Comuputer Vision Lab 1

Maximilian Schlögel UvA
Anca-Diana Vicol VU
Benjamin Kolb UvA
Philipp Lintl UvA

1 Introduction

In the field of Computer Vision understanding concepts such as Photometric Stereo, Color Space, Intrinsic Image Decomposition and Color Constancy is crucial in order to find solutions for a wide range of problems. The Photometric Stereo is a method of reconstructing the 3D shape of a surface using several images of the surface with different illuminations. Color Spaces are the ranges of colors that can be expressed using different Color Models and understanding them gives any researcher a better grasp on the methods of encoding chromatic information. In order to not only study an image but also edit it, the image can first be split into its components, like reflectance and shading, through a process called Intrinsic Image Decomposition. These components can then be easily processed and merged solving important problems like recoloring. As many of the images used in the aforementioned processed are photographies of the surrounding world, the environmental variables such as the color of the light source can have a grave impact on the quality of the data. Thus, algorithms for Color Constancy are developed to combat this issue and correct the overall color range of the image.

The exercises described in the current report were devised for a better understanding of the theory, implementations and limitations of the abode cocepts and methods.

2 Photometric Stereo

In the first exercise we will implement the photometric stereo algorithm. This section is divided into four main parts. We will recreate the algorithm for gray images step by step in the first three subsection. Here we estimate the albedo and recreate the surface normal of the images. After testing the integrability of the resulting surface normal in part two, we will use those normals in part three to recreate the shape of the displayed object by integrating over the derivatives. The fourth subsection will be used to generalize the algorithm onto colored images and furthermore to experiment with different and more complex objects.

2.1 Estimating Albedo and Surface Normal

We start off by estimating the albedo and surface normal of gray spheres. This part is based on chapter 5 of [4]. By solving the following linear system for $g(x, y)$ we are able to extract the surface normal by normalizing vector field g and the albedo by taking the normalization factor for each pixel.

$$Ii = IVg(x, y)$$

Here I is the shadow trick explained later, i is the vector containing the pixel values at (x, y) for all training images, V contains the information about the camera and the relative positioning of the light sources.

2.1.1

Looking at the image of the sphere (figure 1), where the light source is right above, we would assume that the albedo is constant for the white parts and grey parts. Furthermore we would assume the

Preprint. Work in progress.

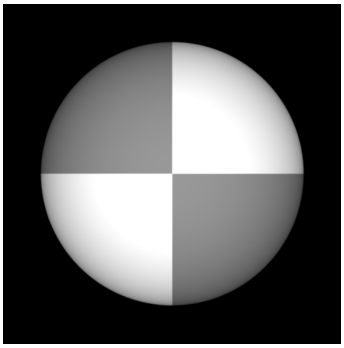

Figure 1: Original image

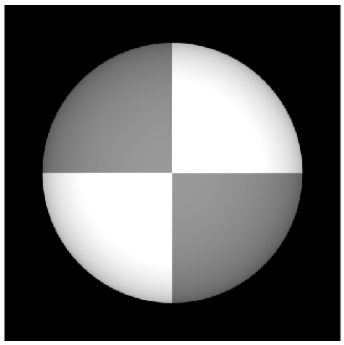

Figure 2: Estimated albedo with 5 images

darkening close to the sphere boundary is due to shadows and not a decline in albedo. The results looks similar to what we expected (figure 2), but the gray and white areas seem to get a bit darker towards the boundary of the sphere. Whether this is due to noise, perception or really a decrease of albedo is hard to tell, but the portrayed albedo seems to correspond strongly with our expectation.

2.1.2

Theoretically the minimum number of images needed to estimate the albedo and the surface normal is at least three, because with less images the matrix V used to compute the normal cannot have full rank and therefore the normal does not have to have a unique solution. When comparing the estimated albedo using five training images in figure 2 with the estimated albedo using 25 training images in figure 3, we see that the darker areas near the boundary are vanishing, when using more training images. Especially for the normal map we see a large difference in quality. When using 5 images we clearly see the lines of the shadows in figure 4, while they are not visible anymore when using more images (see figure 5). These differences are due to the fact that using more images allows us to distinguish better between shadows and actually darker areas of the objects.

