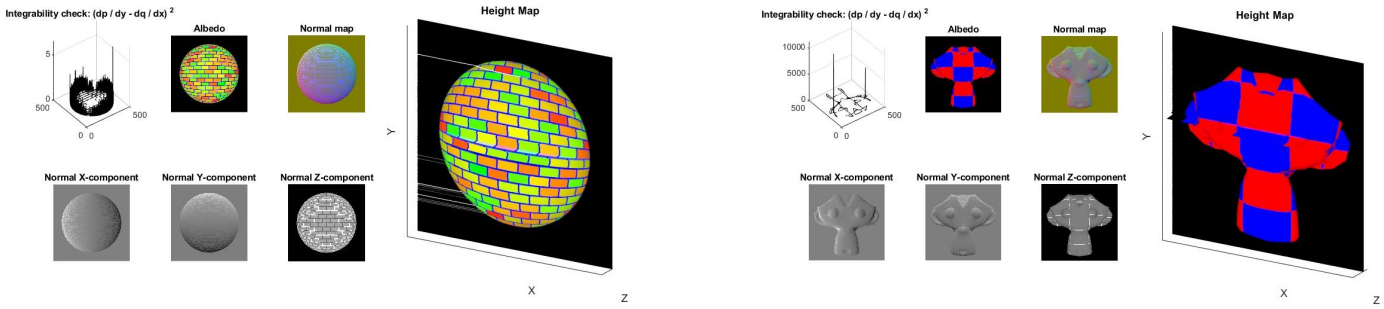



Figure 6: An overview of plot results using all 121 images in MonkeyGray with shadow trick

| Case | Threshold | Amount of outliers |
|---|---|---|
| 10 MonkeyGray images | 0.005 | 12453 |
| 25 MonkeyGray images | 0.005 | 15516 |
| 50 MonkeyGray images | 0.005 | 13535 |
| 75 MonkeyGray images | 0.005 | 12259 |
| 100 MonkeyGray images | 0.005 | 11032 |
| 110 MonkeyGray images | 0.005 | 10514 |
| 121 MonkeyGray images | 0.005 | 9997 |

Table 2: Outlier results of test of integrability

height maps summed and divided by 3 to take the average. The NaNs of normals and albedo that are used to construct the height map are replaced with zeros in order to avoid fail attempt to construct an height map. So the solution to this is to replace it with zeros and take the average of height maps of the three channels to make sure it will mostly have values that are above zero. The updated implementation on SphereColor and MonkeyColor can be seen in 7b and 7a.




(a) An overview of results of SphereColor

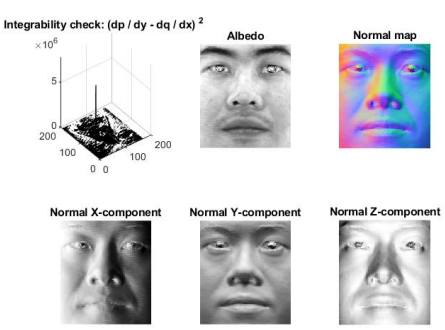
(b) An overview of results with MonkeyColor

Figure 7

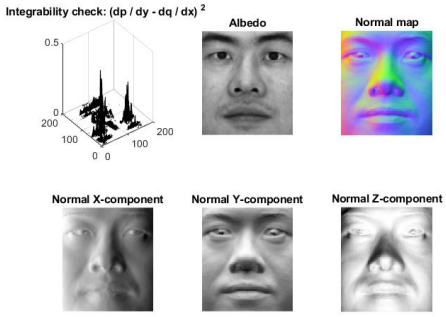
### 2.4.3

An overview of results on the Yale data set can be seen 8 with and without shadow trick. In addition, an overview of height maps can be seen in Figure 9. The height map on the left side is with the use of the shadow trick and the other three heights maps are without the shadow trick. From these results it can be observed that without shadow trick gives more sensible plots, especially observable in the height maps. Comparing the albedo images from the two cases, it can be seen that the eyes differ from each other. With the shadow trick, the eyes become more illuminated or white. This consequence leads to an height map that is distorted at the position of those eyes in the z axes. It assumes that the eyes are shadows and try to remove them basically. This violate the assumptions of the shape-from-shading methods as the color of the eyes should stay the same and are not shades. Furthermore, different integration paths can be seen without the shadow trick in Figure 9. The second, third and fourth height maps are the column, row and average integration path respectively. The column path gives the best result, the row gives the worst as creates a sort of peak at the nose and mouth that points out a bit as seen before with the sphere case in section 2.3. The average integration path combines the former two to make a balance in depth and construction of face. However, there is still a little deformation as the mouth creates a more strict

face and the nose is pointy. Thus the column path gives the best result in construction of the albedo or original face in shape.




