Institute of Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3565

Appearance Transfer for Augmented Reality

Ivan Pilipenko

Course of Study:	Informatik
Examiner:	Jun.-Prof. Dr. Martin Fuchs
Supervisor:	Sebastian Koch M. Sc.
Commenced:	October 1, 2013
Completed:	April 2, 2014
CR-Classification:	H.5.1 I.4.1 I.4.6

Abstract

Prototyping is an important phase of designing a product. It involves creating an early model of the final product and evaluating the visual appearance under different lighting conditions with different materials applied. There are two distinct approaches to creating a prototype: creating a physical prototype, or creating a virtual prototype. A physical prototype is usually formed out of clay or Plasticine, resulting in an untextured model. This model only conveys the form, but omits the actual material, not showing the final appearance of the object under different lighting conditions. A virtual prototype approach consists of a 3D geometric model combined with image synthesis algorithms, offering a convincing textured preview of the final product. This approach allows to quickly apply different textures, but is limited by the selection of materials available.

We propose a new method which combines the advantages of the physical and virtual approaches, allowing to directly transfer the material of a physical reference object to another. Our goal is to design a method which is fast enough to allow relighting the source object and see the lighting dynamically change on the destination object in real time. We introduce a setup which is portable and only consists of components that are widely available. This allows the user to easily transport the setup to new locations, with the setup not requiring much time to be assembled.

Contents

List of Figures

List of Tables

1. Introduction

Prototyping is an important phase of designing a product in many different branches of the industry. It involves creating an early model of the final product and evaluating the visual appearance under different lighting conditions with different materials applied.

There are two distinct approaches to creating a prototype: creating a physical prototype, or creating a virtual prototype. The former is an approach which feels natural, allowing to inspect the prototype under real light conditions and from arbitrary directions. A physical prototype is usually formed out of clay or Plasticine. The result is usually an untextured model. This model only conveys the form, but omits the actual material, hence not showing the final appearance of the object under different lighting conditions. Painting the model by hand does not work for many materials, and is also time consuming as well as impractical if one desires to quickly test different materials.

The latter, a virtual prototype approach, consists of a 3D geometric model combined with image synthesis algorithms, offering a convincing preview of the final object. A virtual approach allows to apply any texture at a click of a mouse button, but is limited by the selection of materials available. Acquiring new materials for the model often requires a stationary or expensive setup, or sometimes even both. The acquisition process also tends to take a lot of time with most techniques.

We propose a new method which combines the advantages of the physical and virtual approaches, allowing to directly transfer the material of a physical reference object to another. Our approach consists of several methods to calculate the correspondences between the source and destination objects. These correspondences can then be used to map the pixels of the destination object to the source object, thus displaying the texture of the source object on the destination object. The user can then proceed to relight the source object and see the illumination changing accordingly on the destination object.

Our goal is to design a method which is fast enough to allow relighting the source object and see the lighting dynamically change on the destination object in real time. We introduce a setup which is portable and only consists of components that are widely available. This allows the user to easily transport the setup to new locations, with the setup not requiring much time to be assembled.

The thesis is structured as follows: first, we will discuss prior work done in the field, describing its relation to our thesis. Then, we will cover the concepts important for this thesis, such as calculating the normals of our objects and search algorithms used to calculate the distance between a point on the source object and a point on the destination object.

Following that, we will describe our hardware setup, our method to acquire the evaluation data used for the calculation of correspondences between the objects, as well as our implementation of the actual calculation algorithms. The same chapter will also cover the improvements we made to make our methods more robust.

In the next chapter, we will evaluate our results as well as discuss the problems encountered while working on this thesis and the decisions we made to solve them. We will also present selected benchmarks comparing the runtimes of methods at different resolutions and settings. The same chapter will also offer insights into mapping quality of each algorithm, as well as show a few examples of objects relighted using novel light positions. To conclude the chapter, we will offer a few words on the time the whole process takes from acquisition to being able to see the results of the appearance transfer under dynamic lighting conditions.

The last chapter offers a summary of the thesis as well as possible future work based on the methods presented in this thesis. The appendix contains additional results created for different objects under different lighting, followed by acknowledgments pertaining any foreign material used in the thesis

2. Related Work

This chapter will touch upon prior work done in the related fields. The work we see as related can be classified into three fields: *Material Acquisition/Correspondence Generation*, *Search Algorithms* and *Augmented Reality*. In the three sections below, we will explain the essence of each related work and show how it is connected to our thesis.

2.1. Material Acquisition/Correspondence Generation

There has been a lot of related work done on the field of acquiring a sample material and transferring it to a 3D model. One of the methods to achieve this is by sampling the bidirectional reflectance distribution function (BRDF) of the object. Goldman et al. [GBCHS10] attempt to reconstruct the BRDF of an object by exploiting the fact most objects consist of multiple so-called base materials. Combining these base material results in a correct BRDF. This allows the objects to be re-rendered under a novel lighting while maintaining a realistic appearance. Oxholm et al. [ON12] proposed a method of reconstructing the BRDF and the geometry of an object without relying on controlled illumination as traditional Photometric Stereo approaches do. The work assumes known, but uncontrollable illumination and reconstructs the BRDF and the geometry by joint simulation of both.

We decided that reconstructing the BRDF is too exact and also requires a complex setup. Choosing a simpler approach should still deliver believable results while reducing complexity. This lead us to the Photometric Stereo approach. Photometric Stereo was first covered by Woodham [Woo80], proposing a method for acquiring the normals of the object by taking multiple photos with a fixed view and differing lighting using multiple light sources, reconstructing the normals from the intensity information varying between the different photos. This approach relies on a controlled light setup, which poses a restriction, but is relatively simple to implement.

Basri et al. [BJK07] presented a photometric approach which does not rely on a controlled light setup, making it more suitable for everyday use. The work assumes convex, nearly Lambertian objects. The reconstruction is based on spherical harmonics, making the implementation more complex.

Tingdahl et al. [TGG12] use a traditional approach to Photometric Stereo but enhance it with an automatic decomposition of an object into its base materials. One example shown is a strawberry, which is decomposed into the seeds, the leaf, and the fruit's specular and Lambertian base materials. This approach is more exact than the traditional Photometric Stereo, but it requires a complex setup consisting of a 260 LED dome, which speaks against pursuing this route in our case.

Aliaga et al. [AX10] proposed a self-calibrating method for photometric acquisition using structured light. Their setup allowed for three ways of reconstruction: geometric, photometric or a combination of both. A special aspect of the work was to use hardware components available to the masses, specifically a projector and an uncalibrated camera. This made the approach interesting for us. However, the acquisition process takes up to 15 minutes per material, which we deemed to be too slow. In the end, we decided to go with the traditional Photometric Stereo approach to maximize the acquisition speed while still retaining an acceptable degree of accuracy by increasing the light sources count.

2.2. Search Algorithms

In order to be able to map between the source and destination objects, we need a reasonably fast method of finding the closest match for each point on the destination object. Muja et al. [ML09][ML12] describe a fast approximate nearest neighbor approach which resulted in the *FLANN* library[1]. This library proved to be fast enough for our needs.

Another way to calculate correspondences between two images called *Patch Match* was described by Barnes et al. [BSFG09]. The algorithm matches square patches between images. It consists of several stages, starting with an initialization with random guesses, followed by several iterations to improve the guess as well as a random search stage to improve the guess even further. This appeared to be perfect for our needs, so we decided to adapt the algorithm according to our needs.

One more approach appeared to be suited for our needs: *Region Growing* presented by Adams et al. [AB94]. The algorithm segments an image into distinct regions, with seeding points supplied by the user figuring as starting positions of each region. We have adapted the idea and implemented our own variation as one of the attempts to generate correspondences.

[1] http://www.cs.ubc.ca/research/flann/

2.3. Augmented Reality

To look for a method to display the results of our texture transfer, we decided to look into the field of Augmented Reality. Bradley et al. [BRB09] has presented a method to paint a video on a moving piece of cloth in realtime, with correct lighting applied to the scene. The work uses black and white coded ring markers to track the cloth, then impaints the scene and superposes the video on top of it.

Since we assume a fixed view and a static object with only the lighting being dynamic, we had to look for methods better suited for our needs. Umakatsu et al. [UMKT12] introduced an Augmented Reality interface which allows the user to touch the source object in a scene to select its material, then touch the destination object and hereby transfer the texture between two objects. To transfer the texture, they first generate the texture by unwrapping the 3D mesh of the object. This is done utilizing Least Squares Conformal Maps by Lévy et al. [LPRM02]. The texture is then decomposed into regions of interest by an approach inspired by cross-boundary brushes by Zheng et al. [ZT10]. The texture then can be transferred between the region on the source and the region on the destination object. Another work by Umakatsu et al. [UMKT13] based on that further expanded the functionality, allowing the user to also pinch the texture before transferring it to the destination object.

3. Basic Concepts

This chapter covers the basic concepts used in this thesis. This includes a rough overview of Photometric Stereo, used to calculate object normals, as well as linear, FLANN and Patch Match search algorithms, used to calculate the correspondences between two points of the source and destination objects.

3.1. Photometric Stereo

Photometric Stereo is a method used to derive the orientation of each point of an object from a set of images. The first work to deal with Photometric Stereo was Woodham's "Photometric method for determining surface orientation from multiple images" [Woo80]. The method requires a set of pictures of an object under differing light conditions to deliver accurate results. It assumes one camera with a fixed viewpoint, as well as distant lighting. This means the intensity and direction of the light is assumed to be the same for every point of the object.

The normals can be reconstructed by observing the varying intensities of a point when lit by different light sources, with one light source active at a time. This means we require as many photos per set as we have light sources. As per findings of the paper [Woo80], two light sources are sufficient to derive the normals of the object, provided neither of the light sources are coplanar with respect to the azimuth. Additional light sources help to overdetermine the resulting non-linear equations and lead to more reliable results.

The system of equations to derive the normals is given as follows:

$$I = L \cdot N$$

$$\begin{pmatrix} I_0 \\ I_1 \\ \vdots \\ I_n \end{pmatrix} = \begin{pmatrix} L_{0,0} & L_{0,1} & L_{0,2} \\ L_{1,0} & L_{1,1} & L_{1,2} \\ \vdots & \vdots & \vdots \\ L_{n,0} & L_{n,1} & L_{n,2} \end{pmatrix} \cdot \begin{pmatrix} N_0 \\ N_1 \\ N_2 \end{pmatrix}$$

I is a vector of intensities, each component corresponding to lighting under one of the light sources. L is a 3×n matrix consisting of transposed vectors representing the light sources' position. Each line represents one light source. N is the vector representing the unknown normals. Solving this system of equations gives us the normals we are looking for.

3.2. Search Algorithms

3.2.1. Linear Search

Linear search is the most naïve approach, which iterates over all pixels until it finds a match when searching for an exact match. A slight modification of the linear algorithm iterates over all pixels, minimizing the distance between the query and the current point. The calculation of the distance is done component-for-component at first, summing up the individual distances to acquire the distance for this point. This approach is very slow due to its quadratic runtime in relation to the image size. In our evaluation (see Section 5), we show that it is not viable to use this approach with our data sets.

3.2.2. FLANN: Fast Library for Approximate Nearest Neighbors

FLANN is a very fast approximate nearest neighbor search algorithm, which has its origins in the two papers by Muja et al.: “Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration” [ML09] and “Fast Matching of Binary Features” [ML12].

The algorithm utilizes several *kd-trees*. A kd-tree is a k-dimensional tree structure where each non-leaf node defining a hyperplane which subdivides the points into two half-spaces: *to the left* from the hyperplane and *to the right* from the plane. Each further non-leaf node either to the left or to the right of the previous node further subdivides the space, with the spaces getting smaller and smaller until one decides to stop the subdivision. This structure enables to locate any given point in linear time in the worst case, but logarithmic times can be achieved in practice. This is significantly faster than the linear approach described above.

Additionally, FLANN creates multiple kd-trees to parallelize the search. The library has innate multi-threading support, so no additional work is required to accelerate execution even further. This results in a runtime which is “orders of magnitude faster than the algorithms performing the exact searches” according to its documentation[1], while still delivering accurate results. The dimensionality of the kd-tree is determined by the dimensionality of the points.

[1] http://www.cs.ubc.ca/research/flann

To find approximate matches, FLANN uses the L2 distance measure, also known as the Euclidean distance, in the default case. The L2 distance between two points p and q is defined as follows:

$$d(p,q) = \sum_{i=0}^{n} \sqrt{(p_i - q_i)^2} = \sqrt{(p_0 - q_0)^2 + (p_1 - q_1)^2 + ... + (p_{n-1} - q_{n-1})^2 + (p_n - p_n)^2} \tag{3.2}$$

Further, the FLANN algorithm builds a list of all points, only passing a *subset* of these points to the function which builds the actual k-dimensional searching structure. The point picking algorithm is very efficient, maintaining a high accuracy of the results.

3.2.3. Patch Match

The Patch Match algorithm was created as a part of the paper titled "Patch Match: A Randomized Correspondence Algorithm for Structural Image Editing" by Barnes et al. [BSFG09]. The algorithm uses a randomized nearest neighbor approach, which takes two images as input, then proceeds to find correspondences by minimizing the Euclidean distance between a *patch* in image A and another patch in image B. A patch is defined as a square area, with the pixel position defining the upper left corner of the patch. The basic implementation we use supports matching translating patches *only*, meaning it cannot find an optimal match if the matching patch is rotated or scaled. However this does not pose a problem for us, because the quality of the algorithm is high enough to deliver good results. The algorithm itself is iterative and the paper has shown that the results converge after only a few iterations, making the algorithm very fast.

Figure 3.1 illustrates the search process described below. The first phase of the algorithm is the *Initialization*: The correspondence map is initialized with random values. The function then proceeds to iterate over all pixels that have the patch still lying within the image boundaries, calculating the distances between the current patch and the randomly assigned position of the other patch. In the next step, the program attempts to improve the guess by "shaking" the currently assigned patch position – the *Propagation* stage. This means that for iterations with an even number, the patch position is shifted to the left and to the top. For iterations with an odd number, the patch position is shifted to the right and down. This is the stage where the results converge with each iteration.

Another attempt to improve the guess is the random *Search* stage, in which the algorithm is searching in boxes of exponentially decreasing size around the current best guess. In the end, we obtain a map of correspondences as well as their distances for each pixel in the image. The results are accurate, even with as few as *five* iterations in the *Propagation* stage.

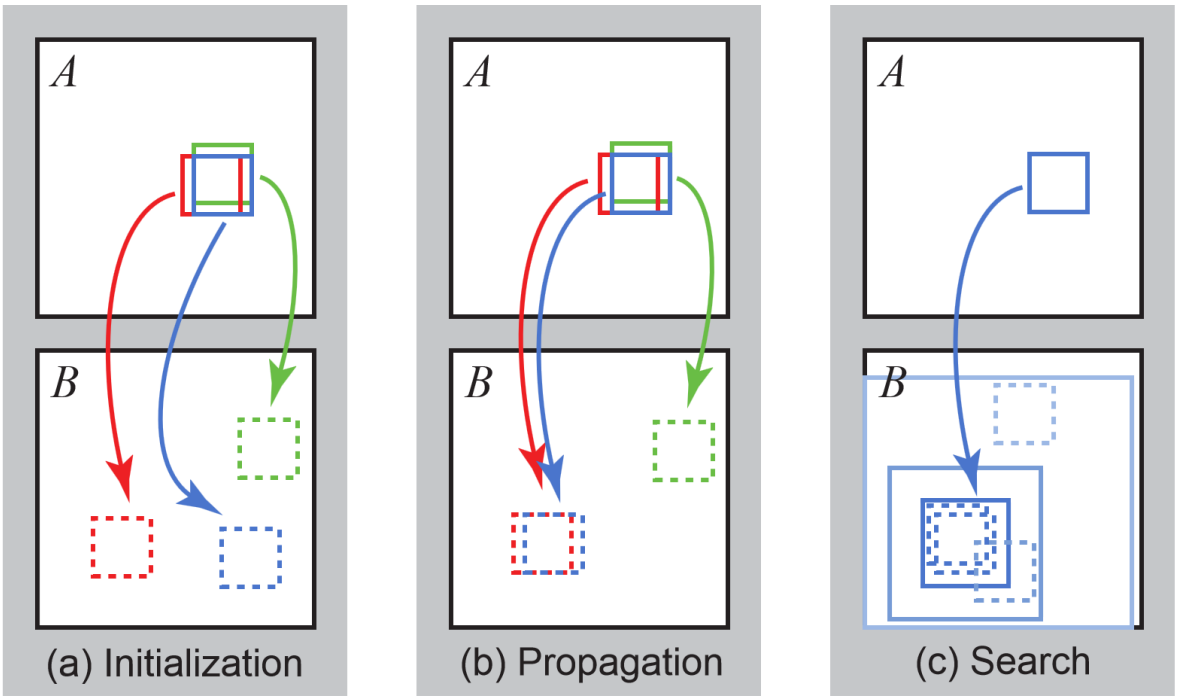

Figure 3.1.: Phases of the randomized nearest neighbor algorithm: (a) patches initially have random assignments; (b) the blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, propagating good matches; (c) the patch searches randomly for improvements in concentric neighborhoods. Image and description taken from [BSFG09]

4. Setup and Implementation

In this chapter, we will provide an overview of our workflow, detailing the hardware setup, the process of acquiring the data sets needed for calculating the similarities between objects, as well as the methods used to calculate the correspondences between the source and the destination object and perform the associated mapping.