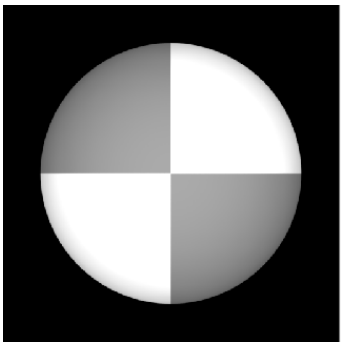

Figure 3: Estimated albedo with 25 images

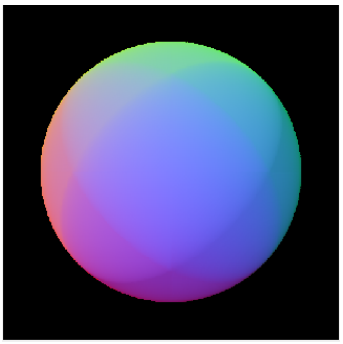

Figure 4: Normal map with 5 images

Figure 5: Normal map with 25 images

2.1.3

Shadows have a negative impact on the quality of the photometric stereo algorithm. Practically this is due to the fact that the image does not deliver any useful information about parts of the image that lie in the shadow. In theory a pixel that lies in the shadows on too many images leads to a linear system in our algorithm that does not have a unique solution. This could lead to a worse estimation of the surface normal or in the worst case to an ambiguous surface normal.

Optimally the pixels in the shadows should have no contribution to the estimation of the albedo and the surface normal. By multiplying the linear system (I from above) on both sides by a matrix that has the pixel values of all images on the diagonal, we multiply those rows by zero, that correspond to the images where the pixel is dark (i.e. has a pixel value of zero). Using this, stops the linear systems to solve for non-sensical values. When using the shadow trick, the unwanted lines in figure 4 vanish and the normal map looks very similar, if not smoother, to the normal map displayed in figure 5. Combining 25 training images and the shadow trick leads to a slightly smoother normal map than the one in figure 5

2.2 Test of Integrability

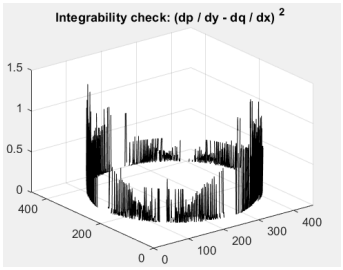

Figure 6: SE values for 5 training images

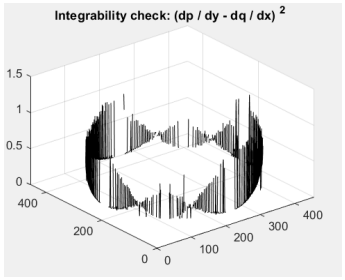

Figure 7: SE values for 25 images

To recreate the surface height map, we need a function describing the height map in dependence of the pixel coordinate. Since the surface normal can be described by the partial derivatives of this function,

$$N(x,y) = \frac{1}{\sqrt{1 + \frac{\partial f^2}{\partial x} + \frac{\partial f^2}{\partial y}}} \left\{ -\frac{\partial f}{\partial x}, -\frac{\partial f}{\partial y}, 1 \right\}^T$$

we can use the normal vectors to compute the derivatives. In part three we will integrate over those derivatives to compute the actual height values as a function of the pixel coordinates. This method relies on the fundamental theorem of calculus which also has as a consequence that we only know the relative height values.

To test the integrability we will check whether the mixed second partial derivatives $\frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial y \partial x}$ are equal. Since we operate on the space of pixels, we use the method of 'neighbor difference' which allows us to apply differentiation on the discrete space and approximate the differentiation of the underlying real-valued height function. To do this we go along the axis and take the difference of value of the corresponding first-order derivatives. Since we generally will expect the mixed partial derivatives to be almost equal and assume the numerical computations to only make small approximation errors, we set the threshold, that detects outliers, to a value of 0.005. When looking at figure 6 and 7 we see that all high errors occur along the boundary of the surface. This suggest that the approximation leads to higher errors when dealing with relatively large changes in the first-order derivatives. We hypothesize this may be due to the discretization of the second-order derivative, which is more error prone in areas with fine-grained changes. When comparing both figures, we see that the error values for the case with 25 training images are generally lower than for the five image case. This might be due to the fact that more training image lead to a more accurate approximation of the real second derivative. Since in theory they should be the same, a better approximation also would lead to a smaller difference.