(a) An overview of results with all Yale images with shadow trick




(b) An overview of results with all Yale images without shadow trick

Figure 8: Overview of results with shadow trick on the left and without on the right

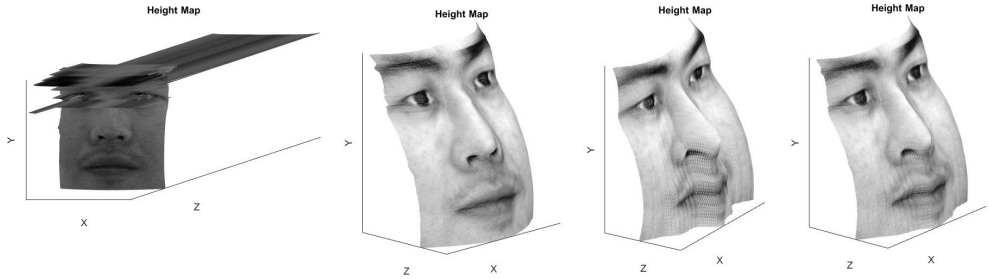



Figure 9: Overview of height map on the Yale dataset, with shadow trick on the most left, the other three without shadow trick with column, row and average major order path construction respectively

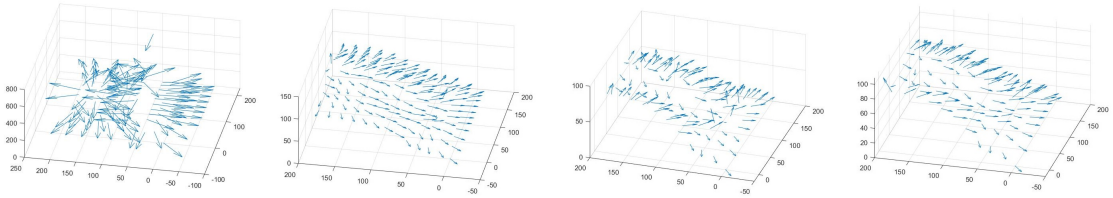


Figure 10: Overview of normals on the Yale dataset, with shadow trick on the most left, the other three without shadow trick with column, row and average major order path construction respectively

Some further properties, most parts of the face are opaque and from the images containing the Yale data set, the nose can be very reflective as well. Removing images that are not too informative and are rather noise might improve the plots. Examples of problematic images are; images that have noise as diagonal lines, images that are too dark as large parts of the picture are black and some images are too bright as the whole face turns white almost. Images that are in-between are more well-balanced and desired. Removing some of these problematic images, in total 16, results were plotted without shadow trick and using column integration path that can be seen in Figure 11. From the height map, it can be observed that the face is longer slightly tilted as before and the face look similar to the column height map in Figure 9. Other improvement that less images are needed to construct it, which decreases the overhead in computing them.

# 3 Color Spaces

## 3.1 RGB Color Model

Human color vision is the result of three different photoreceptor cells on the retina with different absorption spectra. A camera should capture light so that the difference in response of the different photoreceptor cells between the human