4.1. Physical Setup

One of our goals was for our setup to only use parts widely available. Therefore, no specialized or overly expensive components are required. Our setup consists of a Canon 5D Mark II camera for photo and video capture and a chipboard which can hold up to eight light sources. The chipboard is 8 mm thick and measures 1x1 m and is mounted on the tripod along with the camera, as shown in Figure 4.1. By doing so, the setup is centered with respect to the camera sensor, allowing to easily adjust the camera direction and keep the orientation of the light sources and the camera sensor synchronized. The chipboard also has several sets of holes to allow adjusting the chipboard's relative position to the camera sensor. This allows to use different cameras with the same setup with minimal adjustments.

The chipboard holds eight E27 sockets, which are fitted with seven *Philips Hue*[1] lamps. The sockets are positioned in a circle with a radius of 0.47 cm around the approximate center of the camera sensor. The bottom socket is not used, because the light from it would be obstructed by the tripod. Each socket is equipped with its own switch to allow manually switching the light sources on or off if desired, for example to restrict the number of light sources.

The Philips Hue lamps are calibrated RGB LED lamps, with the deviation between the brightest and darkest of our seven lamps being 0.7%. Each lamp can communicate with the base station over Wi-Fi. The base station allows setting different parameters of each individual light bulb: the color (either as RGB values, from 0 to 255, or as xy-coordinate, from 0.0 to 1.0 in the CIE 1931 system[2]), the color temperature in *reciprocal megakelvin*[3], (from 153 to

[1] http://www.meethue.com/
[2] http://en.wikipedia.org/wiki/CIE_1931_color_space#CIE_xy_chromaticity_diagram_and_the_CIE_xyY_color_space
[3] http://en.wikipedia.org/wiki/Mired

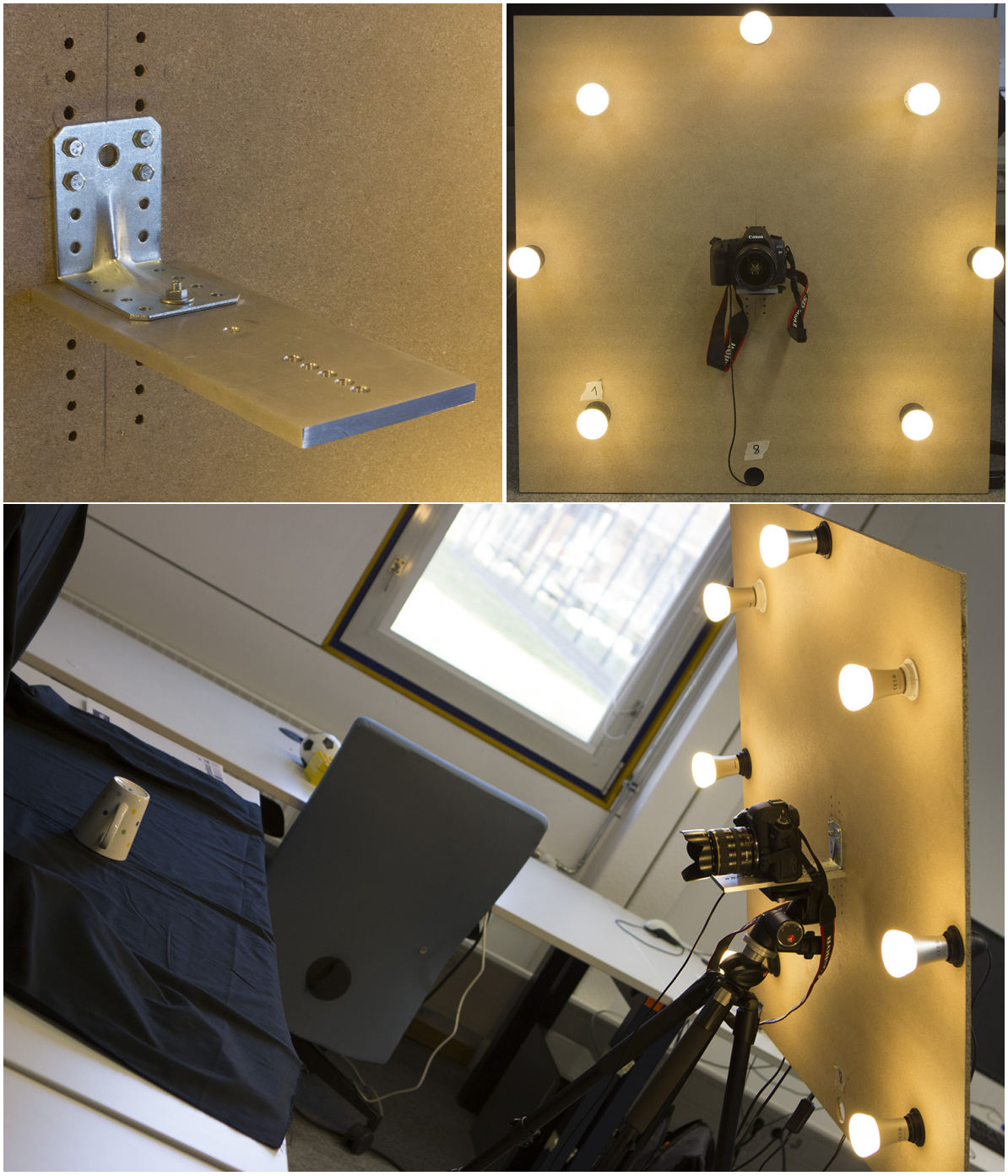

Figure 4.1.: Our setup: closeup of the camera mount (upper left), the chipboard with the eight light sockets and seven lights mounted (upper right) and the fully assembled setup on a tripod, ready to take photos of the object on the table.

500, corresponding to 6500K to 2000K respectively) and the brightness (from 0 to 255). Each instruction has a minimal latency of 100 ms, however this only holds if all instructions for all lights are sent to the base station as one JSON[4] object. Otherwise, each JSON object will be processed sequentially, each change taking 100 ms before the result is applied. Sending instructions can either be done via the web interface, by writing each JSON object by hand, or by writing a program using the API provided[5]. We decided to do the latter, as it allows for much more flexible control and scripting.

4.2. Implementation

This section describes the implementation of our algorithms. The Data Capture and Light Control Section 4.2.1 describes how the image information required is captured. The Calculation of Correspondences Section 4.2.2 covers our methods to calculate and map correspondences between the source and target image sets. The last Section 4.2.3, Repaint and Display, describes the process of applying the calculated correspondences to the video feed of the source object to be able to display lighting changes on the destination object.

4.2.1. Data Capture and Light Control

To be able to map texture data between two objects, we first need to acquire a set of images for each object. Each set consists of seven photos, each photo taken with one light source active at a time. Each of the photos of a set have to be taken with the same aperture and exposure time, as well as ISO[6]. This satisfies the conditions needed to calculate the normals of the object using Photometric Stereo, as described in Section 3.1. That said, the aperture and exposure can differ between different sets, which allows to choose values appropriate for the object to be acquired.

The camera is connected to the computer via USB. The camera response curve is assumed to be linear. A simple script is used to take the photos utilizing *gphoto2*[7] and to convert the raw image into the *ppm*[8] format using *dcraw*[9]. The same script also controls the application in charge of controlling the lights. The user can modify the script behavior by supplying the aperture and exposure arguments. ISO is fixed at 100 in order to minimize noise. Since the camera is fixated on the tripod, the resulting longer exposure times do not pose a problem.

[4] http://json.org/
[5] http://developers.meethue.com/
[6] en.wikipedia.org/wiki/ISO_(photography)
[7] http://www.gphoto.org/
[8] http://netpbm.sourceforge.net/doc/ppm.html
[9] http://www.cybercom.net/~dcoffin/dcraw/

To be able to use our correspondence map without having to correct for the resolution differing between photo and video mode, we resize our captured data to match the resolution of the camera in video mode. Using our Canon EOS 5D Mark II, *gphoto2* delivers a video resolution of 1024×680. The photos taken with the camera have the same aspect ratio, meaning we can resize the photos without having to correct for the aspect ratio.

Each object is placed one meter away from the camera sensor, with the position of the camera sensor being indicated by the marks on the camera body. The object is photographed on a dark background, in our case a piece of black cloth. We require the destination objects to be captured first, and the source object to be captured last, as *moving the source object after capture would invalidate the calculated correspondences*.

Considering our objects' sizes are much smaller than their distance from the camera sensor, we can assume distant lighting. This means we assume the intensity and direction of the light is the same for every point of the object. The objects we selected are nearly Lambertian, the form being both convex and concave. We assume there are no global lighting effects like indirect illumination and that our seven light bulbs are the only light sources in the scene. This means we have to darken the room as much as possible.

4.2.2. Calculation of Correspondences

Before we can calculate the distances and find correspondences, we have to be able to separate the object from the background. This is important to prevent the mapping of pixels which are part of the background to be displayed as part of the destination object. To achieve this, we ask the user to specify a segmentation threshold for both the source and the destination objects. The user is also asked to select the object on the photo or specify the coordinates via command line to further limit the working area. To determine whether the current point should be added to the working set, we sum up the components of the *intensity vector* representing the intensities of each image of a set and only add the position, if the resulting value is above or equal to the segmentation threshold.

The mapping step calculates the correspondences between two points of the objects using several different methods. These correspondences will then be used to transfer the pixels from the source object to the destination, thus transferring the texture and the lighting of the source object. These methods can be classified by the way they build the correspondence map into three categories: *direct*, *Patch Match* and *Region Growing*. Each method minimizes the Euclidean distance between either normals or intensities to find the best correspondence for a query. Intensities are defined as the *normalized sum of R, G and B values* of a pixel:

$$I(x,y) = \frac{R(x,y) + G(x,y) + B(x,y)}{3}$$

We use different criteria to minimize the distance by. Our first approach is calculating the Euclidean distance between the source and the destination *normalized* vector of intensities, where each component stands for the intensity of one of the seven photos calculated from the RGB value at the current position. Equation 4.1a is used in this case, with $d_I(x, y)$ describing the distance over intensities for the current xy-coordinates, I_{s_i} and I_{d_i} being the *i-th* component of the *s*ource and *d*estination intensity vectors respectively and *n* being the number of lights. Dividing by the sum of intensities performs the normalization.

Another approach is to calculate the Euclidean distance between the source and the destination normals as shown in Equation 4.1b, $d_N(x, y)$ being the distance over normals for the current xy-coordinates, N_{s_i} and N_{d_i} being the three components of the *s*ource and *d*estination normal respectively. Normals are assumed to be already normalized, so no additional normalization is needed in this case. One approach would be to use angle distance for normals, but angle and Euclidean distances are equivalent for normals, so we decided to use the Euclidean distance for uniformity reasons.

To increase robustness of the match, we not only use the current pixel information, but extend it to include the neighbor pixel information. The resulting Equation 4.1c shows the calculation of the distance for a patch of intensities or normals.

(4.1a)

$$d_I(x,y) = \sqrt{\sum_{i=0}^{n} \left(\frac{I_{s_i}(x,y)}{\sum_{j=0}^{n} I_{s_j}(x,y)} - \frac{I_{d_i}(x,y)}{\sum_{k=0}^{n} I_{d_k}(x,y)} \right)^2}$$

(4.1b)

$$d_N(x,y) = \sqrt{\sum_{i=0}^{2} (N_{s_i}(x,y) - N_{d_i}(x,y))^2}$$

(4.1c)

$$d_P(x,y) = \sum_{x_p=x-\lfloor \frac{p}{2} \rfloor}^{x+\lfloor \frac{p}{2} \rfloor} \sum_{y_p=y-\lfloor \frac{p}{2} \rfloor}^{y+\lfloor \frac{p}{2} \rfloor} d_m(x_p, y_p) \quad | \; p \in \{1,3,5,7\};\; m \in \{I, N\}$$

The most basic set of methods maps correspondences per pixel. Each pixel from the destination object which lies above the segmentation threshold is used as a query for the matching function. The program first iterates over all pixels of the segmented *source* object image set to build and initialize the FLANN search structures (see Section 3.2). The next step consists of iterating over all pixels of the segmented *destination* object image set and using each valid pixel as a query, which is then passed to the search function.

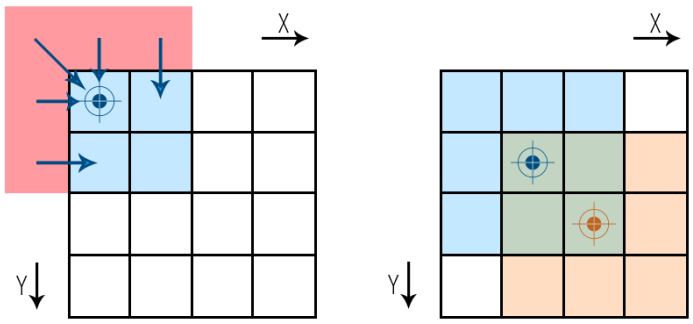

Figure 4.2.: Left: illustrates picking mirrored pixels when neighbors are out of bounds. Right: illustrates two overlapping patches, with center pixels acting as neighbors for the other patch.

The result of the search function represents the correspondence between the two objects at this pixel position. It is written to an output image, the xy-position in the image representing the location on the destination object (current query), while the red and green components hold the x and y location on the source object respectively. The blue component is not used, but we fill it with a value of 1.0 for visual consistency.

To be more robust we extended our methods to match *patches* of a size of 1×1, 3×3, 5×5 or 7×7 pixels instead. The process stays essentially the same as with the pixel-based method described above. In case of border pixels, we supplement the missing pixels by mirroring. The difference is that if a pixel passes the check, its neighbors are also added to the same entry vector. This means that for a patch of a size of 3×3 pixels, the entry vector has the size of $7 \cdot 3 \cdot 3 = 63$ components. The approach leads to overlapping patches, so that each pixel's intensities and normals are used by more than one patch, each time appearing at a different position in a patch. Figure 4.2 illustrates the overlap.

Depending on the object, we run into the problem of more than one point on the source object qualifying as a match for a point on the destination object. In the worst case, for example choosing a cube as the source object, the algorithm will always pick one and the same point of a side, because it is sequentially the "first" match our algorithm finds. Other positions on the side of the cube will not improve the match, because they have exactly the same distance between the query and the current position on the cube.

To work around that, we save a *list* of potential matches sorted by ascending distance, then pick the entry which is "*locally closer*" to the query position in the image. Figure 4.3 illustrates the process of picking a candidate. "*Locally closer*" is defined as follows: first, we translate the matched pixel's image coordinates to destination object's coordinates using Equations 4.2. f_{cor} stands for the correction factor to be applied to the x-coordinate of the source (x_{src}) or the destination (x_{dst}) object, f'_{cor} being the correction factor for the y-coordinate.

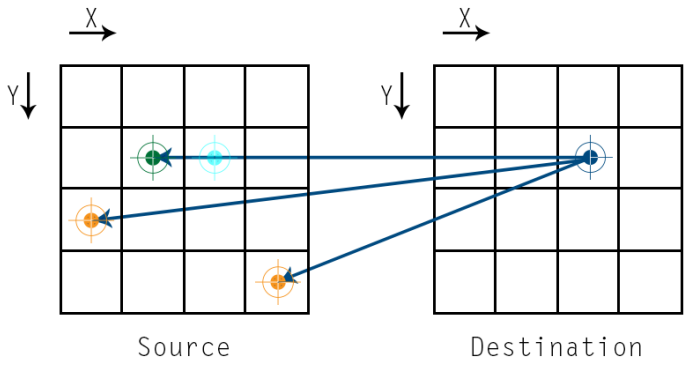

Figure 4.3.: Picking the right candidate: the query (blue) returns several matches. Transferring the query to source coordinates (cyan) results in green point being picked as match, as orange ones are further away.

Once we have translated the coordinates, we subtract the x-coordinate of the potential match from the query to get the distance on the x-axis. We repeat the same for the y-coordinates, then add both up to get the total distance between the pixels. The potential match with the smallest distance between itself and the query is then picked.

The intention is to prevent the algorithm from getting ”stuck“ at first match, thus allowing it to actually transfer the texture in case of multiple matches. The reasoning behind this approach is that points, for example in the middle of the destination object (as captured on the image) should most likely be mapped to the ones near the middle of the source object to correctly transfer the texture.

$$\begin{aligned} f_{cor} &= \frac{x_{src}}{x_{dst}} & f'_{cor} &= \frac{y_{src}}{y_{dst}} \\ x_{dst} &= \frac{x_{src}}{f_{cor}} & y_{dst} &= \frac{y_{src}}{f'_{cor}} \\ x_{src} &= x_{dst} \cdot f_{cor} & y_{src} &= y_{dst} \cdot f'_{cor} \end{aligned} \tag{4.2}$$

We have also adapted the Patch Match algorithm[10] described in Section 3.2.3 according to our needs. The most important change was to account for segmentation. This involved not allowing coordinates containing ”empty“ normals to be randomly matched to valid normals and vice versa. Another change we had to do was to define the distance to an ”empty“ normal as maximal, otherwise such a match was deemed to be optimal in some cases. The Patch Match algorithm returns a complete correspondence map.