2.3 Shape by Integration

Now we use the calculated derivatives to recreate the height map.

2.3.1

To compare the results of both path integrations, we first looked at the resulting images of the height maps, but the differences seemed too minute to make out with the eye. Therefore we compared the height of the highest point (center) of the sphere of both methods and found a height of 46.1244 pixels for the column-major order and 46.1059 pixels for the row-major order. This results in a difference of 0.0185 pixels. We took the highest point as a measure of difference, because it is the point in the image with the highest amount of change in the integral and thus has the most potential to display potential errors. The computed difference is not very high, but we also have to take into account that the image and the contained object is very simple and furthermore symmetrical along the diagonal. Later on we will work on images that are more complex and do not have that property. As we will see this leads to a higher difference in height maps.

2.3.2

One way to deal with the differences in column-major and row-major order is to take the average of both. Since in the above case we did not have a large difference between both, taking the average will only change the outcome in neglegible size. We get a average pixel height of 46.1151.

Does the quality of our results change with more training images? When using 25 images we get the pixel heights of 45.8034 for column-major, 45.7795 for row-major and therefore 45.7915 for the average path integration. It is interesting to observe that in this case the difference of 0.0239 is larger then for the case when only using five images. In theory we would expect the quality of our approximation to increase with more training data. In the real world there should be no difference between the generated height maps, no matter what integration path we take so with a better approximation we would also assume smaller differences between column-major and row-major.

2.4 Experiments with different objects

2.4.1

In earlier sections we applied the photometric stereo algorithm to images that had very simple structure, such as the sphere. Those images also had the property of being symmetric along the diagonal, which made it hard to see potential differences in different path integration methods. To better analyze the photometric stereo algorithm we will see if it generalizes well onto more complex images, such as image in figure 8. Looking at the albedo image (figure 9), we see for example around

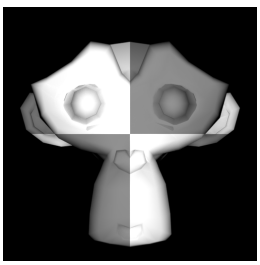

Figure 8: A more complex image

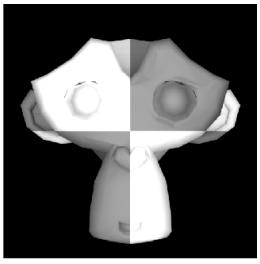

Figure 9: Estimated albedo of the monkey image

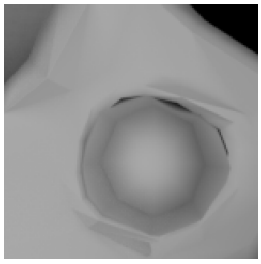

Figure 10: Estimated albedo using all 64 images

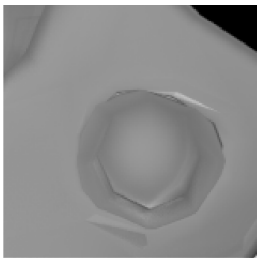

Figure 11: Estimated albedo using selected images

the eyes that the computed albedo is lower than the rest of the gray parts. A reason for that might be the local more complex geometry of the surface. The carved in parts around the eyes lead to shadows for most light sources. This in turn leads to a the assumption that this area is darker and therefore leads to a lower albedo. Mathematically speaking, darker pixel lead to a lower pixel value which in turn leads to a smaller albedo value. To portray our solution to this problem, we will specifically look at the albedo around the eyes. For that we only took a data set of nine images, containing the image with the light source right above the object and those images where the light source was right above the eyes (In the code file those images are contained in the folder "MonkeyGrayEyes"). As we

can see in figure 11 the albedo error is not as strong when using our smaller experimental image set compared to using all images (figure 11). Unexpectedly we also found a better albedo estimate in the mouth area. This finding can be generalized in the following way: When dealing with images that have a complex surface geometry it is advisable to choose the training images wisely. Instead of taking every image available, better albedo estimates can be achieved by using only those training images that focus the lightning on the complex surface areas. In our case one of those areas were the eyes. By iterating over all complex areas one could extract the respective parts of the albedo images and combine them to a better overall albedo estimate.