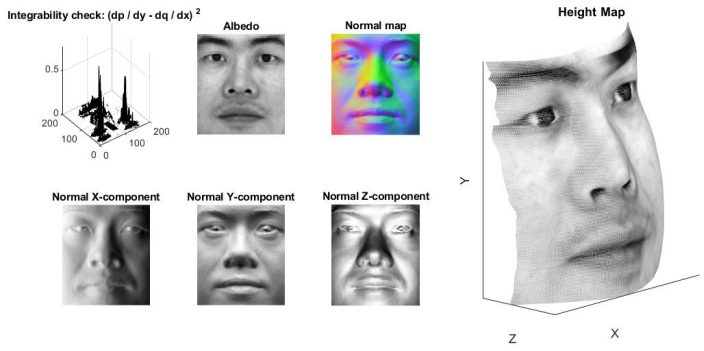
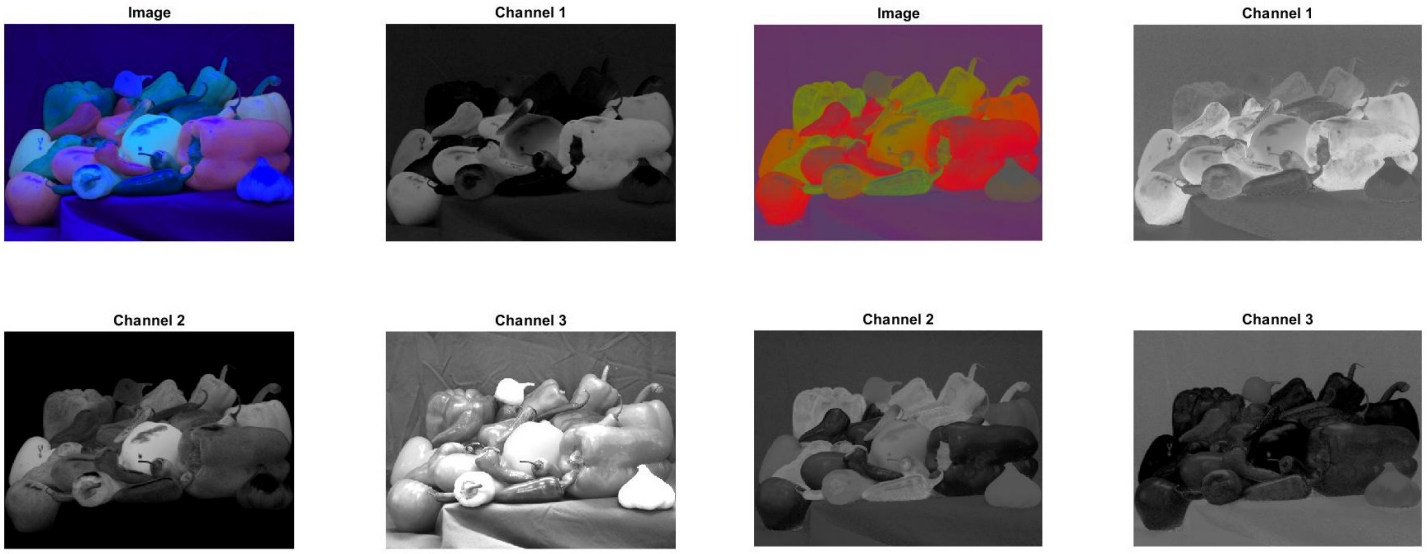



Figure 11: Overview of results on Yale data set with 48 images instead of the previous 64 images with column integration path and no shadow trick




(a) RGB converted into the opponent color space and the three channels visualized separately.

(b) RGB converted into the normalized RGB (rgb) color space and the three channels visualized separately.

Figure 13

directly observing the light source and the human observing the light from the image produced by the camera is as small as possible. Therefore the idea is to represent colors in terms of additive primary colors that isolate the different photoreceptor cells as much as possible, so that the response of the photoreceptor cells to different wavelengths can effectively be recreated by a combination of these primary colors. Although this isolation is difficult, as two of the three photoreceptor cells have very similar absorption spectra, red, green and blue turn out to be a very good choice. A camera captures the RGB values with red, green and blue sensors that absorb a specific portion of the spectrum and integrate over the incoming spectrum of light for each of these sensors to acquire RGB values.