[10] http://gfx.cs.princeton.edu/pubs/Barnes_2009_PAR/index.php

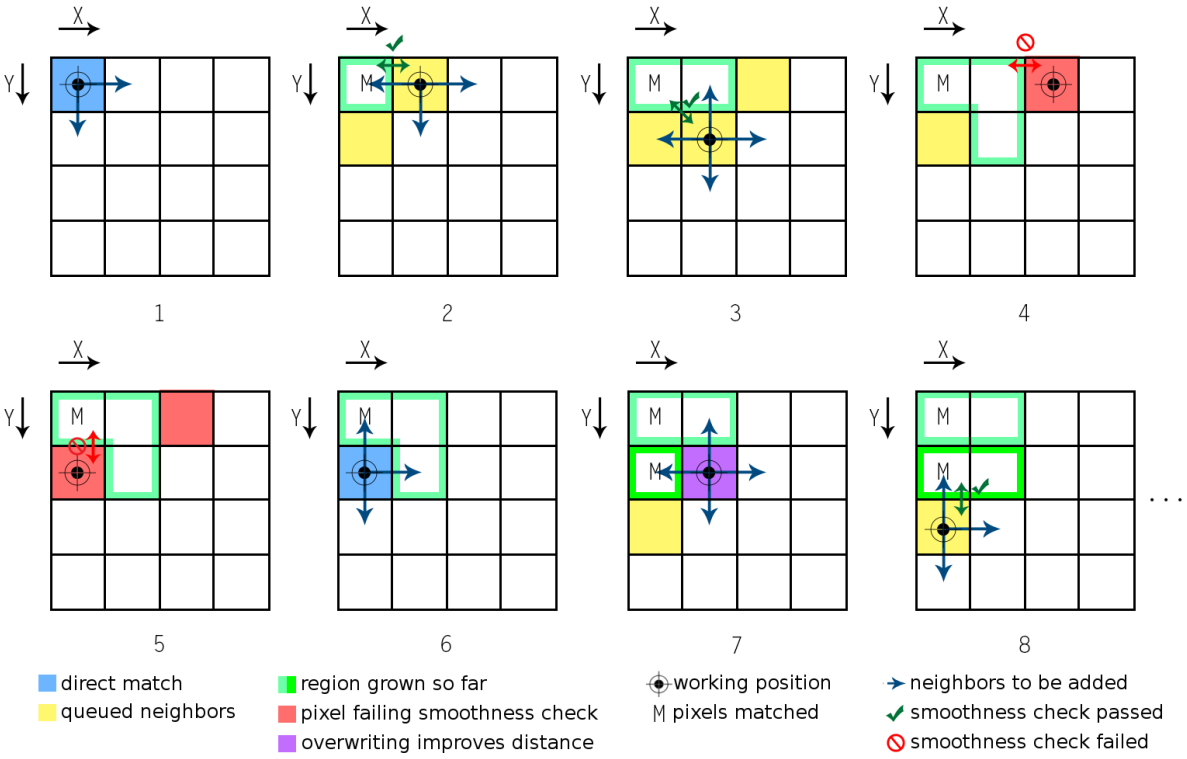

Figure 4.4.: Region Growing: 1) Finding direct match and queuing neighbors.
2,3,8) Passing neighbor distance check, growing region.
4,5) Failing neighbor distance check, stop growing.
6) Skipping to next unfilled pixel, finding direct match again (new region).
7) Neighbor check shows overwriting improves distance.

Another method to improve the texture transfer is the Region Growing algorithm. The algorithm iterates over all pixels of the *destination* object and consists of two phases. Each pixel can have three states: *empty*, *matched* and *filled*. *Empty* means that the pixel has yet to be mapped. *Matched* means that the pixel was successfully mapped using the matching function utilizing either Equation 4.1a, 4.1b or 4.1c. *Filled* means that the pixel was mapped by the neighborhood checking algorithm. Figure 4.4 illustrates the Region Growing process.

In the first phase, *seeding*, we check whether the current pixel does not have a correspondence mapped yet, meaning the result image at that position is *empty*. If that is the case, then we search for the optimal match using either Equation 4.1a, 4.1b or 4.1c. Otherwise, the pixel is skipped. Once the match is found, it is written to the correspondence map, setting this position's state to *matched*. Following that, the four neighbors (above, below, left, right) are added to a set of pixels to be filled via Region Growing.

This set is then passed to the Region Growing function, along with the current *matched* pixel position, an array holding the state of the pixel (*empty*, *matched*, or *filled*) as well as an array holding the current distances between the pixels. This corresponds with step one on Figure 4.4 and concludes the first phase.

The second phase consists of a series of checks to determine whether each of the points currently in the set of neighbors can be filled without searching, thus growing the region. First, the function checks if the current neighbor pixel is *empty*. If it is, then we have to check whether it is safe to *fill* the current pixel.

For this purpose we make three checks. Failing any of these will result in the cancellation of the growth process in the current direction. First we check if the distance between the *parent* pixel and its neighbor pixel is below the defined threshold. This would mean the neighbor pixel is a part of a smooth region on the *source* data set. We also have to check if the pixel we are about to fill is part of a smooth region on the *destination* data set. We further constrain the growth process so that the distance between the current neighbor on the *source* data set and the current position on the *destination* data set is below the defined threshold as well.

These three constraints guarantee that only pixels which are a part of a smooth region are added, but only if the added pixel does not break the smoothness of the region (with respect to intensity or normal changes) on the destination object. If the pixel can be *filled*, then its neighbors are added to the end of the set of neighbors (see steps two and three on Figure 4.4). Otherwise, we have reached the end of the smooth *region*, so adding the neighbors is not needed (steps four and five on Figure 4.4). In both cases the function then proceeds to work on the next neighbor in the set.

If the current neighbor pixel was not empty, we check whether the pixel was *matched* or *filled*. If it was *matched*, we can skip it, as we will not be able to improve our match any further. However, if the pixel was already *filled* by a previous pass of the algorithm, then we can check whether we can improve our estimate. To do this, we check if filling with a new neighbor decreases the distance between the destination pixel and the current neighbor.

If this is the case, we overwrite the previous result with our new one, then proceed to check the rest of the neighbors (see step seven on Figure 4.4). Otherwise, we can safely cancel growing the region in that direction. The Region Growing function continues iterating through the neighbors until all of them have been checked. Once this is the case, the algorithm returns to phase one and skips to the next empty pixel of the correspondence map. This continues until all positions have either been *filled* or *matched*.

4.2.3. Repaint and Display

In this section we will describe the process of applying the correspondences and displaying our results to the user. The camera is switched to video capture mode, with the source object still being at the same position. This is important, since moving the object would change its position in the picture, invalidating the calculated correspondences.

The program loads the generated correspondence map, then displays the source and destination objects side by side. As we mentioned in Section 4.2.1, the aspect ratio of the video and the correspondence map are the same. Since we have also resized our data sets to match the resolution of the video, no further adjustments are needed to be able to transfer the appearance of the source object to the destination object using the generated correspondence map.

The destination object is repainted by taking the pixels of the video capture and transferring them to the position on the destination object according to the correspondence map. The user can now move the light around the source object and see the light change accordingly on the repainted destination object on the computer monitor.

5. Evaluation

In this chapter, we will perform an evaluation of different aspects of our implementation, as well as the results produced by our different approaches. The Runtime Comparison Section 5.1 will cover the performance of our implementation, comparing the runtimes at different resolutions, as well as between different approaches. The Problems and Decisions Section 5.2 will discuss the problems encountered while working on this thesis and give reasons for our decisions on how to solve them. In our Results Section 5.3, we will offer a deep insight on how different approaches affect the results. We will first evaluate our direct matching methods for a few representative objects and pick the approach showing the best results. Following that, we will evaluate how the results the algorithm we picked produces at different settings compare to two other methods we implemented: Patch Match and Region Growing. Next, we will show a few examples of objects relighted under novel light conditions. Finally, the Setup and Processing Times Section 5.4 will evaluate the time the whole process from start to finish takes.

5.1. Runtime Comparison

To evaluate the performance of our implementation, we have run a series of benchmarks on our test system with an Intel Core i7-2600K CPU @2.67GHz. All benchmarks were single-threaded.

The first benchmark (Table 5.1) illustrates the importance of segmentation, as well as the speed advantage the FLANN library offers compared to linear implementation. The *Pixels* column shows how many pixels have to be processed by the search algorithm. The reduction of calculation times gained by segmenting a set of seven images before building the data set are massive. Even taking the fact segmentation itself takes a certain amount of time into account, enabling segmentation is worth it: The calculation times listed on the right of Table 5.1 include the time needed to segment the image.

Looking at the speed gains provided by FLANN show a similar picture, with the calculation process taking an acceptable amount of time even without segmentation: under a minute at 1280×800 pixels. With our resolution being limited by the camera's video output resolution of 1024×680 pixels, we have a calculation time of 21.514 seconds without segmentation. Enabling the segmentation delivers a large speedup, netting a total runtime of only 0.258 seconds (segmentation: 0.089 seconds; mapping: 0.169 seconds).

	Segmentation Off			Segmentation On		
Resolution	**Pixels**	**Linear**	**FLANN**	**Pixels**	**Linear**	**FLANN**
640×426	0 272 640	00 h 32 m 36.940 s	00 h 00 m 06.671 s	08 532	00 h 00 m 04.964 s	00 h 00 m 00.091 s
800×533	0 426 400	01 h 20 m 57.010 s	00 h 00 m 08.948 s	13 242	00 h 00 m 12.042 s	00 h 00 m 00.147 s
1024×680	0 696 320	03 h 34 m 51.200 s	00 h 00 m 21.514 s	21 161	00 h 00 m 32.285 s	00 h 00 m 00.258 s
1280×853	1 091 840	08 h 46 m 12.200 s	00 h 00 m 44.719 s	32 375	00 h 01 m 18.635 s	00 h 00 m 00.439 s

Table 5.1.: Linear search vs. FLANN over *intensities*: Time to calculate the correspondences between one and the same object, at different resolutions and patch size of 1×1, with segmentation disabled (left) and with segmentation enabled (right).

	FLANN		Patch Match		Region Growing	
Patch Size	**Over Intensities**	**Over Normals**	**Over Intensities**	**Over Normals**	**Over Intensities**	**Over Normals**
1×1	0.657 s	0.677 s	0.569 s	0.495 s	16.970 s	16.053 s
3×3	1.041 s	1.037 s	0.650 s	0.617 s	17.092 s	16.137 s
5×5	1.667 s	1.684 s	0.833 s	0.767 s	17.124 s	16.349 s
7×7	2.401 s	2.392 s	1.035 s	0.982 s	17.183 s	16.083 s

Table 5.2.: Patch Sizes: *Total runtime* for each algorithm with segmentation enabled, depending on patch size. Linear not measured as it only supports a patch size of 1×1. Resolution: 1024×680. Region Growing uses a threshold of 0.02.

The second benchmark (see Table 5.2) shows the performance of each of our correspondence calculation methods at our target resolution of 1024×680, as well as the impact of increasing the patch size (as described in Section 4.2.2). Since the patch size is also used for segmenting both the source and the destination image set, basing the decision on whether to add a point to the list of valid positions on the patch of pixels centered at this point, we decided to show the *total runtime* of the program instead. The discrepancy between the intensity and the normals approaches for each method comes from the fact the intensity approach uses a *seven-component* vector, while the normals approach uses only a *three-component* vector. This means that calculating the distance takes slightly longer for the intensity approach.

The two FLANN based methods (over intensity and over normals) use the patch size for both the segmentation and the correspondence map calculation steps. Our Patch Match based approaches do not use the FLANN search structures at all and are more efficient, because the underlying algorithm uses the structure of the image to it's advantage, thus managing to be consistently faster than our FLANN based methods. Most of the impact in case of Patch Match comes from segmentation taking slightly longer due to a bigger patch size. The Region Growing methods use the FLANN search structures for the first stage of the algorithm (*seeding*) only. The second stage, the actual Region Growing, has such a large impact on the speed that increasing the patch size does not show any significant differences.

Patch Size	Candidates: 1	Candidates: 25	Candidates: 50	Candidates: 100
1×1	0.677 s	2.952 s	4.847 s	8.302 s
3×3	1.037 s	5.634 s	7.691 s	11.301 s
5×5	1.684 s	7.451 s	9.892 s	14.616 s
7×7	2.392 s	9.133 s	12.372 s	18.719 s

Table 5.3.: Impact of enabling the additional improved candidate picking step (as per Section 4.2.2) for FLANN over *normals* at different patch sizes. Resolution: 1024×680.

Our last comparison (Table 5.3) shows how enabling our improved matching method of keeping a *list* of candidates and picking the locally closest one, as described in Section 4.2.2, affects the time needed to calculate the correspondences. For this benchmark, we picked *FLANN over normals* as our representative method and looked at the impact at different patch sizes. The *Candidates* column refers to how many candidates, sorted by ascending distance between the normals of the query and the candidate point, are stored in the list. One candidate means we do not enable the additional step.

The impact is relatively large due to the fact our current optimization function uses a linear approach. This penalty could be lessened by using a FLANN search structure instead, but for short candidate lists the gains are negligible. This is because initializing a FLANN search structure per pixel introduces a performance hit in itself. Increasing the candidate list length is not viable beyond a certain length, as we are going to detail in Section 5.2, so switching from linear to FLANN approach for this step is debatable.

5.2. Problems and Decisions

In this section we will discuss the problems encountered while working on this thesis and explain our reasoning behind the decisions on how to solve them.

5.2.1. The Data Capturing Stage

One of the main problems at the capturing stage lies in the controlled lighting required by the Photometric Stereo approach. To reduce any unwanted light influencing the results, objects have to be captured in a low-light environment, ideally a windowless room. One has to also be wary of unwanted reflections on the objects which might come from unexpected sources.

For example, our early image sets were taken in a completely darkened room, with only a computer monitor at the opposite end of the room. The light from the monitor did not seem to be bright enough to reach the object at first glance, but upon closer inspection, a reflection

caused by the monitor could be seen even on brightest of objects. These reflections in turn caused artifacts on the normal map of the captured object.

Another problem at the capturing stage is posed by the fact the object must be in exactly the same position in all seven photos of a set. Otherwise, the images will not align properly and the calculation of the normals and intensities will not be exact. This also means that the tripod has to be standing perfectly still, as well. Due to the weight of the setup, the tripod is more susceptible to vibrations. This means that depending on the flooring, walking around the setup might suffice to misalign the images.

People walking around the setup while taking the pictures could also lead to light conditions changing due to light from the setup reflecting from the clothing, or the person blocking a wall which reflected more light in a prior photo within a set. The best approach is to position the setup in a way the user can start the capturing process and remain sitting until the process is finished.

5.2.2. The Correspondence Calculation Stage

The approach utilizing the normals require segmentation to prevent our algorithms from mistakenly mapping parts of the background to the object, resulting in ugly black artifacts. To prevent this from happening, we load the corresponding image set along with the normals and perform the segmentation on the *intensities*, setting a normal which pixel failed the segmentation check to a predefined value. This value has to be chosen in a way that the distance between the invalid and a valid normal is always higher than between two valid normals to avoid mistakenly matching to an invalid normal.

In case of Patch Match, we also had to additionally prevent the random initialization from assigning an empty pixel to a value, or vice versa. We achieved that by instructing the algorithm to repeat the assignment until a non-empty pixel is assigned.

5.2.3. The Relighting Stage

The newest version of gphoto2 at the time of writing (2.5.4) introduced a regression which marked our Canon EOS 5D Mark II camera as incapable of video capture. We were able to downgrade to an earlier version (2.4.14) without any side effects.

The most obvious problem however is the fact that just as it was the case at the capturing stage, the source object must not be moved, otherwise the correspondence map will end up pointing to wrong pixels. This means we have to first capture our destination object, then the source object we wish to use, and do not move the source object after that.

5.3. Results

Since we cannot easily obtain a ground truth result, we had to find some sort of a baseline to compare our methods against. For this purpose, we have implemented a baseline algorithm which resizes the segmented source object to the dimensions of the destination object and creates an artificial correspondence map which merely copies the texture of the source object for every pixel that is not empty in both the source and the destination objects. This works well for objects of similar forms, but fails for an object pair with different geometries.

In our evaluation, we will also show the correspondence map to better illustrate the position each pixel is mapped to. Figure 5.1 shows a sample correspondence map with each pixel mapped to itself. This means that on our actual results, the higher the red value of the pixel is, the more to the right of the source image it was taken from. The green value on the other hand shows how far from the top of the source image the value was taken from. A combination of red and green gives yellow as a result, marking the lower right corner.

We will also show a few images describing where some pixel's origins are. For this purpose we created a tool which requires the correspondence map, it's rendition in RGB, the source and the destination object images, as well as the mapped result as input. The user can click anywhere on either the correspondence map, the destination object or the mapped result and the tool will draw a line between that point and the matched point in the correspondence map.

To be able to better evaluate our results, we have picked an appropriate light direction for each of our examples. All light positions used are generated from the captured image sets of the corresponding object. The light direction is stated in the image description and within the text. For example, "*light: coming from the right*" means we took the intensities from the second photo of the source data set and used the correspondence map to transfer them to the destination image. Appendix A provides more light positions for selected object pairs. The examples shown in Section 5.3.3 however, come from screen captures of a video recorded while moving the light around the object to positions different from the seven light positions our hardware setup provides.