2.4.2

Until now we only dealt with 1-channel grayscale images. We will now try to generalize the model onto colored 3-channel images. We approach this problem by splitting the 3-channel image into three 1-channel images. To get the albedo and the surface normal we will average over all three channels and for the p and q values we will just add them. As we can see in the resulting normal map we have these horizontal and vertical lines. When comparing the normal map to the original images (figure 12 and 13), we see that those lines occur exactly where we have a change of color, or expressed differently, where in one channel (e.g. the red channel) we change from zero pixel to high-value pixel. This sudden change in pixel-value in a channel leads to a normal map that does not correlate to the original image anymore, even when adding the normal maps together for different channels. This problem is very obvious when looking at the example of the colored sphere. Instead of a smooth normal map we get a very complex normal map that has the same structure as the underlying color changes. Our proposed solution to this issue is the following: Instead of computing the normal map for all three channels and then averaging over them, we transform the colored images into grayscale images and then apply the original 1-channel method. This would prevent zero pixels and therefore also unwanted artifacts described above.

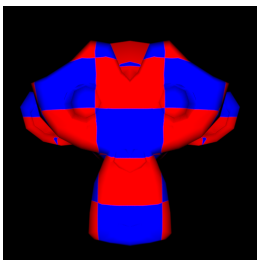

Figure 12: Original colored monkey images

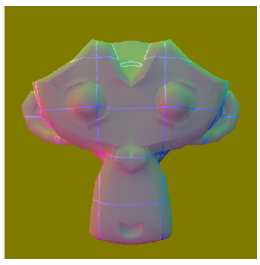

Figure 13: Normal map of colored monkey after averaging over all channels

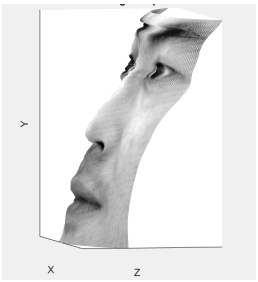

Figure 14: Height map using column-major order

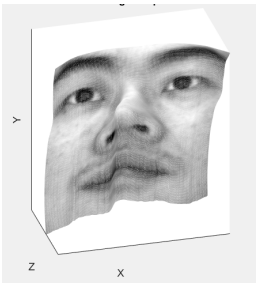

Figure 15: Height map using row-major order

2.4.3

The Yale Face dataset is another example of more complex images. Here we do not have the symmetry along the diagonal of the images and therefore should expect larger differences between column-major and row-major, when compared to the gray sphere. This time these differences are clearly visible. In figure 15 we can clearly make out the recreated height map that uses row-major. We can observe a realistic reconstruction of the height map along the columns of the image and non-realistic reconstruction when looking at the height map along the rows. As a result the right side of the face seems to come out of the image. We have opposite issues when only using column-major order. Here (figure 14) we see that the bottom part of the image seems to be coming out of the image. By taking the average over both path integrations we get a smoother and more realistic height map of a face.

3 Color Spaces

3.1 RGB Color Model

The usage of the RGB color model relies on the Trichromacy theory. According to it, the human eye perceives color primarily through stimulation of cones which are one of two appearing receptors within the retina. More accurately, three pigments are stimulated and combined form a mixture of three colors with respective intensity. The three receptors respond the most for red, blue and green light with corresponding wave length. These colors are referred to as *primary colors* and the range of possibly perceived colors is called *garmut*. In order to map the stimulus values of each color space, a linear function is used. [6]

The key component to a digital camera is a sensor that converts light into electrical charges. As these sensors are only capable of measuring the intensity of the incoming light, filters are drawn on. One possible approach is to divide the light into its primary colors by using one sensor per color. That way, similar to human color vision, the combined intensity values form the finally perceived image. `https://electronics.howstuffworks.com/cameras-photography/digital/digital-camera5.htm`