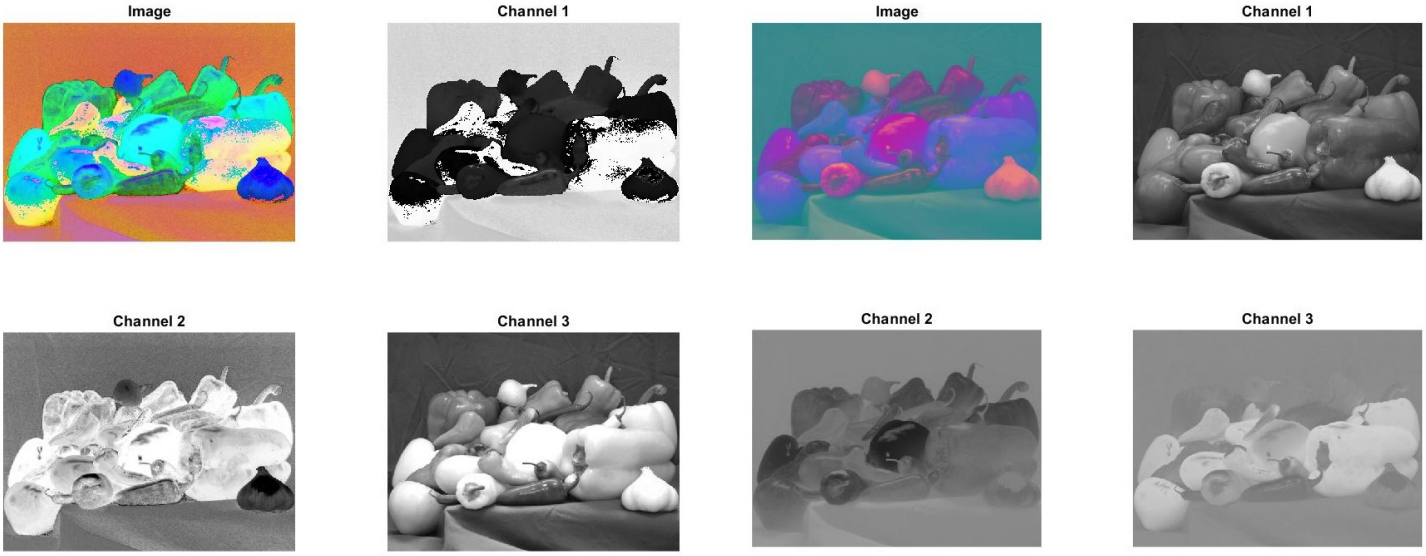
## 3.2 Color Space Conversion

The resulting images for the conversion of RGB color model images to grayscale and the opponent, rgb, HSV and YCbCr color spaces are shown in figures 12, 13 and 14.

## 3.3 Color Space Properties

Opponent
In this color space, the intensity is represented in channel O3 and the color information in O1 and O2. It is also shift-invariant with respect to the light intensity.

rgb
Normalized RGB or rgb are simply the color ratios, so there is no intensity information in this color space. This can be useful to eliminate the effect of different light intensities from the same light source, for example in computer vision algorithms where an object that is being tracked moves to a location with a lower intensity from the same light source.




(a) RGB converted into the HSV color space and the three channels visualized separately.

(b) RGB converted into the YCbCr color space and the three channels visualized separately.

Figure 14

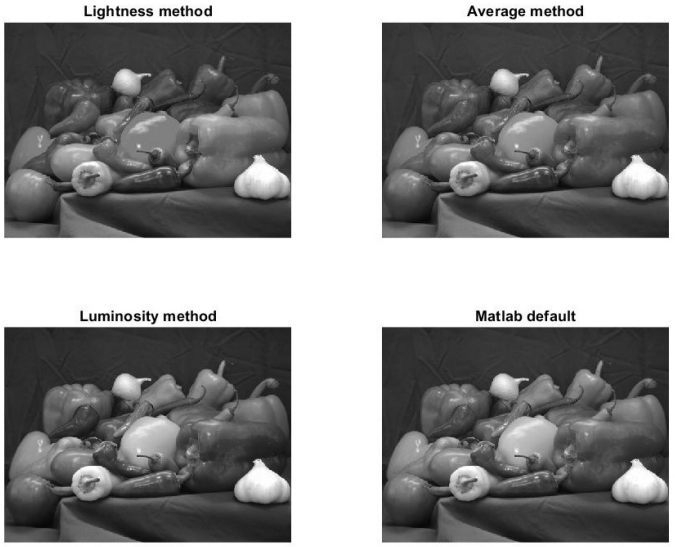



Figure 12: Different methods for converting RGB images into grayscale

HSV

An alternative way of representing RGB colors that is more interpretable for human designers is HSV. The three channels of HSV represent the hue (direction on a color wheel), saturation (combination of light intensity and how much it is distributed across the spectrum of different wavelengths) and value (brightness relative to white).

YCbCr

In the YCbCr color space, Y is the luminance (brightness) component, while Cb and Cr are chrominance (color) signals. A benefit from this split is that the chrominance and luminance signals can be compressed to different levels of fidelity. Since humans have poorer frequency response to chrominance than to luminance, the chrominance signal can be compressed with lower fidelity for a more effective compression of the image or video.

Grayscale

Images and video in grayscale only consist of one channel, representing luminance. The advantage of this is that the image or video takes up less memory space than the colored variants, while they are still easily interpretable by humans.

### 3.4 More on Color Spaces

The CIELAB color space [7] expresses color in the channels L* (lightness), a* (green-red) and b* (blue-yellow). The model is device-independent with respect to a given white point and at the time of its creation, the idea was that a change in value of one of the components corresponds to a similar visually perceived change. Because the space encompasses both the RGB and the CMYK color model gamuts, CIELAB can for example be used when an RGB image has to converted to CMYK for printing.