Figure 5.1.: A sample correspondence map showing the mapping of each pixel to itself. The R component contains the x coordinate, G the y coordinate. B is always set to the value of 1.0 for consistency reasons.

5.3.1. Evaluation of Direct Distance Measures

Our direct distance measures are represented by our two FLANN based methods: FLANN over intensities and FLANN over normals. In this section, we will compare the two methods using several object pairs and decide which one delivers better results.

Figure 5.2 shows the results of mapping the texture of an apple to a banana. For this example, we have picked the light coming from the right. The baseline method does not preserve the geometry of the object in the results, as can be seen in the top right corner. It does however succeed in transferring more of the finer details of the texture, as visible in the yellow zoomed area.

For the other methods, there is a rather obvious flaw in the mapping, namely the black area in the middle of the banana (green and blue zoomed areas). Looking at the correspondence map in the same area, it does not appear smooth, the higher red value signifying the pixel was matched from around the indentation of the apple. Figure 5.3 shows this assumption is correct. This flawed mapping comes from the fact intensities do not discriminate between colors and the intensity happened to be closest to the intensity on that area of the banana.

Another reason for this type of mismatch can be caused by segmentation. Choosing a higher cutoff value in this case could help to avoid including pixel too dark for a reasonable mapping into the matching list. However this is not the case here, as visible on the FLANN over normals results (blue zoomed area), the dark area is much smaller in this case. The mismatches in this case come from the fact there are two or more intensity or normal vectors which happen to have nearly the same distance to the query vector. This misleads the algorithms into picking the one with the wrong color, since neither intensities nor normals contain color information.

It is also visible that our methods working on intensities have lost the highlight on the right, while the texture on the left side of the banana is not smooth (green zoomed area). The normals approach manages to transfer the highlights and keep the surface smooth (blue zoomed area). The highlight also looks glossy as it is on the apple (source object), not the diffuse kind as visible on the banana (destination object).

Overall, the intensity approach seems to be more susceptible to mismatches should the intensity appear to be closer in an area of the source object with a completely different orientation. On the other hand, the normals approach manages to produce less mismatches, but suffers from a “grainy” appearance, the effect especially visible around the highlight of the result. However, depending on the texture, the “grain” might improve the subjective appearance, by emulating the fine details of a texture. The normals method also transfers more highlights than the intensity approach, leading to more promising results for this object pair.

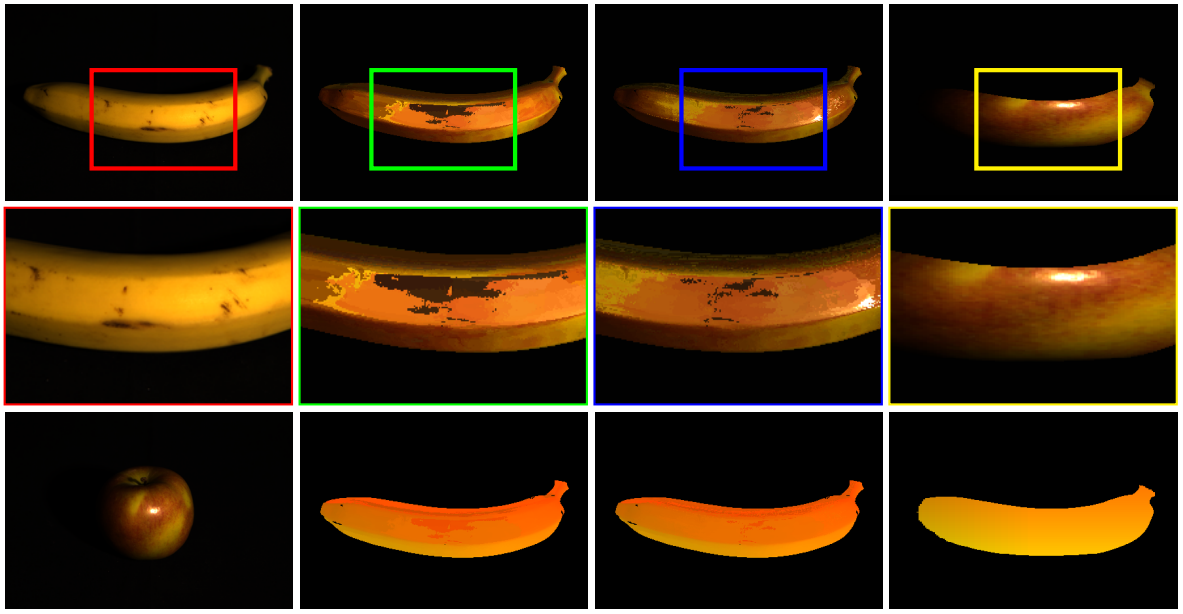

Figure 5.2.: Comparison of Direct Distance Measures I. Top: Destination object, FLANN over intensities, FLANN over normals, baseline methods. Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps for the methods above. Light: coming from the right.

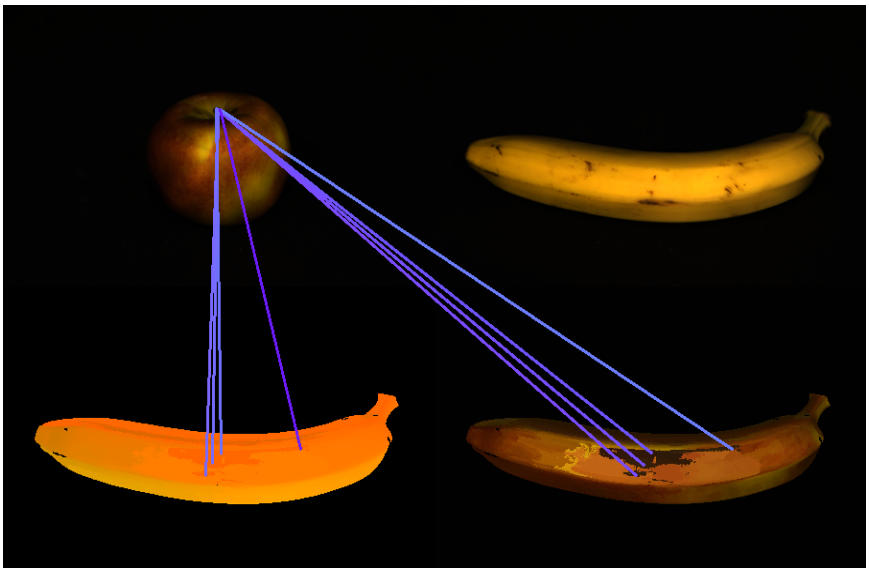

Figure 5.3.: The Origins of (Mis)matches I. Top row: source and destination objects. Bottom row: the correspondence map and the result of mapping the texture. Method: FLANN over intensities. Light: median of intensities.

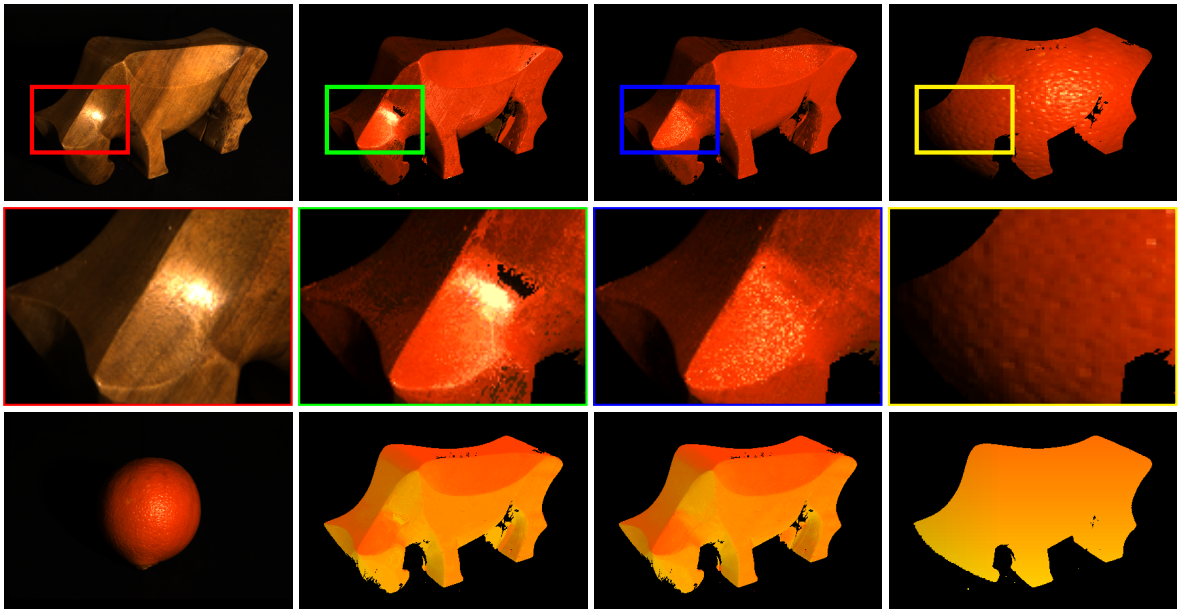

Figure 5.4.: Comparison of Direct Distance Measures II. Top: Destination object, FLANN over intensities, FLANN over normals, baseline methods. Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps for the methods above. Light: coming from the right.

Before we decide which algorithm to choose as our representative for further evaluation, we would like to look at more examples. Our previous object pair had many different normals for the source object, with the surface having specular highlights and a rather fine texture detail. The destination object had a diffuse, monochrome surface except for a few black spots. Both objects were overall convex, except for the indentation on the apple's top and a slight concavity around the banana's stem.

For our next example, we picked a pair of objects which have a restricted variation of normals for the destination object and kept a spherical object as source. Figure 5.4 shows the result of mapping the texture of an orange to a wooden bull statue. The light is coming from the right, demonstrating the behavior of the highlights on the statue's head. The concave form of the destination object poses a challenge for our methods. There are also some black holes in the statue due to the wood being rather dark, so our segmentation step discards parts of the object to not let any background pixels make it into the image.

Our FLANN over intensity approach succeeds at mapping the diffuse highlight of the orange to the head of the statue, but overemphasizes it, making it appear overexposed (green zoomed area). The resulting image does not appear to belong to a statue with an orange skin texture as the texture is too glossy.

There is a rather peculiar mismatch in the form of a dark spot in the same area. Looking at the correspondence map, that patch was matched from a completely different position on the map (from the bottom right of the orange), as the yellow value is higher. This means our approach once again falls victim to the problem of two intensity vectors on the source object having nearly the same distance to the position on the destination object and the wrong one being picked.

Looking at the results produced by FLANN over normals (blue zoomed area), shows an image much closer to the desired result. The highlights are present at the correct locations, but are diffuse, just like they are on the skin of an orange. This again shows the higher robustness of the normals approach for objects with a fine texture. The shadows appear natural around the bull's feet.

The result produced by our baseline approach does not have much resemblance with the bull statue anymore, the geometries of the source and destination objects are too different for the approach to deliver reasonable results. The highlight is on a completely different position, with the area around the hind feet being bright instead of staying shadowed as with our FLANN base approaches, appearing unrealistic. The edges of the wooden statue are also not visible anymore.

Finally, we tested our two direct methods with an object with a much coarser texture: a cup with several colored dots. The result can be seen in Figure 5.5, showing a mapping of a dotted cup to a white one. The light is again coming from the right to be able to better illustrate the artifacts near the handle of the cup.

With the geometries of the source and destination objects being so similar, the baseline method (yellow zoomed area) manages to deliver very good results. The highlights are in the correct position, the colored dots are perfectly visible and the overall appearance of the cup looks convincing, except for an unclean cut around the cup's handle due to its form and position differing between the two cups.

Both FLANN over intensities and FLANN over normals methods seem to fail to produce smooth results, with a lot of dark patches being matched to a light area (green and blue zoomed areas respectively). As before, multiple match candidates with nearly the same distance to the query point lead to our algorithms picking the wrong match. Figures 5.6 and 5.7 illustrate where the mismatches come from. Smoothness of the results can be improved by increasing the patch size and taking the position of the match in the account, as we described in Section 4.2.2. We will demonstrate the effects of doing so in the next section of this chapter.

Looking at the rest of the matching, both methods seem to succeed in roughly pinpointing the location of the colored dots. This is rather unsurprising in case of our FLANN over normals approach as it merely succeeds in approximately matching the correct normals. FLANN over intensities is not aware of the orientation of the surface however, so seeing less mismatches in its results is unexpected.

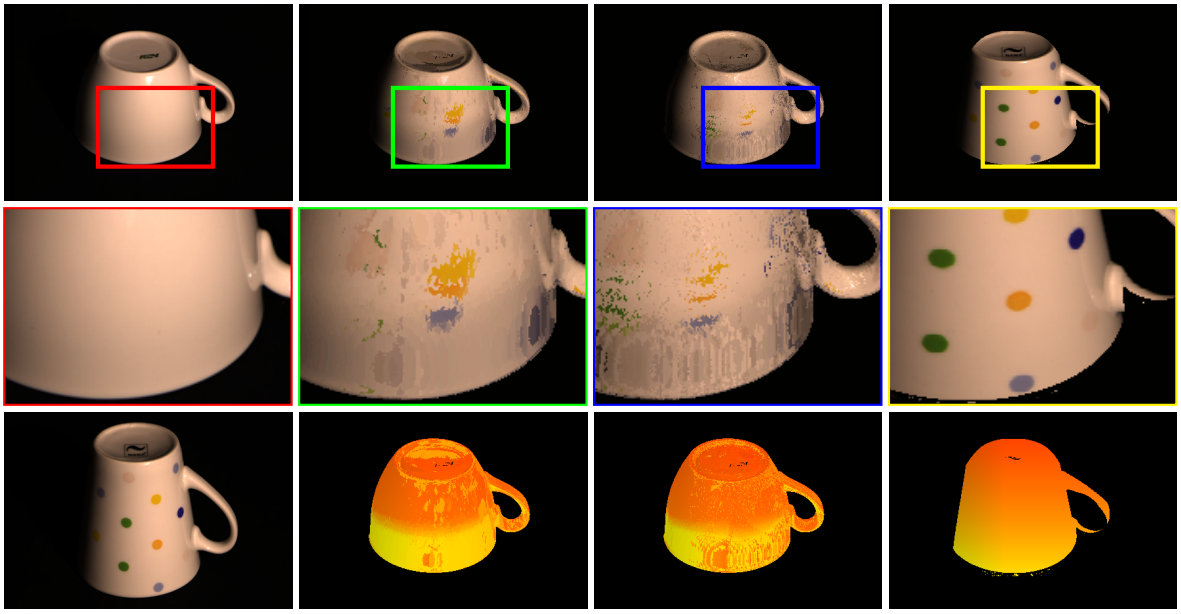

Figure 5.5.: Comparison of Direct Distance Measures III. Top: Destination object, FLANN over intensities, FLANN over normals, baseline methods. Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps for the methods above. Light: coming from the right.

However, even though there are less visible mismatches, the ones that *are* visible, such as the black spot near the handle in the green zoomed area, are rather bad. Looking at the correspondence map, this spot was matched to a pixel near to the edge of the cup, as shown in Figure 5.6.

The highlights are kept at the same positions as they should appear on the destination object, with the FLANN over intensity approach keeping them closer to the original. This is because the surface properties of the two cups are similar, so finding the correct match by intensity is easier than picking the correct normal out of several similar ones in this case. However, the highlights are still at the correct position for the FLANN over normals method as well, and the results can be further improved by making the method more robust. Even though the results produced by the method based on normals are inferior to the intensity approach for the last object pair, considering the normals method was able to show better results in the two other cases, we think it is the superior method. For this reason, we are going to present the improvements we implemented to make the methods more robust and show how they affect the results by using FLANN over normals in the following section.

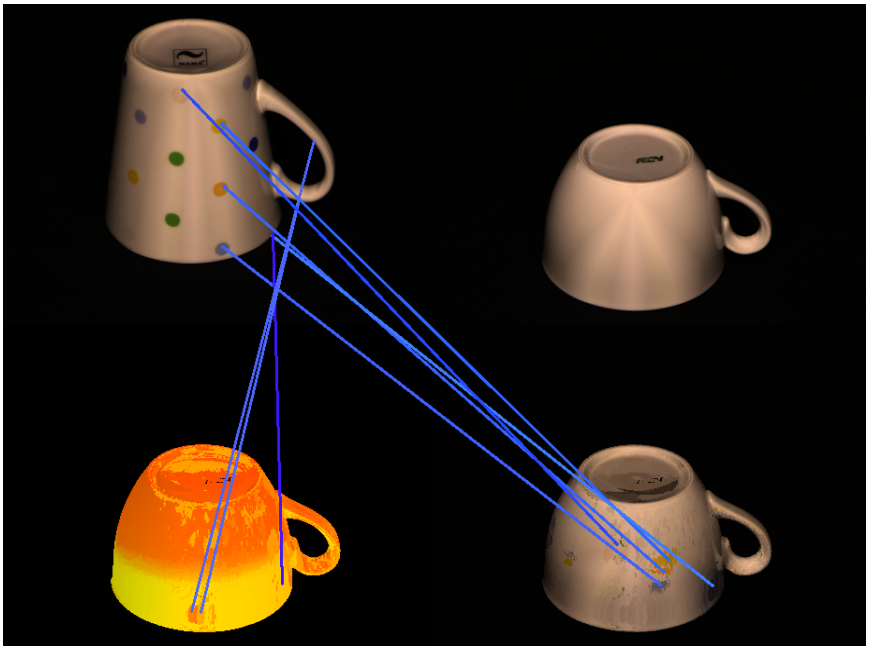

Figure 5.6.: The Origins of (Mis)matches II: FLANN over *intensities*. Top row: source and destination objects. Bottom row: the correspondence map and the result of mapping the texture. Light: median of intensities.