3.2 Color Space Conversion

First, Figure 16 depicts the *Opponent* color space along with its three channels. Quite contrary to the RGB model, it assumes the human eye to perceive color based on three kinds of opposite colors. In particular, green and red (First channel), yellow and blue (Second channel) and white and black (Third channel), which represents intensity. The RGB model inherits correlation, which means that high values in one channel often correspond to high values in the remaining channels. Image representations in terms of those components complicate the discrimination within the field of object recognition. Decorrelating the RGB values by rotating the original RGB values, as shown in the assignment sheet, yields the opponent space, which is preferable for many Computer Vision (CV) tasks. [6]

The *normalized RGB* model is obtained by normalizing the original RGB channels by their intensity (I = R + G + B). This transformation allows a photometrically invariant representation of an image and is vital to many CV applications. This feature is met, if variations in the imaging conditions, such as variation in color or intensity of the light source or the camera position, do not yield major changes in the color space. Regular RGB modeled pictures, as seen in the original image (Figure 21), are dependent on these conditions. In fact, the color values of each pixel depend highly on source characteristics and object geometry, for instance in the form of shading. The invariance property is clearly visible in Figure 17. Thus, the first image has no shadows, but rather pure color representations and color changes induced by the geometry only occur infrequently in some peppers. This is visible as dark spots on the red peppers. As in the normal RBG model, the channels represent the respective intensities of the primary colors. For example, all red peppers in the first channel are intense (white), the same applies to the green and blue. [6]

Compared to the RGB space, the HSV color space tries to form an approach more intuitive to humans. It describes the way different colors mix together by separating the hue, saturation and value or intensity. The dominant wavelength describes the hue dimension through a radial slice around a central axis of neutral colors. The saturation dimension resembles various shades of brightly colored paint, and thus defines the purity of a color. The value dimension resembles the mixture of those paints with varying amounts of black or white paint. A benefit of using the HSV color space, similar to the opponent model, stems from using it as a mean of preprocessing images for CV tasks, as well as being more intuitive for human use, such as photo editing. [6]

The YCbCr represents colors in terms of one luminance component (Y) and two chrominance components (Cb and Cr). The latter reflect a relative sensitivity of human vision to the blue and red portions of a visible light. This color space is commonly used in the digital video domain. Similar to grayscale, the separation between luminance and chrominance components enables to get rid of redundant color information. Furthermore, only choosing the channel needed for a task, allows a more efficient storage, which is why image and video compression standards like JPEG rely on this color space.

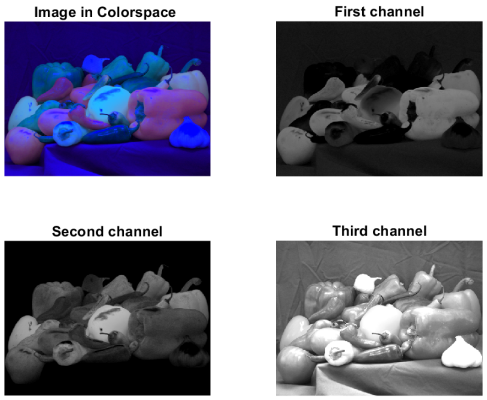

Figure 16: Opponent with Channels O_1, O_2 and O_3.

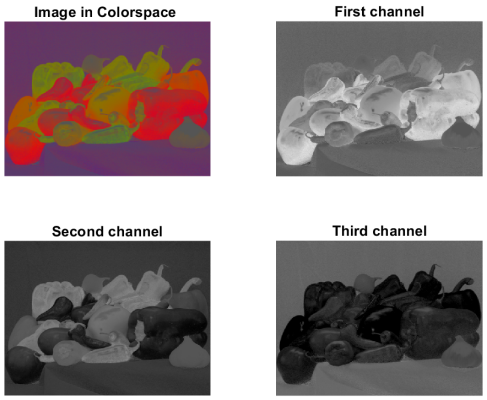

Figure 17: Normalized RGB with Channels r, g and b.