## 4 Intrinsic Image Decomposition

### 4.1 Other Intrinsic Components

Besides albedo and shading, an image can be decomposed in multiple different components. Firstly, light and color, represented by hue, saturation and intensity. Hue is the difference or resemblance to a different stimuli that is described

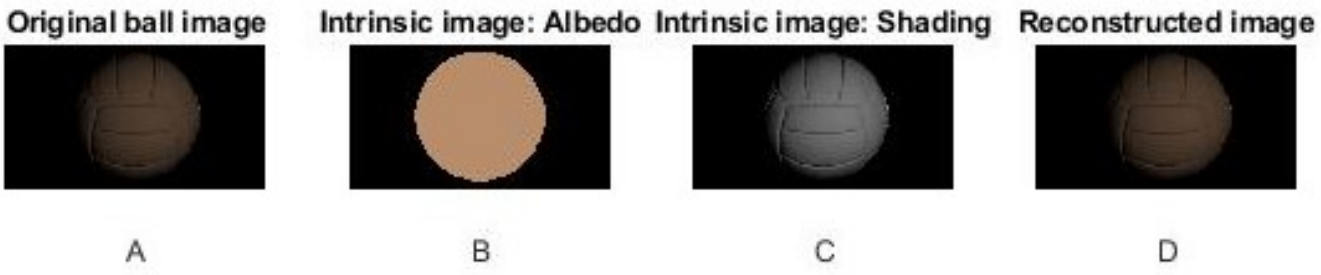



Figure 15: A representation of the original image; the two intrinsic images, albedo (B) and shading (C); and the reconstructed ball image (D).

as red, green, blue and yellow; saturation is the purity of the color; and intensity is the brightness of the color. Color is composed of the trichromacy, which is a representation by giving values for the redness, greenness and blueness (RGB) of the color. One other intrinsic component an image can be decomposed of is the reflectance of the image. Different surfaces have different reflectance spectra [6]. This visual appearance is called specularity

## 4.2 Synthetic Images

> *"Intrinsic image decomposition is a challenging, long-standing computer vision problem for which ground truth data is very difficult to acquire."* [5]

Rendering synthetic scenes is used to generate a vast amount of training data for intrinsic images. Nonetheless, the existing synthetic data sets are limited to images of single objects [3, 8]. Alternatively, crowdsourcing is used to collect ground truth for real images. However, the annotations in such datasets are sparse and difficult to collect accurately at scale [5]. Thus, it is difficult to find and generate labeled training data sets of non-synthetic images.

## 4.3 Image Formation

Recomposing the ball from the two intrinsic images, albedo and shading, is done by elementwise multiplication of both the albedo and shading image. Firstly, the albedo and shading images were cast to doubles. Because it is not possible to do the elementwise multiplication, with albedo and shading being of type uint8. Secondly, dividing the shading values by 255 (is the maximum grayscale value) to get double values between 0 and 1. Thirdly, the elementwise multiplication of albedo and "normalised" shading. The result is cast back to the uint8 format. In Figure 15, the output figures for the recomposition can be found.

## 4.4 Recoloring

### 4.4.1

The true material color of the ball in RGB space is [184 141 108]. These values were found using the albedo image. By using Matlab's *datacursormode('on')* and scrolling over the albedo image, gave the RGB values, which indeed are uniform.

### 4.4.2

To answer this question, the color values of the ball image should be changed. The color green is represented by (0, 255, 0). The third column of an image represents the color scheme. Thus the first and the third values of the the 3rd column should be replaced by 0, which corresponds with the color green (0, 255, 0). This means that the green values in the RGB are not changed and thus still represents the ball and his shading features.

### 4.4.3

The reason that the reconstructed image does not display pure green is due to the fact that the G values (in the RGB scheme) are not all 255 (pure green). Because only the values for R and B are changed to zero, this results in a representation of the G values, which still represents the shading and texture of the ball.


Figure 16: A representation of the original ball on the left and the green recolored ball on the right

# 5 Color Constancy

The gray-world algorithm is implemented and can be found in Algorithm 4, Appendix. The result of the transformation on the given image in the assignment can be found in figure 15, Appendix.

The gray-world assumption encounters problems when an image is presented containing akin colours. An example of such an image[1] is visible in the figure below, where the gray-world algorithm fails due to the existing, dominant color.