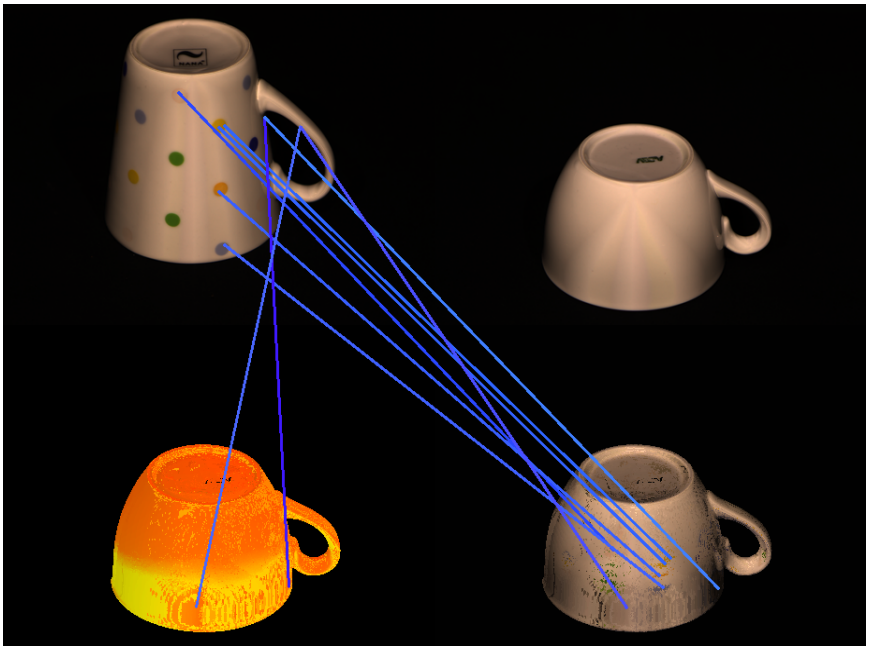

Figure 5.7.: The Origins of (Mis)matches III: FLANN over *normals*. Top row: source and destination objects. Bottom row: the correspondence map and the result of mapping the texture. Light: median of intensities.

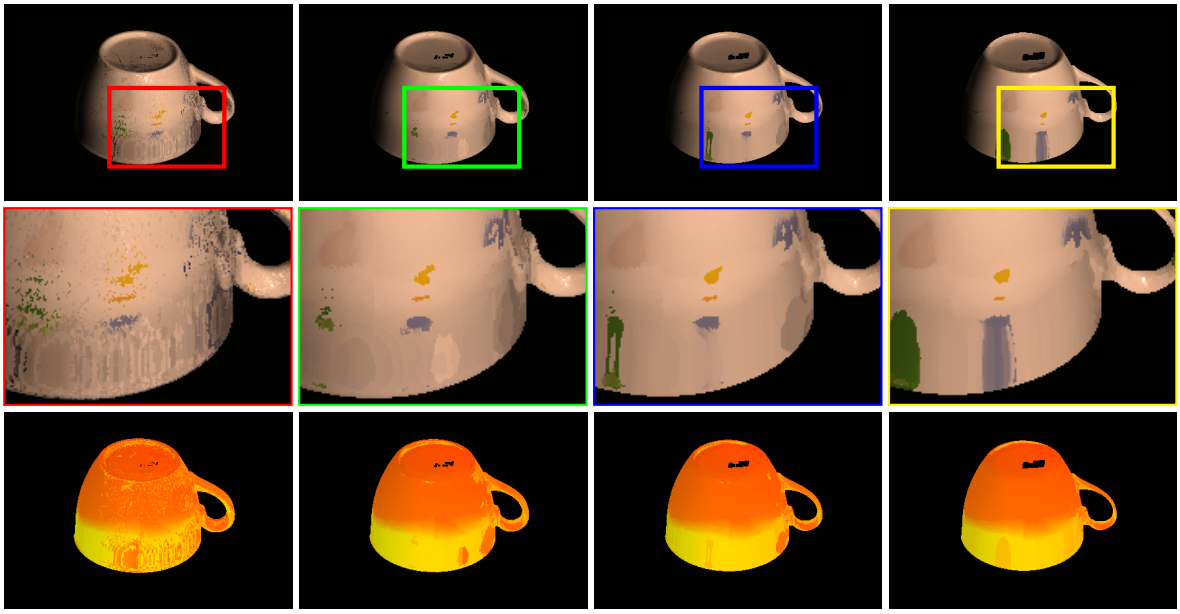

Figure 5.8.: Comparison showing the effect of increasing the patch sizes using the FLANN over normals method. From left to right: Patch size 1×1, 3×3, 5×5, and 7×7. For source and destination see Figure 5.5. Light: coming from the right.

5.3.2. Evaluation of Improvements

Now that we have decided which approach we wish to improve, we can look at any gains enabling the improvements described in Section 4.2.2 provides. There are three types of improvements we described: improving direct matching itself by varying the patch size and taking the position of matches in the image into account, changing to a different approach by using the Patch Match algorithm and finally taking a hybrid approach with Region Growing.

As we have shown in Table 5.2, increasing the patch size has an impact on the runtime of about two seconds for our FLANN based approach. To look at which advantages we might gain from doing so, we have run the FLANN over normals algorithm with different patch sizes on our third object pair, the dotted and the white cup. Figure 5.8 shows the result. The light remains unchanged from Figure 5.5, coming from the right.

Increasing the patch size leads to less mismatches. Nearly all of the black spots that marred the results in Figure 5.5 are gone after increasing the patch size to 3×3 pixels (green zoomed area). Further increasing the patch size leads to the results growing progressively smooth (blue and yellow zoomed areas). Due to the differing geometry of the objects, the bottom of the texture is stretched, while the middle is squished.

However, there were only small improvements where the actual texture, the colored dots, is considered. The position is correct for those points, with them still being present after smoothing out the result by increasing the patch size, but the texture is still very distorted.

To further improve the texture features' positioning, we have also implemented an approach which prefers matches locally closer to the query (candidate optimization). Figure 5.9 shows the result of matching two spheres of a different color to a Lego block. We have picked two spheres of different colors as the source object in order to have at least two different normals with the same orientation. This way the effect our candidate optimization step has on the result for this object pair can be visualized in a better way.

The destination object is constructed from bricks of four different colors: yellow, green and two shades of blue. The Lego block has a restricted number of normal orientations, so in order to be able to clearly see both sides of the block, we decided to show the scene with light generated from the median of intensities. Matching the two spheres directly using FLANN over normals leads to poor results (green zoomed area). The two colors are seemingly randomly scattered around the Lego block.

Turning on the candidate picking optimization immediately changes the color distribution (third column). While not perfect at the top of the cube, the color distribution is correct on the cube's sides (blue zoomed area) with as few as *three* candidates in the list. As visible on the correspondence map, the black line on the right side of the cube comes from segmentation, as the green brick is very dark.

Sometimes the candidate optimization step does not work well. Each object pair requires carefully tuning the number of candidates in the list. Choosing a too low count leads to little to no effect compared to matching directly, while picking a too high candidate count leads to bad matches being picked, breaking the texture completely. This effect can be seen in the last column of Figure 5.9, with the list containing *seven* candidates. Looking at the middle of the zoomed area (yellow), there are some dark pixels becoming visible where previously were none. These pixels do not appear due to segmentation but are actual mismatches, as can be seen on the correspondence map.

An example where our candidate picking optimization does not work as expected is shown in Figure 5.10. For this scene we have picked two Lego blocks constructed from bricks of four different colors: yellow, green and two shades of blue. Both objects have a distinct texture and the same geometry, with only two distinct normal orientations on the sides, one orientation per side (not counting the edges of the individual bricks and the top side).

This makes it very hard for our matching algorithm to work without candidate picking optimization enabled, as the algorithm ends up always matching the first normal with the correct orientation, coloring most of area yellow. The second column of Figure 5.10 illustrates this case, with the left side only having a few dark spots and the right side being randomly assigned (lime zoomed area).

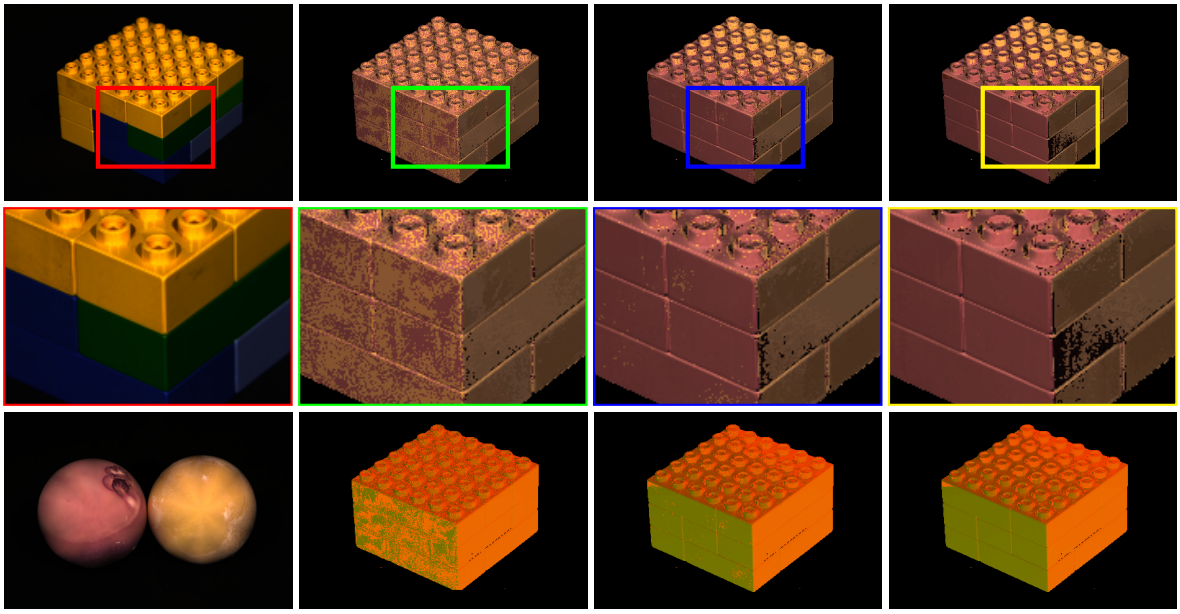

Figure 5.9.: Effect of candidate picking optimization: Picking the locally closer match, even if it is not optimal distance wise, improves the result. Top: Destination object, FLANN over normals with one, three and seven candidates to pick from. Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps. Light: median of intensities.

Turning on optimization does not improve the situation. On the contrary, the distribution of the colors has become even more random, with many green spots making it to the result on the left side of the block and the previously correctly mapped dark blue part of the brick on the corner is now incorrectly matched with yellow (cyan zoomed area) and light blue (orange zoomed area), which was supposed to be mapped further to the right. It is interesting to see that the dark green brick on the right side of the block starts to roughly gain a green color with a higher candidate count (orange zoomed area), even if it is not respecting the brick boundaries.

The algorithm does not work correctly due to three reasons. One reason is that because the Lego block has a limited number of normals per side, the list of candidates is more likely to be filled up with candidates of the same color, as we see happening on the left side of the cube.

The other reason is that our algorithm matches the image by iterating through it column- or line-wise. This explains why the brick boundaries are not respected and a hard horizontal line separating the colors is visible instead.

And the last reason is that increasing the candidate list to include normals with other colors results in normals with a too big distance being included into the list as well, harming the result quality. This is again visible at the corner block, where dark pixels suddenly start to appear.

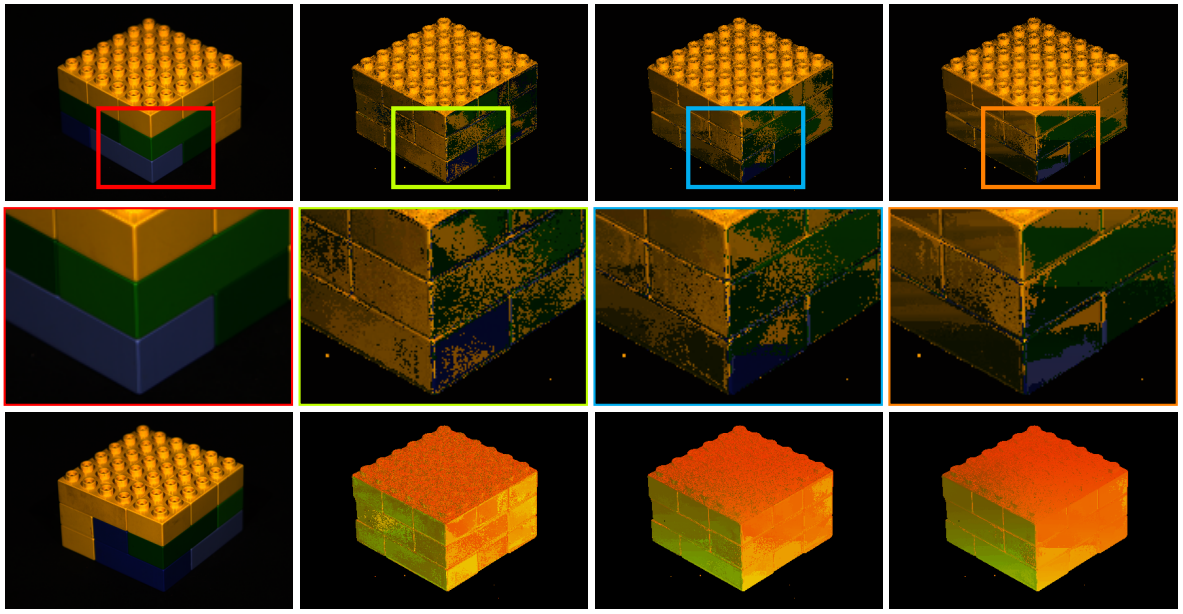

Figure 5.10.: Candidate picking optimization failing to improve mapping. Top: Destination object, FLANN over normals with one, seven and 50 candidates to pick from. Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps. Light: median of intensities.

The methods and improvements presented so far manage to transfer the general appearance of the object, as long as the *source* object does not have a limited normals count, like in our Lego cube (Figure 5.10) example. The *destination* object's geometry does not seem to affect the result, with even some concave objects (our wooden statue, Figure 5.4) appearing sufficiently realistic.

However, looking at more coarse textures like our two cups example (Figure 5.5) shows that the improvements are not enough to achieve satisfactory results. To work around these limitations, we implemented the Patch Match and Region Growing algorithms.

To compare the results we have reused the scene from Figure 5.4, transferring the texture of an orange to a wooden statue of a bull and the light coming from the right. The results produced by Patch Match over normals are nearly the same as the results produced by our FLANN over normals approach, as visible in Figure 5.11. The quality of the results produced by our Patch Match implementation suffers from the same flaws as the FLANN approach, with the only benefit being the faster calculation time, as shown in Table 5.2.

Running Region Growing over normals for the same object pair does not seem to improve the result. The highlights look the same, but the texture is more uneven, due to the bull statue being concave. This means that the regions cannot be grown too big without deviating too much from the destination object's surface.

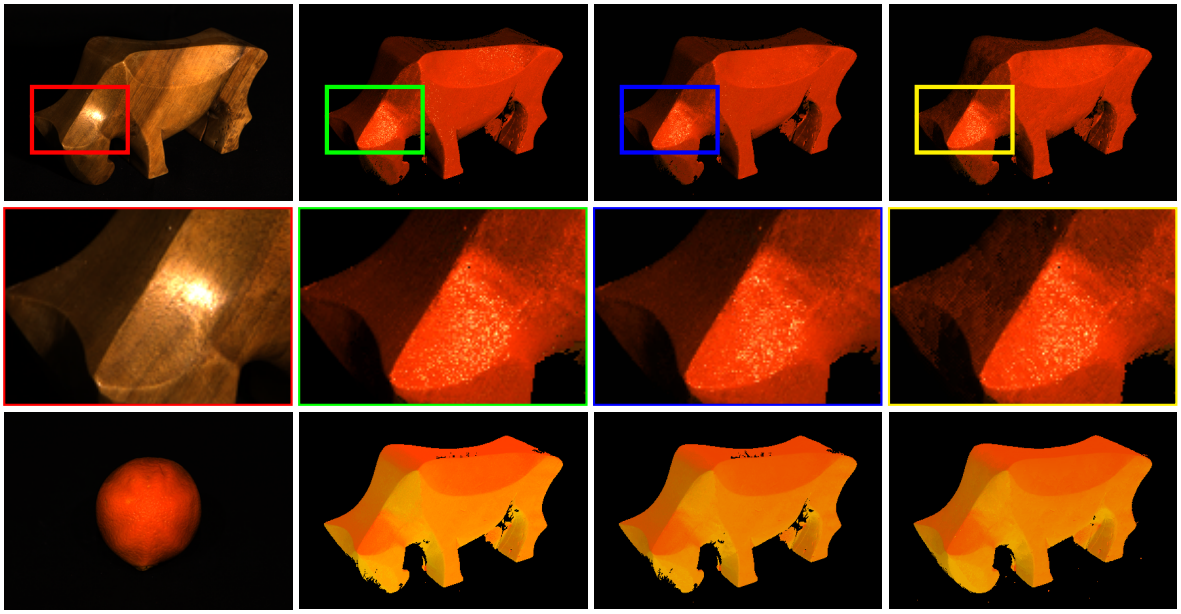

Figure 5.11.: Top: Destination object, FLANN over normals, Patch Match and Region Growing (growing threshold of 0.06). Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps. Patch size: 1×1; Light: coming from the right.