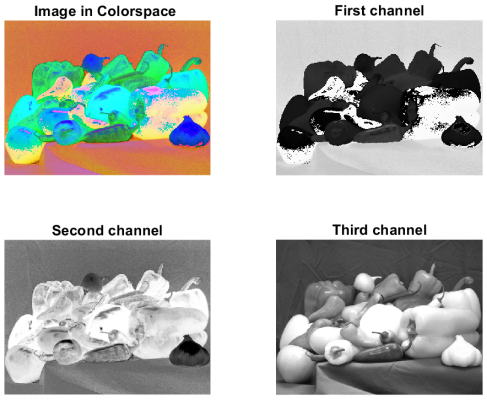

Figure 18: HSV with Channels Hue, Saturation and Value.

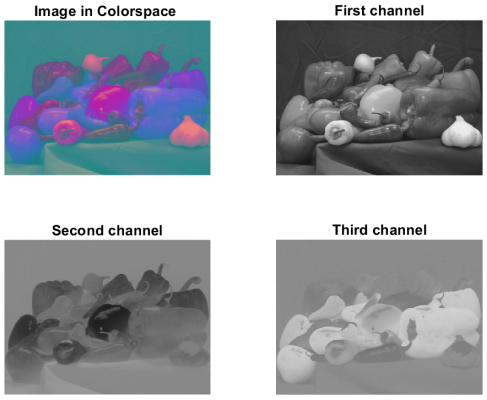

Figure 19: YCbCR with Channels Y, C_b and C_r.

The grayscale space is particularly characterized by only measuring one value per pixel. This value only carries intensity information about the incoming light. Thus, grayscale images are solely composed of shades of gray, with a contrast ranging from black at the weakest intensity to white at the strongest. For some tasks, the color representation does not provide the prominent source of information. For instance, edge detection does not necessarily require color information and thus using grayscale simplifies the process both computationally and perceptionally. The former, as only one value per pixel is looked at, the latter, as colors heavily depend on their representing color space. Figure 20 displays four different methods to come up with the grayscale image along with slight differences among them.

3.3 More on Color Spaces

Another class are the *Perceptually Uniform Color Spaces*. They are motivated by drawbacks, which originate in the RGB or the opponent color space. Namely, they try to overcome the device dependence of a color space, as well as perceptional un-uniformity. The former describes the case, when the values of a color space depend on the device they are displayed on. Perceptional uniformity is a property, according to which numerical distances between two colors also coincide with perceptional differences. One specific example is the *CIElab* color space, which is obtained by nonlinear transformations from RGB. A possible use case is the characterization of colored surfaces and dyes. Furthermore, CV applications, such as image retrieval, that try to align to human vision also benefit of the transformations to CIElab. [6]

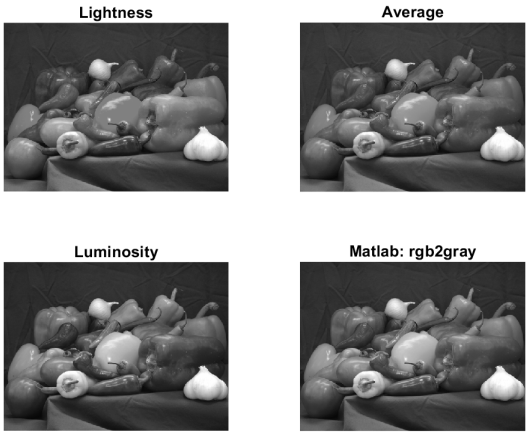

Figure 20: Grays with the four specific methods as titles

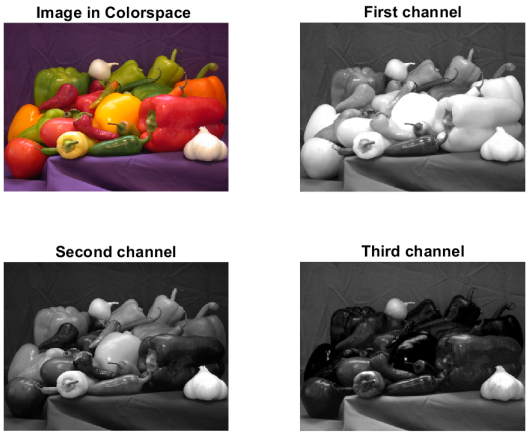

Figure 21: Original peppers image in RGB representation.