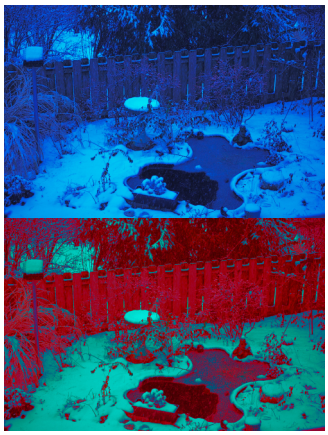


Figure 17: Failed gray-world assumption

Other color constancy algorithms are proposed in literature. Vivek Agarwal et al.[1] describe, amongst others, the scale-by-max algorithm, where instead of taking the mean of each channel, the maximum is calculated. Several other algorithms are also elaborated in their paper, such as the Multiscale-Retinex algorithm based on the single-scale retinex method proposed by Land and McCann [4], which follows the human anatomy of the eye. More recent approaches rely moslty on machine learning. Simone Bianc et al.[2] introduce a Convolutional Neural Network, sampling patches from the image that do not imbricate, calculating the illuminant for every patch. Zhenmin Zhu et al. propose the least square support vector regression technique for estimating the illumination.

# 6 Conclusion

In this assignment four topics were covered: Photometric Stereo; Color space; intrinsic image decomposition; and Color constancy.

The distribution of workload of this assignment is as follows: Marvin did Photometric Stereo (Section 2), Bobbie did Color Spaces (Section 3 ), Milan did Intrinsic Image Decomposition (Section 4, Introduction (Section 1 and Conclusion (Section 6 ) and Anne did Color Constancy (Section 5.

[1] http://therefractedlight.blogspot.com/2011/09/white-balance-part-2-gray-world.html

# References


[1] Agarwal, V., Abidi, B., Koschan, A., and Abidi, M. An overview of color constancy algorithms. *Journal of Pattern Recognition Research 1* (06 2006), 42–54.

[2] Bianco, S., Cusano, C., and Schettini, R. Color constancy using cnns. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (June 2015).

[3] Janner, M., Wu, J., Kulkarni, T. D., Yildirim, I., and Tenenbaum, J. Self-supervised intrinsic image decomposition. In *Advances in Neural Information Processing Systems* (2017), pp. 5936–5946.

[4] Land, E. H., and McCann, J. J. Lightness and retinex theory. *J. Opt. Soc. Am. 61*, 1 (Jan 1971), 1–11.

[5] Li, Z., and Snavely, N. Cgintrinsics: Better intrinsic image decomposition through physically-based rendering. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 371–387.

[6] Ma, Y., Feng, X., Jiang, X., Xia, Z., and Peng, J. Intrinsic image decomposition: A comprehensive review. In *International Conference on Image and Graphics* (2017), Springer, pp. 626–638.

[7] CIE Technical Committee. Cie technical report colorimetry 3rd edition, 2004.

[8] Shi, J., Dong, Y., Su, H., and Yu, S. X. Learning non-lambertian object intrinsics across shapenet categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1685–1694.