However, when the source texture belongs to an object which is rough to begin with, Region Growing over normals is able to provide more appealing results compared to the FLANN and Patch Match approaches. Figure 5.12 shows how mapping of a stone to the same wooden bull statue works out. With both the FLANN over normals and Patch Match over normals approaches, the results manage to transfer the colors correctly, but the result does not appear to belong to a stone, with the surface being too smooth and clean.

Region Growing however, delivers a more rough result, making the surface appear more believable. Especially the region around the head (yellow zoom area) looks convincing, with a light stone vein being seemingly transferred from the stone's middle right corner.

One more example of Region Growing showing its strength is our "two cups" example, as shown in Figure 5.13. This time we used light generated from the median of intensities to be able to show the dots on both sides of the cup. While the results are not perfect (yellow zoom area), it certainly manages to transfer much more of the texture than the FLANN over normals and Patch Match approaches. There is still an issue with dots not quite converging, like the blue dots at the left and the right side of the yellow zoom area. The green dots do not make it to the final image either. There are also still some dark regions at the bottom edge of the cup, but comparing the result to the ones produced by FLANN over normals and Patch Match, Region Growing definitely is an improvement where texture is considered.

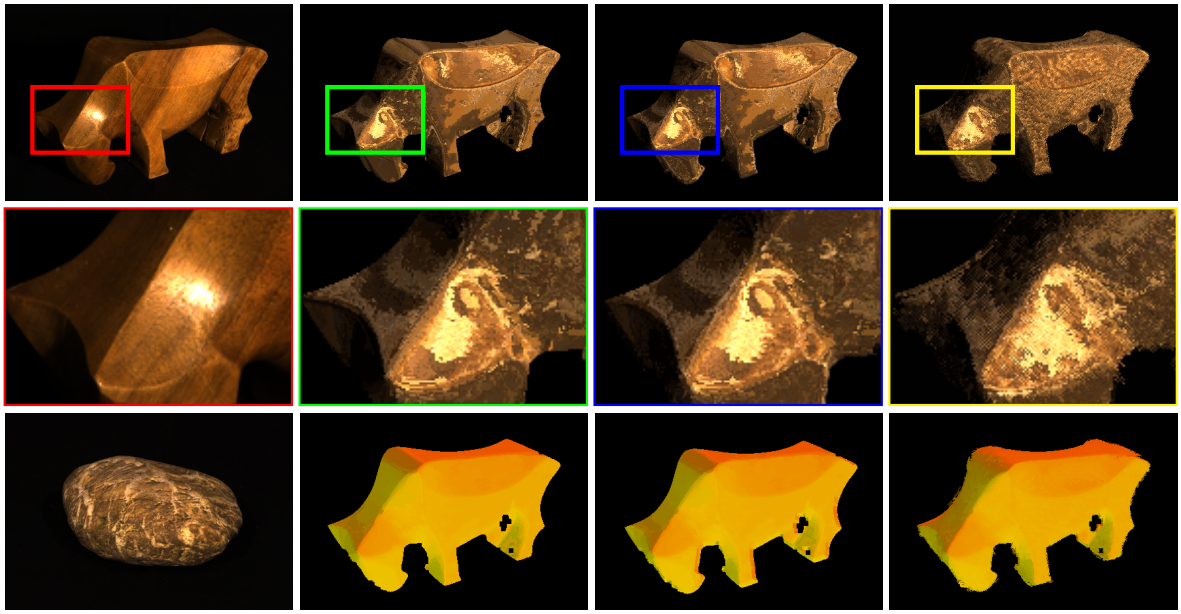

Figure 5.12.: Top: Destination object, FLANN over normals, Patch Match and Region Growing. Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps. Patch size: 7×7; Light: coming from the right.

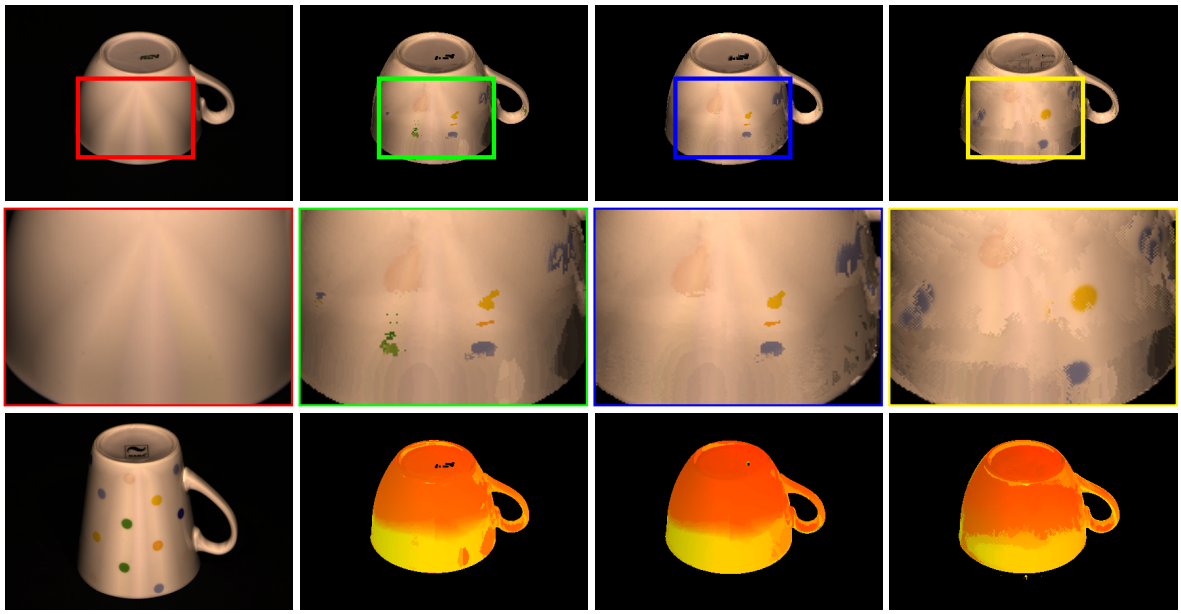

Figure 5.13.: Top: Destination object, FLANN over normals, Patch Match and Region Growing (with a threshold of 0.04). Middle: Respective zoomed regions. Bottom: Source object and the correspondence maps. Patch size: 3×3; Light: median of intensities.

In conclusion, FLANN or Patch Match over normals produce good results for smooth textures, both glossy or matte, as we have seen on our examples in Figures 5.2 and 5.4. Our Patch Match implementation is faster than our FLANN approach but lacks the capability to improve the match by picking the locally closer one. Running our Region Growing algorithm with a threshold of 0.0 is equivalent to our FLANN approach, as it then does not grow the region, searching for direct matches only. For this reason, we see it as the superior, albeit slower approach, allowing to handle both smooth and rough textures.

5.3.3. Examples of Objects Relighted Under Novel Light Conditions

Having looked at the quality of our methods' results, we would like to show how our tested objects with a new texture mapped to them behave under novel light conditions. We have extracted several frames from a video of each object, using light positions differing from the seven positions our hardware setup provides.

Figure 5.14 shows the texture of an orange mapped to a wooden statue of a bull. The light is traveling on an arc starting from the top of the object and moving to the left, with the highlight wandering along the neck of the wooden bull statue according with the movement of the highlight on the surface of the orange. The areas light up according to the light source's position, looking convincing with the surfaces shimmering not unlike the orange itself.

The surface overall is sufficiently smooth, without patches with mismatched brightness or orientation visible. There are some pixels at the rear of the bull which appear to be lit even though this should not be the case, but the effect is minor and does not harm the final appearance much. The rest of the lighting appears natural, with only minor discrepancies around the hind feet of the bull statue, where the light continues to shine even though in reality it should have been blocked by the body of the statue.

Figure 5.15 shows the result of mapping two spheres of a different color to the Lego block appear under a different lighting. To illustrate the fact the lighting still makes sense for the limited normal orientation count of the Lego block, we have chosen light positions behind the two spheres: at the left, top and to the right. The last column also shows the light being positioned between the spheres and the camera.

The lighting of the Lego block behaves reasonably well, with shadow artifacts at the top surface for the first three light positions. This is due to the object's concave form. The rest of the surfaces changes the lighting according to the light direction, with only a few pixels standing out (red zoomed area). Some of the pixels appear black due to segmentation (see correspondence maps on Figure 5.9).

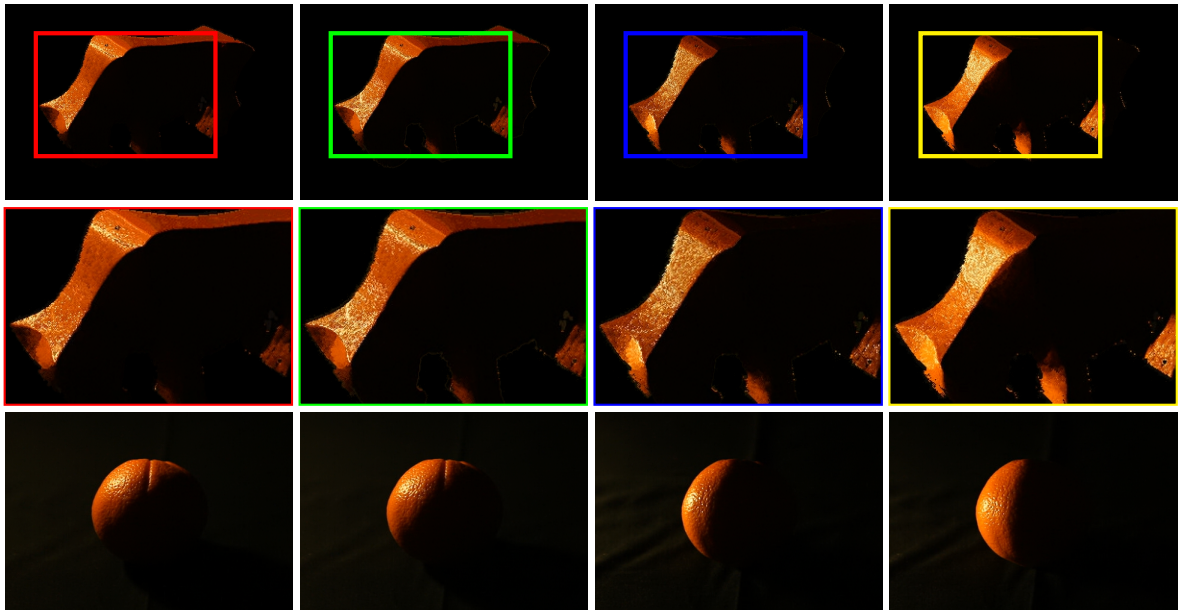

Figure 5.14.: Relighted Example I: Orange texture over a wooden bull statue. Source object at the top, source object at the bottom, light descending from above. Method: FLANN over normals and patch size of 1×1.

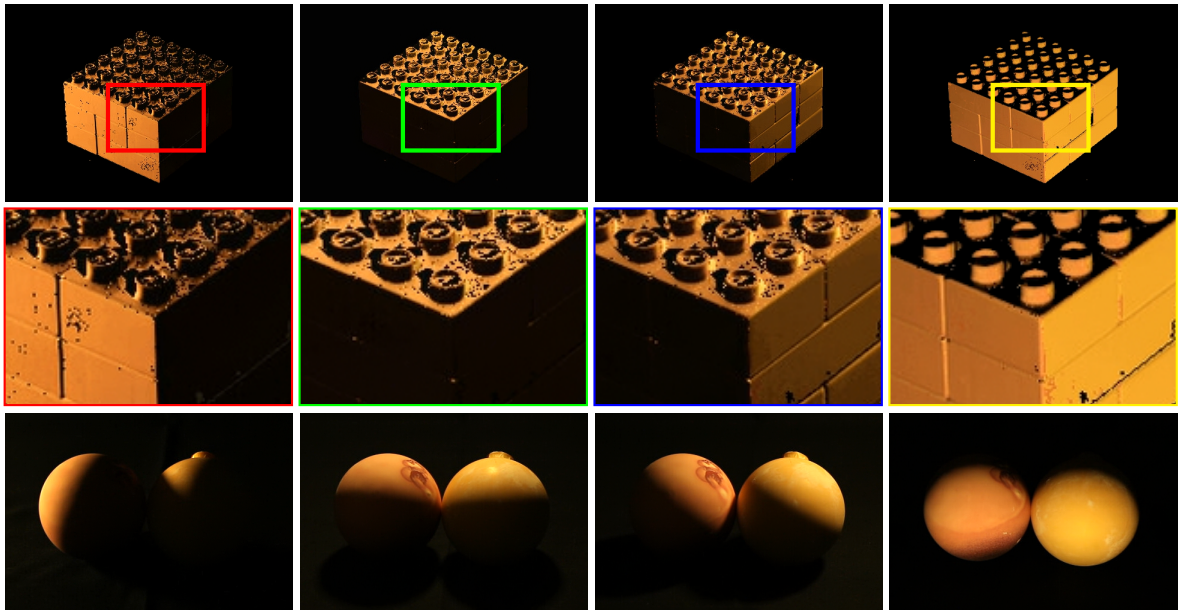

Figure 5.15.: Relighted Example II: Two colored spheres' texture on a Lego block. Mapped object at the top, source object at the bottom. Light moving from left to right behind the object, then to the front and below the object. Method: FLANN over normals with three candidates and patch size of 1×1.

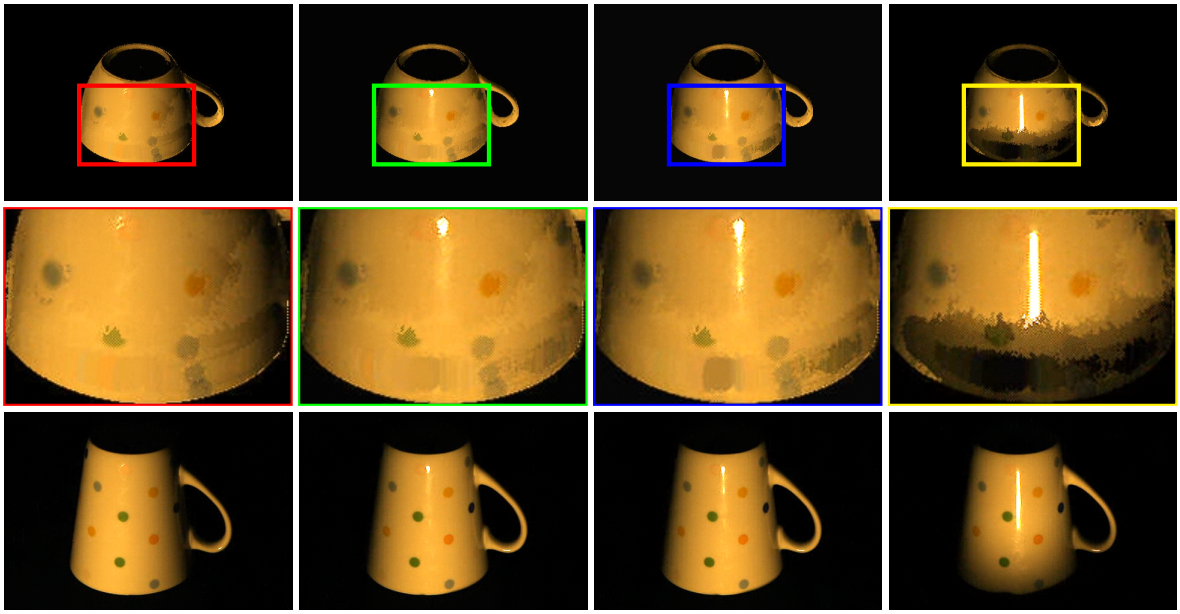

Figure 5.16.: Relighted Example III: A dotted texture on a white cup. Mapped object at the top, source object at the bottom. Light descending from left to right to the front and below the object. Method: Region Growing with 50 candidates, patch size of 3×3 and a threshold of 0.04.

Figure 5.16 shows our example with a dotted cup's texture being transferred to a white cup. The light travels from the left in front of the object on a downward arc, going below the table our source object is placed on. This leads to the light gradually being obstructed by the table, placing a shadow on the object.

We purposely chose this example to show that despite the result not being smooth over the whole surface, the mapping does make sense within the regions themselves. The highlight is traveling along the surface of the destination object, appearing at the correct positions.

When the light is obstructed by the table, a black patch suddenly appears on the destination object, even though the light did not sink deep enough to cause the same shadow on the source object. This is because this area of the destination object was matched to the area at the outermost edge of the rim of the cup. Due to the differences in the cups' geometry, these normals presented a better match for our methods.