4 Intrinsic Image Decomposition

4.1 Other Intrinsic Components

Additionally to the illumination/shading and the reflectance/albedo component, often the so called specular component is looked at. It corresponds to **highlights** that are caused by the viewpoint, geometry and illumination. More accurately, the specular term accounts for highlights in the image, which are caused by light rays that reflect directly off the surface. [9] Removing these highlights from an image benefits object recognition tasks or semantic segmentation, as it allows for a more sophisticated decomposition of an image and thus clearer disregard of redundant information irrelevant to object or scene features. Another component, which is more 'on top of the image' and less intrinsic, are semantic regions. These segmentations are assigned to each object or scene by identifying and semantically grouping pixels. These segmentation components on the one hand benefit from intrinsic decomposition, as semantic regions are separated easier if there are no illumination effects. The other way around, reflectance computation can also benefit from semantic regions, as distinct object or scene labels obtained by semantic segmentation model can guide the intrinsic image decomposition process. [1]

4.2 Synthetic Images

For many CV tasks, such as object detection and stereo, it is vital to have large datasets, which often only need to be annotated by humans in order to be used to train complex models. For intrinsic decomposition however, it is not enough to simply have access to a vast amount of images to annotate, but a tedious procedure needs to be drawn on. This is partly due to the fact that it is hard to acquire ground truth shading and albedo pictures of real world scenes. In fact, simply to obtain a ground truth shading image, the same scene has to be photographed twice, once with the original and once with a grey-painted version of the object, [2] Using synthetically generated images on the other hand allows a much more controlled and reliable environment in terms of illumination conditions, geometric reflections and different viewpoints. Additionally, software tools aid the generation of large training data like the ShapeNet [3] with synthetic pictures for all kinds of illumination strategies (single or multiple illumination sources) or the display of single, as well as multiple objects. This is highly advantageous, as modern deep learning approaches require vast amounts of data. [9] Another reason lies in the fact that real images produce many different likely suggestions for intrinsic decomposition. As an alternative to the costly procedure mentioned above, there are models to come up with handcrafted priors on the reflectance, shape, and lighting conditions found in the natural world. These priors are difficult to tune by hand and therefore have limited scalability, which strengthens the case for synthetic images [8]

4.3 Image Formation

In order to reconstruct an image, given its albedo and shading images, the latter needs to be scaled by dividing over its maximum value (255). Then, element wise multiplication yields the reconstructed image as seen in the bottom left image in Figure 22.

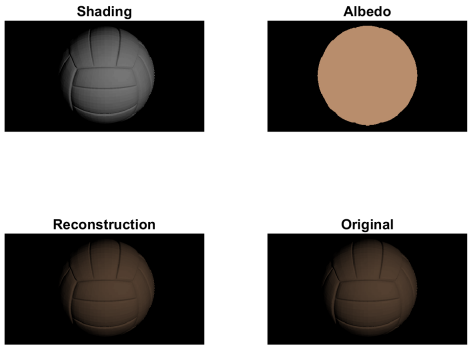

Figure 22: Reconstruction of original image given the albedo and the shading image.

Figure 23: Recoloring of original image to green.

4.4 Recoloring

The true material color of the ball in RGB space is [184, 141, 108]. It can be found by taking the maximum of each color dimension on the albedo image. This is due to the fact that only black and the material color appear in the picture, whereas the black regions are known to be of 0 intensity and thus value 0 and the colored region is given to be of uniform color. Recoloring is accomplished by altering the color values of the albedo image to zero for R and B and full intensity (255) in the green dimension. The result is shown in Figure 23. Even though the objects color is now set to be green, the object does not display a uniformly or in other words pure green surface. This is because the picture is not only composed of the albedo but also of the shading component, as seen in the upper left image in Figure 22. The illumination component accounts for shading effects caused by geometry, shadows and interreflextions. The albedo itself exhibits how the material of the object reflects. [7] As the shadow image is not changed, combining the two yields a surface color that does not appear to be uniformly green.