# APPENDIX

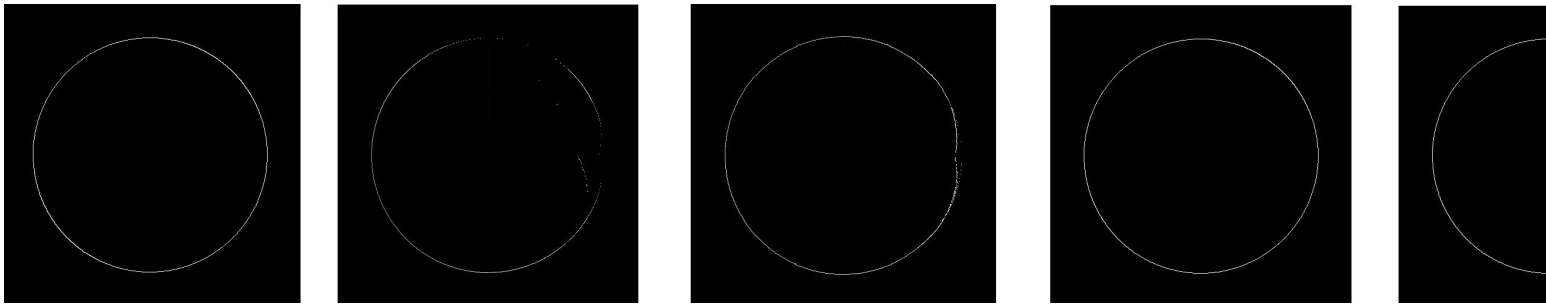


Figure 18: An overview of plotted second derivatives

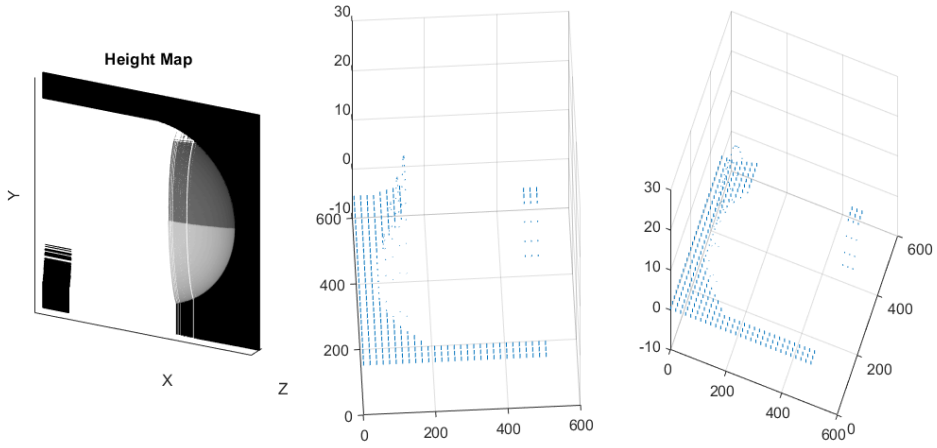



Figure 19: An overview of plots of height maps and normals directions with column major order at the top, row major path on second line and the average path at the bottom for 7 input images of Sphere25

Algorithm 1: Computing albedo and normal

```
stop = size(image_stack,3); %To indicate how many images to use
scriptV = scriptV(1:stop,:);
for x = 1:w
    for y = 1:h
        i = image_stack(y,x,1:stop);
        i = squeeze(i);
        if shadow_trick
            scriptI = diag(i);
            g = linsolve(scriptI*scriptV,scriptI*i);
        else
            g = linsolve(scriptV,i);
        end
        albedo(y,x) = norm(g);
        normal(y,x,:) = g./albedo(y,x);
    end
end
```

Algorithm 2: Computing first and second order (partial) derivatives

```
p = zeros(h,w);
q = zeros(h,w);
SE = zeros(h,w);
second_deriv_xy = zeros(h,w);
```

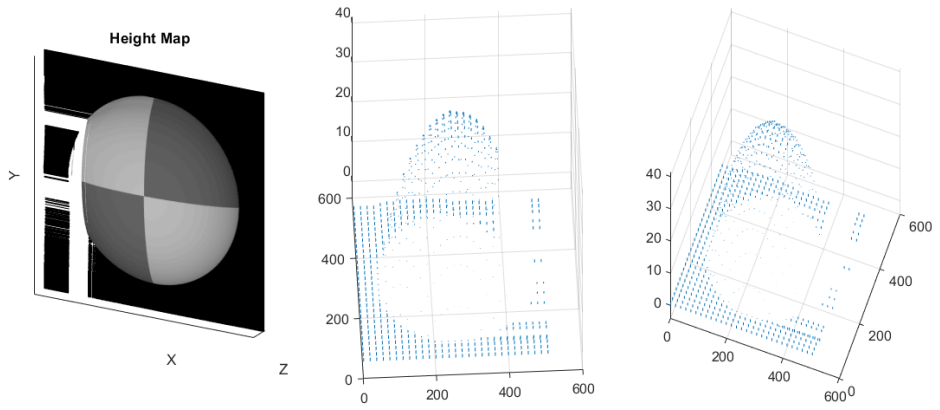



Figure 20: An overview of plots of height maps and normals directions with column major order at the top, row major path on second line and the average path at the bottom for 11 input images of Sphere25

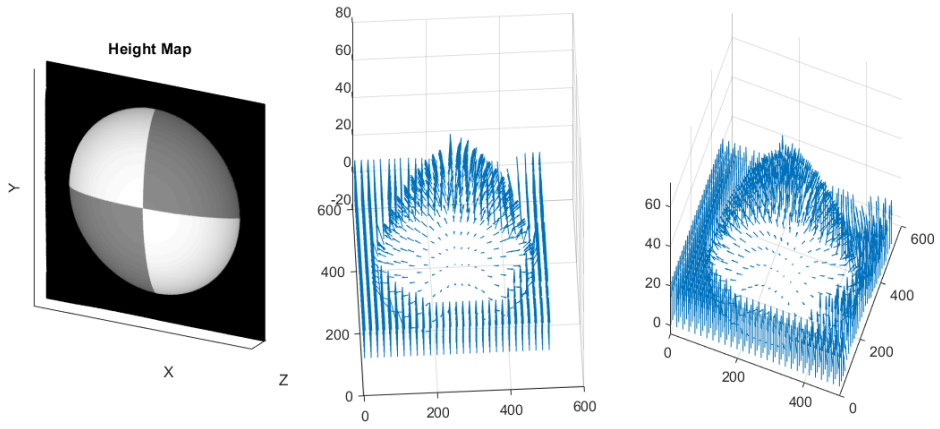



Figure 21: An overview of plots of height maps and normals directions with column major order at the top, row major path on second line and the average path at the bottom for 16 input images of Sphere25

```
second_deriv_yx = zeros(h,w);

for x = 1:w
    for y = 1:h
        normal_point = normals(y,x,:);
        n = squeeze(normal_point);
        p_value = n(1)/n(3);
        q_value = n(2)/n(3);
        p(y,x) = p_value;
        q(y,x) = q_value;
    end
end

for x = 1:w
    for y = 1:h

        if x > 1 second_deriv_xy(y,x) = p(y,x) - p(y,x-1), else second_deriv_xy(y,x) = 0, end

        if y > 1 second_deriv_yx(y,x) = q(y,x) - q(y-1,x), else second_deriv_yx(y,x) = 0, end

        diff = (second_deriv_xy(y,x) - second_deriv_yx(y,x))^2;
        SE(y,x) = diff;
    end
end
```

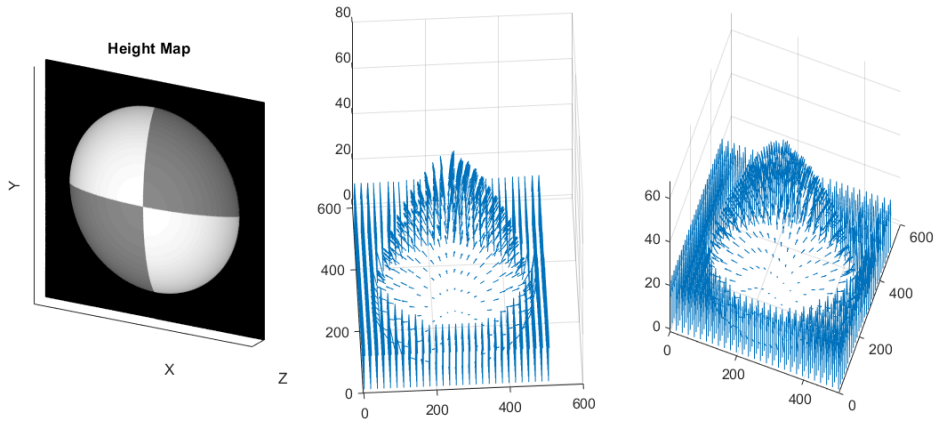



Figure 22: An overview of plots of height maps and normals directions with column major order at the top, row major path on second line and the average path at the bottom for 25 input images of Sphere25


Figure 23: Original image and corrected image using the Grey-World algorithm

Algorithm 3: Constructing surface height map

```
height_map_column = zeros(h, w);
height_map_row = zeros(h, w);
for y = 2:h
    height_map_column(y,1) = height_map_column(y-1,1) + q(y,1);
end

for y = 1:h
    for x = 2:w
        height_map_column(y,x) = height_map_column(y,x-1) + p(y,x);
    end
end

 %----rowcase------
for x = 2:w
    height_map_row(1,x) = height_map_row(1,x-1) + p(1,x);
end

for x = 1:w
    for y = 2:h
```

```
        height_map_row(y,x) = height_map_row(y-1,x) + q(y,x);
    end
end

%----Averaging them------
height_map = (height_map_column + height_map_row)/2;
```

Algorithm 4: The gray-world algorithm

```
function [GW_image] = GrayWorld(I)
    [Row Col Layer] = size(I);
    GW_image = uint8(zeros(Row, Col, Layer));

    R = I(:,:,1);
    G = I(:,:,2);
    B = I(:,:,3);

    R_mean = mean(mean(R));
    G_mean = mean(mean(G));
    B_mean = mean(mean(B));

    Average = mean([R_mean, G_mean, B_mean]);

    R_scale = Average / R_mean;
    G_scale = Average / G_mean;
    B_scale = Average / B_mean;

    GW_image(:,:,1) = R * R_scale;
    GW_image(:,:,2) = G * G_scale;
    GW_image(:,:,3) = B * B_scale;

end
```