There is an interesting effect visible on the cup, with the area near the bottom of the cup (to the top right outside the zoomed area) appearing lit even though the same area is not lit on the source object. This is correct, as the destination object is smaller than the source object, meaning that even though the light does not reach that spot on the dotted cup anymore, it still does reach that area for the mapped one, showing that our mapping also appears realistic when mapping object of different sizes.

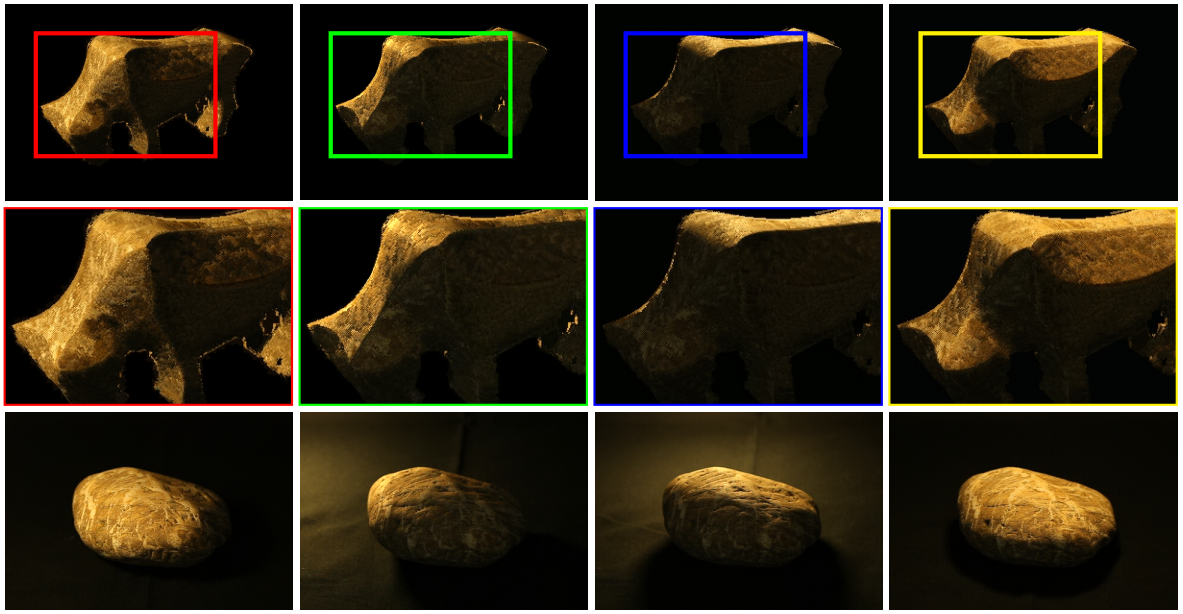

Figure 5.17.: Relighted Example IV: A stone texture on a wooden bull statue. Mapped object at the top, source object at the bottom. Light moving from left to right behind the object, then ascending to right above the object.

Our final example (see Figure 5.17) shows the texture of a stone transferred to the wooden bull statue. The light is moving from left to right behind the stone, finally ascending to a position above it. As with the example in Figure 5.14, the lighting appears convincingly realistic, with only minor artifacts visible around the hind feet and the rear of the statue.

The texture on the bull statue behaves similar to the texture on the stone. While it does not replicate the texture completely, the overall appearance is close to the original, with white veins lighting up on the surfaces when light hits them, remaining dark otherwise.

Overall, the lighting on newly relighted objects appears convincing enough to be called realistic. The texture on the destination object, while not transferred perfectly in all cases, gives the object an appearance which bears a high grade of similarity to the source object's texture. The resulting appearance also manages to inherit the properties of the source object using our approach. This means that transferring an appearance of a glossy object to an object with a matte surface will result in the destination object appearing glossy as well. The same is true for the opposite case.

5.4. Setup and Processing Times

To conclude this chapter, we want to look at the time the whole process starting from acquiring the two objects to starting the video capturing process takes. For this purpose, we have chosen our worst case: two dark objects with a rough texture, our stone to wooden bull statue transfer introduced in Figure 5.12. The time spent on the initial assembly of the setup requires about *one to ten minutes*, depending on how far the setup was disassembled for the transport.

Not taking the time for the initial assembly of the physical setup into account, the longest time is spent photographing an object: one minute and 28 seconds *per object*. This is due to combination of a higher aperture required to keep all parts of the bull statue sharp, the amount of light a single Philips Hue provides at maximal brightness and the dark surface of the wood requiring a long exposure time of *four* seconds.

Once the capturing process is complete, converting the image set from the camera resolution of 5634×3753 pixels to our target resolution of 1024×680 pixels takes about six seconds *per object*. Calculating the normals takes seven seconds *per object*. This sums up to a time of about *three and a half minutes* so far.

The rough texture of the stone requires us to use our slower Region Growing approach. The calculation of the correspondence map with a threshold of 0.08 takes about 33 seconds. The total time needed for our worst case is thus about *four minutes*.

We see this time as acceptable for a worst case scenario. However, this scenario does not account for the time spent testing which aperture and exposure each object requires. Using an object pair which does not require such a high exposure time is one way to reduce the total time needed to start relighting the objects. Another large speedup can be achieved by specifying a smaller threshold for the Region Growing algorithm. This could lead to degraded results depending on the source object's texture, however.

6. Summary and Future Work

In this thesis, we have presented a new method to transfer the appearance of one physical object to another, allowing the user to interactively view the destination object with the texture of the source object applied under novel light conditions. To achieve that, we designed a hardware setup consisting of a Canon EOS 5D Mark II camera and a chipboard holding seven Philips Hue lamps, mounted on a tripod. All parts of the setup are widely available to anyone, requiring no specialized or overly expensive parts.

Our workflow consists of capturing the destination and source objects (in that order), calculating the correspondence map based on the image data and finally displaying the source object's texture on the destination object. Each object requires seven photos to be taken with one of each lights active for each photo. The photos are used to obtain the normals of the object as well as the intensities. We used several approaches to calculate the correspondence map, directly matching two pixels of the source and destination objects by either normals or intensities. By matching two patches of pixels instead, as well as keeping a list of candidates and picking locally closest one, we presented several approaches to make the matching more robust. We also introduced two more approaches, with Patch Match transferring the texture on a patch-to-patch basis and Region Growing representing a hybrid approach of directly matching pixels and attempting to transfer neighboring pixels as long as it does not violate defined constraints.

The results were extensively evaluated and discussed in the second part of the thesis, offering insights into runtime and quality of algorithms presented, also showing the impact and gains brought by our measures to improve the results. We have also discussed the problems encountered while working on this thesis, detailing our reasoning for the decisions made to solve them.

In this final chapter of the thesis, we would like to offer several ideas for future work which can be done based on the work presented, concluding the thesis.

Future Work

One major flaw of our approach is the fact that neither the source object nor the setup can be moved after acquiring the data without the correspondences becoming invalid. A method to be able to re-locate the correspondences is required. This could be done by overlaying a half-transparent outline of the object over the camera picture, allowing the user to re-align the source object with its position on the correspondence map.

The quality of the transferred appearance is lacking in some cases. To improve the matching, we suggest using different distance measures to improve the results, for example using *Normalized Cross Correlation*. Improving our Patch Match implementation by enabling the method to also match rotating and scaling patches could improve the results. Trying out other matching criteria is also worth pursuing, for example switching from the RGB to the HSV[1] color space, and using the value channel to define the intensity. Another approach would be to match by taking color into account, discarding the match if already matched pixels neighboring the current one are not of a similar color.

One could also work towards expanding the types of objects the methods can handle, allowing to transfer the appearance for non-Lambertian objects, for example translucent objects. To be able to do that, segmentation has to be handled in a way to account for the background showing through the object. This could be achieved by improving the GUI[2], allowing the user to paint over the image of an object with a brush of a variable size to allow directly marking valid pixels. There is also still work to be done to improve handling of concave objects, as well as objects with limited normals. In case of objects with limited normals, the user could help the program out by pointing where to map the texture to. We imagine combining the work by Ukamatsu et al. [UMKT12][UMKT13] with our methods of transferring the appearance would allow for an intuitive workflow.

On the performance side, there are several ways to improve it even further. For example, utilizing FLANN's multi-threading support should bring noticeable gains. In case further improvements are desired, moving the whole code to a GPU[3]-based implementation should provide even higher gains. This is viable, since most of the operations are performed per (patch of) pixel(s). This means that the result of the segmentation check for this pixel only depends on its own *value* and the *values* of the neighbors in case of patches, but not on the *results* for other pixels. The same is true for the calculation stage, where the result for a pixel only depends on the set of valid points to match against. Following this route could lead to the calculation of the correspondence map being possible nearly in real time, since our current implementation slows down considerably with increasing patch sizes and number of candidates saved, as we have shown in Table 5.3.

[1] http://en.wikipedia.org/wiki/HSL_and_HSV
[2] Graphical User Interface
[3] Graphics Processing Unit

A. Further Results

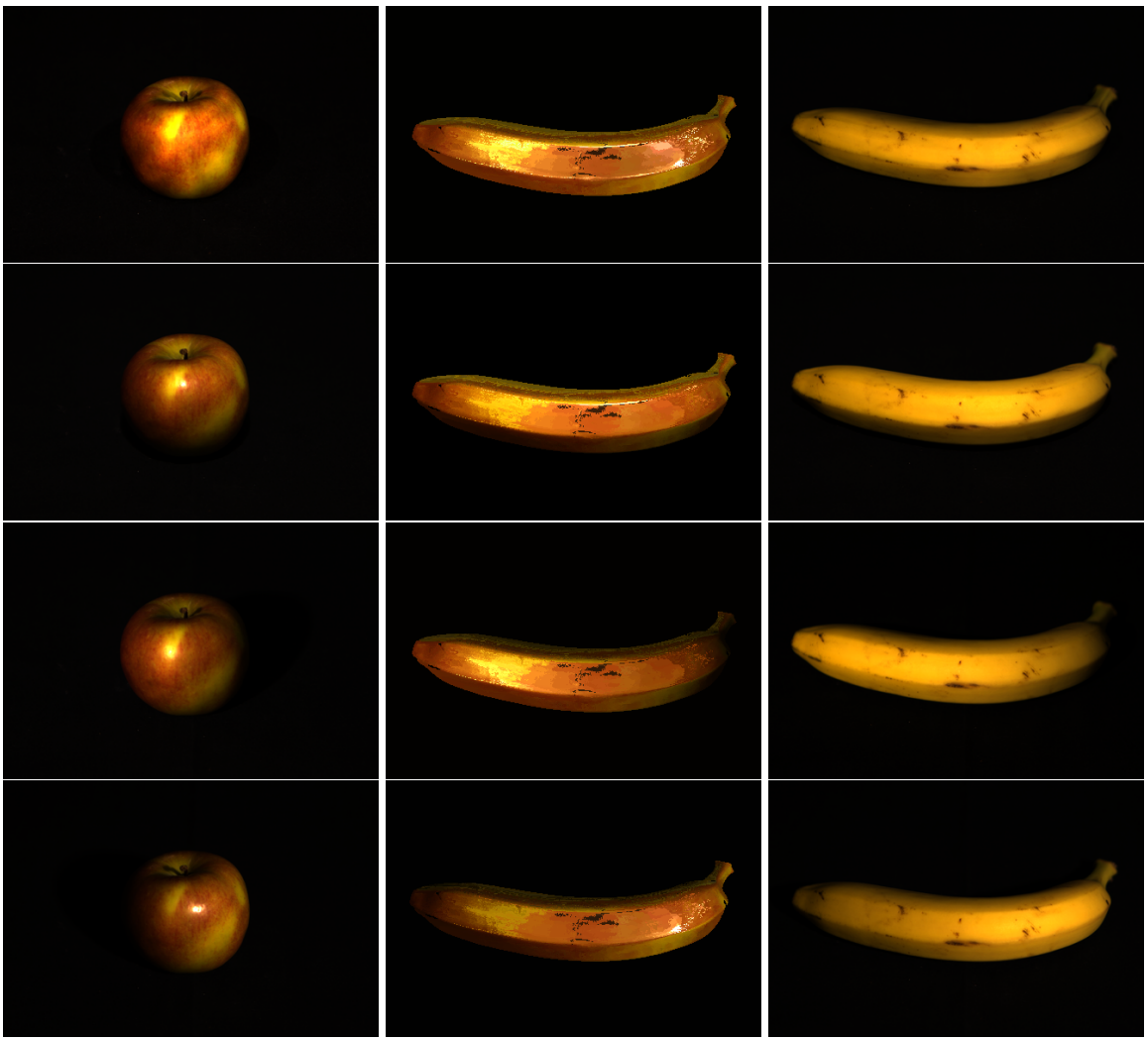

Figure A.1.: Results of mapping the source object (left) to the destination object (right). Middle image is the RGB mapped input. Method: FLANN over normals. Light (from top to bottom): Median of intensities, from the top, from the left and from the right.

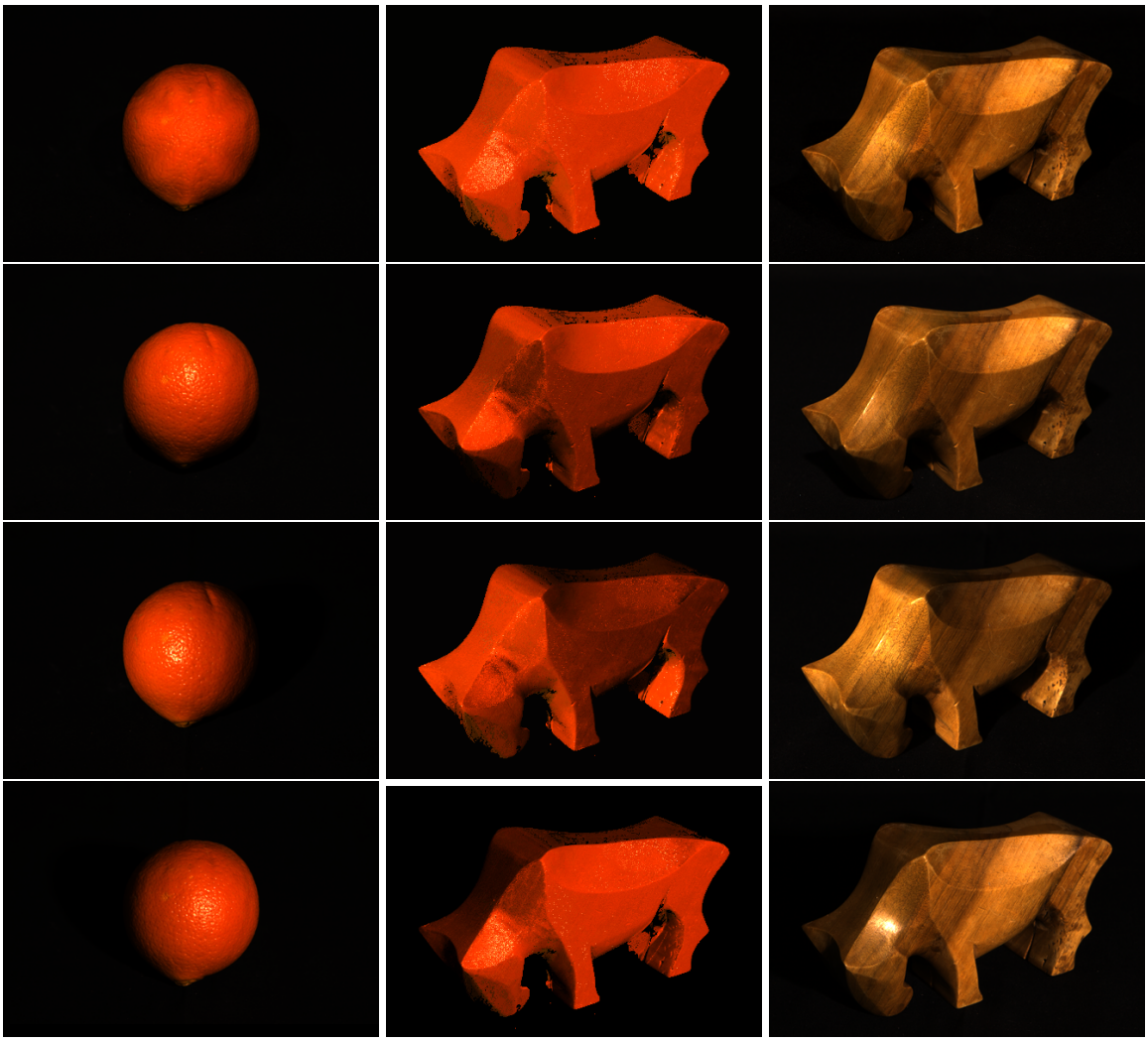

Figure A.2.: Left: source object. Right: destination object. Middle: the mapped result. Method: FLANN over normals. Light (from top to bottom): Median of intensities, from the top, from the left and from the right.

Figure A.3.: Left: source object. Right: destination object. Middle: the mapped result. Method: Region Growing over normals, patch size of 3×3. Light (from top to bottom): Median of intensities, from the top, from the left and from the right.