5 Color Constancy

5.1 Grey-World Algorithm Implementation

We implemented a simple version of the Grey-World Algorithm. As input, it takes an image in RGB format. From that we extract the three matrices corresponding to the three colors and compute their mean values. We divide each matrix by it's mean and multiply with 128, ensuring that the new mean is 128. By concatenating the three matrices back into the RGB format we obtain an image that has average RGB values of $(128, 128, 128)$, which is standard grey. Note that for the sake of simplicity, we do not correct the issue that some pixels may get values higher than 255 by our rescaling. This may be addressed by simply cutting the values at 255, which in turn would result in an average value that is (usually slightly) lower than 128. The results for an exemplary image are shown in figure 24. We see that the light reddish color cast present in the original picture has been removed.

Figure 24: Effect of Grey World Algorithm

Figure 25: Illustration of failure of the Grey World Algorithm

5.2 Example of Failure of the Grey World Algorithm

One assumption the Grey-World Algorithm makes is that everything that is visible in an image is lighted by the same light source, which induces the color cast to be filtered. This assumption may not hold if different parts of an image are subject to diffferent light sources (e.g. inside and outside of a house). Another problem may be a quite monochromatic image (e.g. a bowl full of red fruits), where the algorithm will overadjust. In figure 25 in the third picture on the right, we see an example of this: The grey ball in the foreground obtains a blue-ish color, since the picture contains a lot of red-ish furniture that induces overadjustment.

5.3 White Patch Retinex Algorithm

One Alternative to the Grey World Algorithm is the White Patch Retinex Algorithm as described in [5]. Here, the procedure is similar, but instead of ensuring an average of 128 for each color, one ensures a maximum value of 255 for all of the colors. This is achieved by dividing each color matrix by it's maximum value and then multiplying by 255. This results in a maximum value of 255 per color. The intuition behind the algorithm and it's name stem from the idea, that we may find a pixel that has a "true" color of white and has near maximal values for each of the colors. The difference between it's RGB values and those of pure white, $(255, 255, 255)$, must than be induced by the light source. The algorithm reverses this effect of the light source for the white pixel as well as all other pixels. Additionally, the algorithm ensures that we make use of the whole spectrum of values we have at hand for each color and thus enhance the contrast of the picture.

6 Conclusion

The Photometric Stereo is effective in reconstructing the shape of a surface using multiple grayscale images of the surface under different illuminations. However, it is greatly influenced by the quality and number of the input images as well as by the complexity of the surface. At the same time, generalizing the method, in a simple manner, to work on RGB inputs, lowers the quality of the resulting surface.

Using and understanding different Color Spaces can be useful depending of the processes the encoded image will go through. The RGB Color Model is the most similar to the human eye, but it inherits correlations, making models such as normalized RGB and HSV preferable for CV applications.

Preprocessing the images can make a big difference in the quality of the results when solving CV tasks. One such step is the Decomposing of an image into its components like illumination, reflectance and specularwhich can aid in solving tasks like image recoloring. Another editing step bringing Color Constancy to the image and thus being robust to environmental factors such as different color light sources.

References

[1] A. S. Baslamisli, T. T. Groenestege, P. Das, H. Le, S. Karaoglu, and T. Gevers. Joint learning of intrinsic images and semantic segmentation. *CoRR*, 2018.

[2] S. Beigpour, M. Serra, J. Weijer, R. Benavente, M. Vanrell, O. Penacchio, and D. Samaras. Intrinsic image evaluation on synthetic complex scenes. 2013.

[3] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. *CoRR*, 2015.

[4] F. David A. and J. Ponce. *Computer Vision, A Modern Approach*. 2015.

[5] M. Ebner. *Color constancy*, volume 6. John Wiley & Sons, 2007.

[6] T. Gevers, A. Gijsenij, J. van de Weijer, and J.-M. Geusebroek. *Color in Computer Vision*. 2012.

[7] R. Grosse, M. Johnson, E. Adelson, and W. T Freeman. Ground truth dataset and baseline evaluations for intrinsic image algorithms. pages 2335 – 2342, 2009.

[8] M. Janner, J. Wu, T. D. Kulkarni, I. Yildirim, and J. B. Tenenbaum. Self-supervised intrinsic image decomposition. *CoRR*, 2017.

[9] Z. Li and N. Snavely. Cgintrinsics: Better intrinsic image decomposition through physically-based rendering. *CoRR*, 2018.