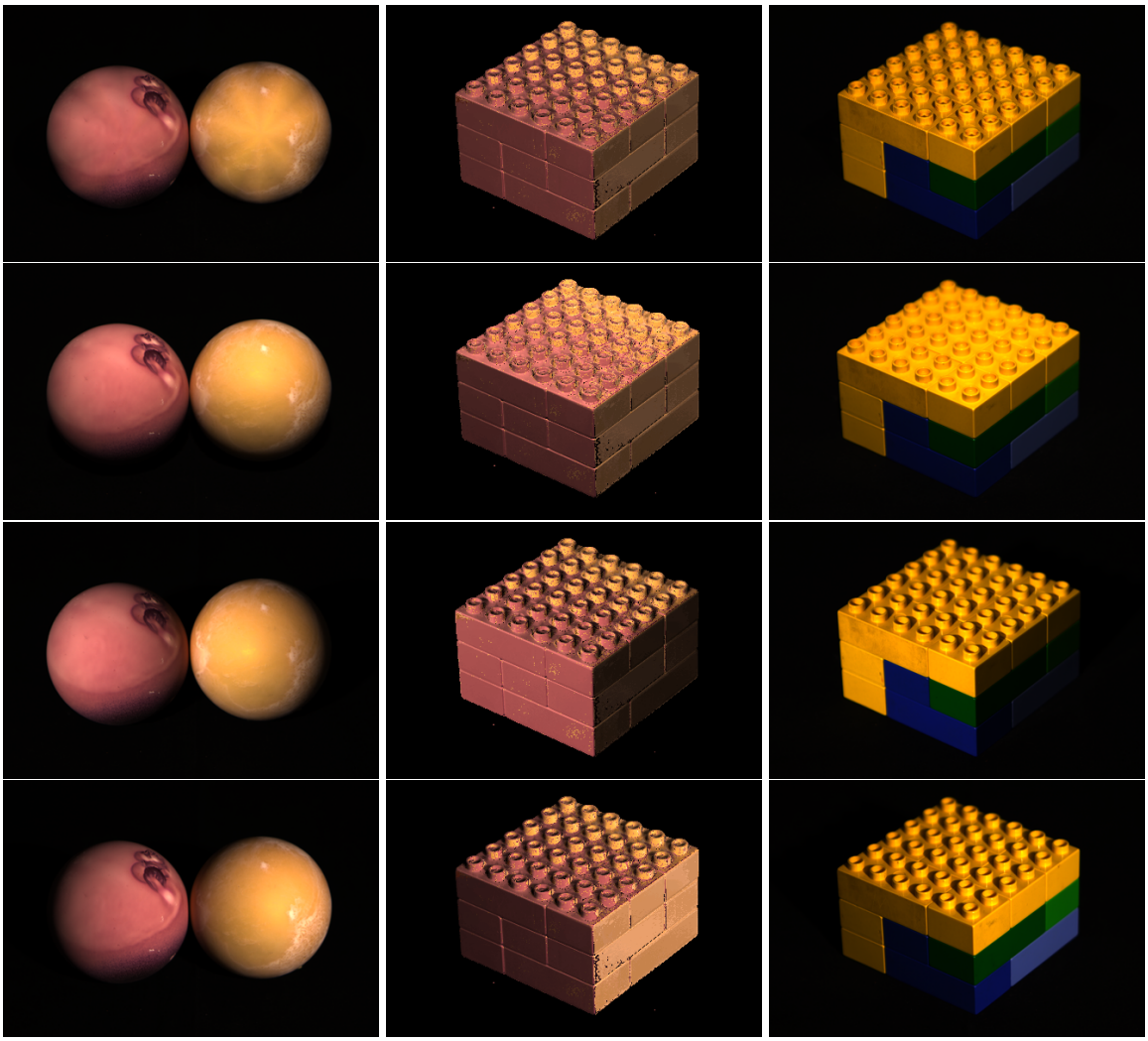

Figure A.4.: Left: source object. Right: destination object. Middle: the mapped result. Method: FLANN over normals, three candidates in the list. Light (from top to bottom): Median of intensities, from the top, from the left and from the right.

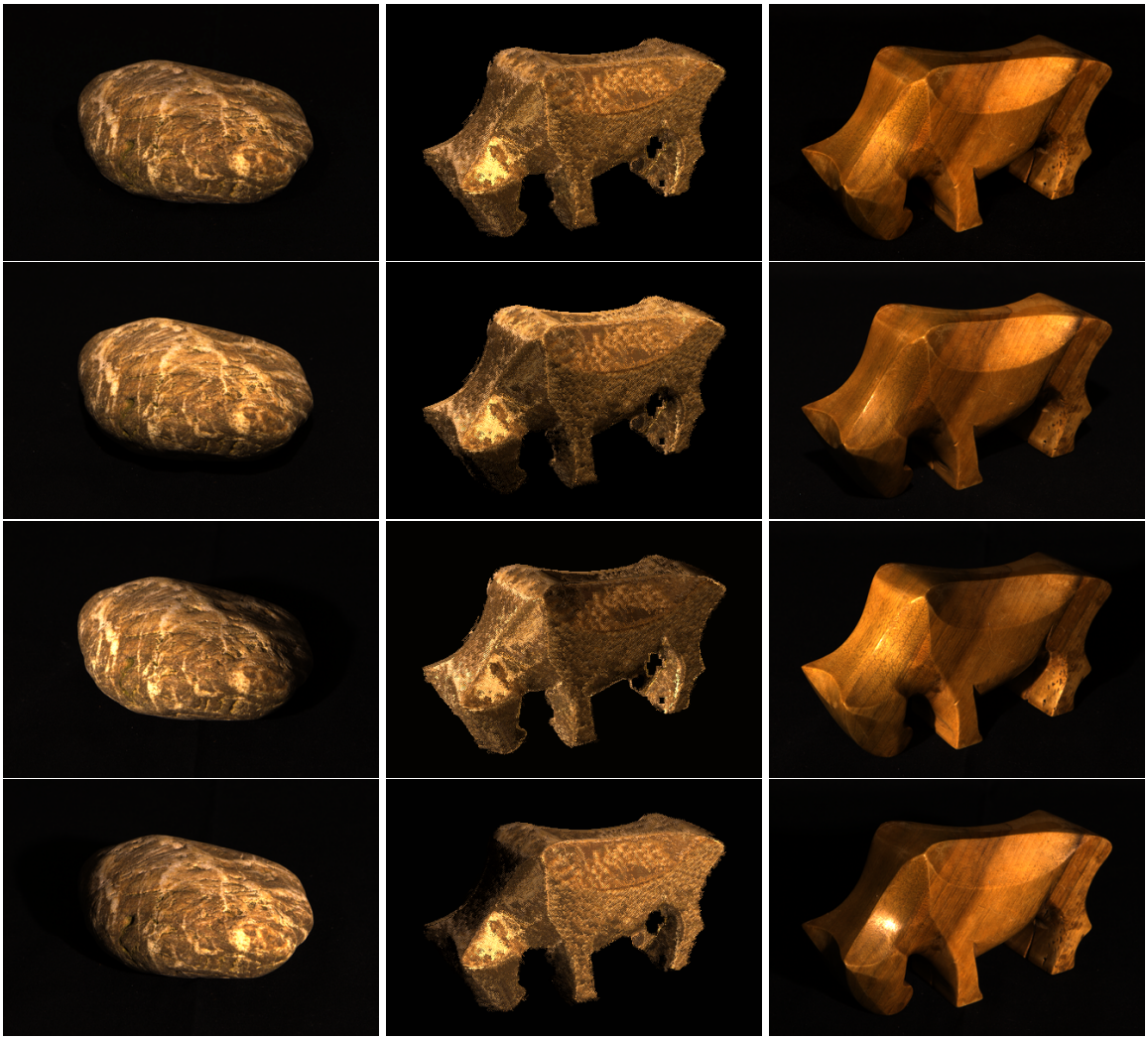

Figure A.5.: Left: source object. Right: destination object. Middle: the mapped result. Method: Region Growing over normals, growing constraint of 0.08, patch size of 7×7. Light (from top to bottom): Median of intensities, from the top, from the left and from the right.

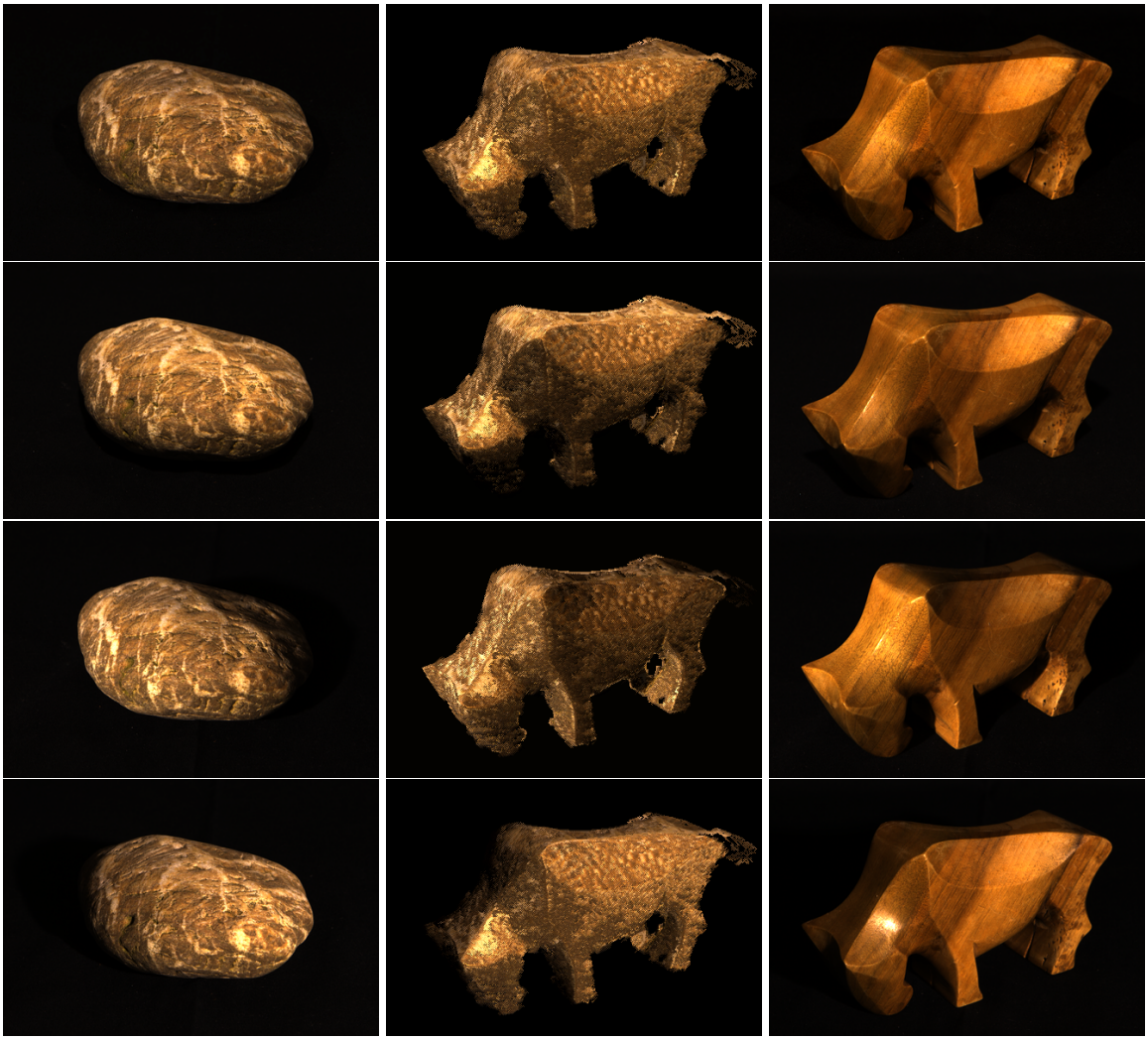

Figure A.6.: A curious result of overgrowing the region. Left: source object. Right: destination object. Middle: the mapped result. Method: Region Growing over normals, growing constraint of 0.09, patch size of 7×7. Light (from top to bottom): Median of intensities, from the top, from the left and from the right.

B. Acknowledgements

This appendix lists all foreign code used in this thesis and states its source.

The *image.h* file contains code for reading and writing images in ppm and pfm format. The code was written for the Visual Computing lecture by Jun.-Prof. Dr. Martin Fuchs at University of Stuttgart and is used with permission. This code was used for most image input and output related operations.

Several programs are using the *openCV* Open Computer Vision Library[1], to be able to display and manipulate images. The *createOne()* function in *comap.cpp* is utilizing the openCV library to combine multiple images so that they can be shown side-by-side and was adapted from the following source[2].

Parts of the main function in *lmv.cpp* which reads a sequence of jpeg images from stdin using *openCV* were adapted from here[3].

The calculation of correspondences is mostly performed by utilizing the *FLANN* Fast Library for Approximate Nearest Neighbor[4] functionality.

Other algorithms adapted include Patch Match by Barnes et al. [BSFG09] and Region Growing by Adams et al. [AB94]

[1] http://opencv.org/
[2] http://answers.opencv.org/question/13876/read-multiple-images-from-folder-and-concat/
[3] http://stackoverflow.com/questions/20899511/opencv-load-image-video-from-stdin
[4] http://www.cs.ubc.ca/research/flann/

Bibliography

[AB94] R. Adams, L. Bischof. Seeded Region Growing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(6):641–647, 1994. doi:10.1109/34.295913. URL http://dx.doi.org/10.1109/34.295913. (Cited on pages 10 and 57)

[AX10] D. G. Aliaga, Y. Xu. A Self-Calibrating Method for Photogeometric Acquisition of 3D Objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(4):747–754, 2010. doi:10.1109/TPAMI.2009.202. (Cited on page 10)

[BJK07] R. Basri, D. Jacobs, I. Kemelmacher. Photometric Stereo with General, Unknown Lighting. *Int. J. Comput. Vision*, 72(3):239–257, 2007. doi:10.1007/s11263-006-8815-7. URL http://dx.doi.org/10.1007/s11263-006-8815-7. (Cited on page 9)

[BRB09] D. Bradley, G. Roth, P. Bose. Augmented reality on cloth with realistic illumination. *Mach. Vision Appl.*, 20(2):85–92, 2009. doi:10.1007/s00138-007-0108-9. URL http://dx.doi.org/10.1007/s00138-007-0108-9. (Cited on page 11)

[BSFG09] C. Barnes, E. Shechtman, A. Finkelstein, D. B. Goldman. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), 2009. (Cited on pages 10, 15, 16 and 57)

[GBCHS10] D. B. Goldman, B. B. Curless, A. Hertzmann, S. M. Seitz. Shape and Spatially-Varying BRDFs from Photometric Stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(6):1060–1071, 2010. doi:10.1109/TPAMI.2009.102. (Cited on page 9)

[LPRM02] B. Lévy, S. Petitjean, N. Ray, J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371, 2002. doi:10.1145/566654.566590. URL http://doi.acm.org/10.1145/566654.566590. (Cited on page 11)

[ML09] M. Muja, D. G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pp. 331–340. INSTICC Press, 2009. (Cited on pages 10 and 14)

[ML12] M. Muja, D. G. Lowe. Fast Matching of Binary Features. In *Computer and Robot Vision (CRV)*, pp. 404–410. 2012. (Cited on pages 10 and 14)

[ON12] G. Oxholm, K. Nishino. Shape and reflectance from natural illumination. In *Proceedings of the 12th European conference on Computer Vision - Volume Part I*, ECCV'12, pp. 528–541. Springer-Verlag, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-33718-5_38. URL http://dx.doi.org/10.1007/978-3-642-33718-5_38. (Cited on page 9)

[TGG12] D. Tingdahl, C. Godau, L. V. Gool. Base Materials for Photometric Stereo. In A. Fusiello, V. Murino, R. Cucchiara, editors, *ECCV Workshops (2)*, volume 7584 of *Lecture Notes in Computer Science*, pp. 350–359. Springer, 2012. URL http://dblp.uni-trier.de/db/conf/eccv/eccv2012w2.html#TingdahlGG12. (Cited on page 10)

[UMKT12] A. Umakatsu, T. Mashita, K. Kiyokawa, H. Takemura. Touch-n-Paste: Direct texture transfer interaction in AR environments. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pp. 325–326. 2012. doi:10.1109/ISMAR.2012.6402596. (Cited on pages 11 and 50)

[UMKT13] A. Umakatsu, T. Mashita, K. Kiyokawa, H. Takemura. Pinch-n-Paste: Direct texture transfer interaction in augmented reality. In *Virtual Reality (VR), 2013 IEEE*, pp. 73–74. 2013. doi:10.1109/VR.2013.6549369. (Cited on pages 11 and 50)

[Woo80] R. J. Woodham. Photometric Method For Determining Surface Orientation From Multiple Images. *Optical Engineering*, 19(1):191139–191139–, 1980. doi:10.1117/12.7972479. URL http://dx.doi.org/10.1117/12.7972479. (Cited on pages 9 and 13)

[ZT10] Y. Zheng, C.-L. Tai. Mech Decomposition with Cross-Boundary Brushes. In *Computer Graphics Forum (In Proc. of Eurographics 2010)*, volume 29, p. to appear. 2010. (Cited on page 11)

All links were last followed on April 1, 2014.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature