EXPLORING THE CONVERSION OF VIDEOS INTO 3D MODELS

by

Frances Chen

A thesis submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Applied Mathematics- Computer Science

at

Brown University

Adviser: James Tompkin
Reader: Matthew Harrison

Spring 2017

Contents

Contents **2**

1 Introduction **4**
1.1 Abstract . . . 4
1.2 Current Methods . . . 4
1.3 Exploring a Different Approach . . . 5
1.4 Task Workflow . . . 5

2 Video to Point Cloud **7**
2.1 Motivation . . . 7
2.2 Background . . . 7
2.3 Results . . . 8

3 Searching for a Similar Model **11**
3.1 Motivation . . . 11
3.2 Background . . . 11
3.3 Results . . . 12

4 Point Cloud Alignment **13**
4.1 Motivation . . . 13
4.2 Background . . . 13
4.3 Results . . . 14

5 Texture Extraction **18**
5.1 Motivation . . . 18
5.2 Preprocessing: Finding a Clear Image . . . 18
5.2.1 Motivation . . . 18
5.2.2 Algorithm . . . 18
5.2.3 Results . . . 19
5.3 Intrinsic Image Decomposition . . . 19
5.3.1 Background . . . 19
5.3.2 IID Algorithm Overview . . . 20
5.3.3 Constraints . . . 21
5.3.4 Optimization . . . 23
5.3.5 Automatic Thresholding . . . 24
5.3.6 Results . . . 24

6 Summary **26**

6.1 Current Progress 26

Bibliography **28**

Chapter 1

Introduction

1.1 Abstract

The goal of my research project is to find a method of converting 2D video into a complete, relightable 3D model. The primary existing method for video to 3D model conversion, Structure from Motion(SfM), results in a noisy, incomplete model that is affected by the lighting conditions in the video. This paper attempts to create a model without these problems by first breaking the problem down into the separate tasks of finding a template to represent the shape of the object and finding the lighting invariant color, then combining these results. We approach this by taking advantage of existing 3D models available online, and methods to remove illumination.

1.2 Current Methods

The currently method for converting video into 3D models is Structure from Motion(SfM). This technique takes in sequential frames of a video, and detects matching feature points in the frames, such as corners or edges. Through these feature points, it estimates the camera geometry and the geometry of the object in the video. While this method is able to convert stereo images or videos into 3D meshes, the meshes they generate are not relightable and are often noisy or incomplete. Figure 1.1 shows an example of a 3D model reconstructed from a video. As you can see, parts of the chair's arms and legs are missing, and the surfaces look lumpy. Additionally, this model would not be relightable. In the environment the video was taken, the lighting shone on the front of the chair, so the front is much lighter than the back of the chair, even though the material is the same. Since this contrast in illumination is ingrained in this reconstructed model, this chair model would not look natural in a new environment where the light is coming from the back. This makes this reconstruction technique unusable for many of the applications that 3D models are commonly used for, such as in virtual reality environments or in video games.

While there are ways to generate more complete 3D models, they require specialized equipment such as a 3D scanner or a Kinect. Since video cameras are much more accessible and widely available, being able to generate a usable 3D model via video would make 3D models accessible to even those who do not

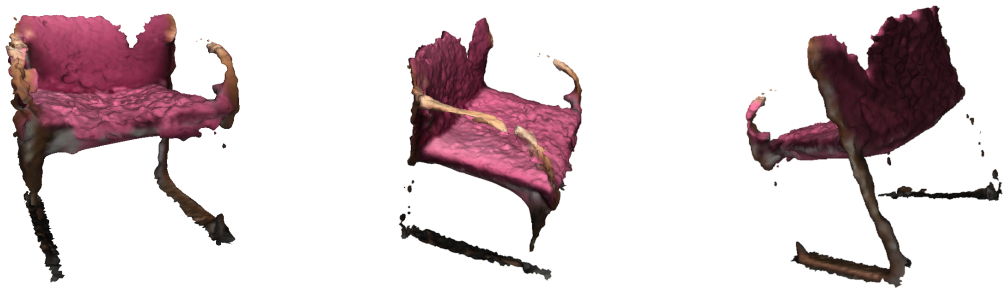


Figure 1.1: 3D mesh constructed from video via Structure from Motion

have access to 3D scanners. In this paper, we explore ways to convert videos into full 3D meshes by using intermediate results such as point clouds to search for a similar mesh.

1.3 Exploring a Different Approach

Since the task of converting 2D video directly to a relightable 3D models is complex and not well explored previously, this paper introduces a possible pipeline for this task, and explores the feasibility of this pipeline. This pipeline is described in the next section. The pipeline attempts to use existing methods and algorithms that would aide in steps of the problem. However, since the methods described are not specifically designed for this conversion task, we explore their use in the context of our problem and the challenges we face in doing so.

We will use the running example of a video of a chair throughout this paper. A chair was chosen because they are relatively simple yet can have many stylistic varieties. Additionally, it is a large category in the ShapeNet 3D mesh database, and it is helpful to have a large selection of models for this conversion problem.

1.4 Task Workflow

The pipeline of this problem is visualized in figure 1.2. This pipeline is designed to take advantage of existing computer vision algorithms. The input of the pipeline is a regularly recorded 2D video that tries to capture different viewpoints of a foreground object.

In the leftmost branch of the pipeline, we are trying to find a similar 3D model in our 3D model database. The first step for this could be an object recognition algorithm, so we know which category of objects in the 3D model database to look through. Then we could generate a 3D point cloud as a shape representation of the object in our video. This shape representation could be used to search for a 3D model by comparing it to the models in the database and finding the most similar one.

In the center branch of our pipeline, we generate a point cloud from the video. For comparisons between the chair in our video and the model we found from the left pipeline, it is useful to align the two point clouds. We use the Iterative Closest Point(ICP) algorithm, with preprocessing steps to optimize the algorithm for the task in our pipeline.

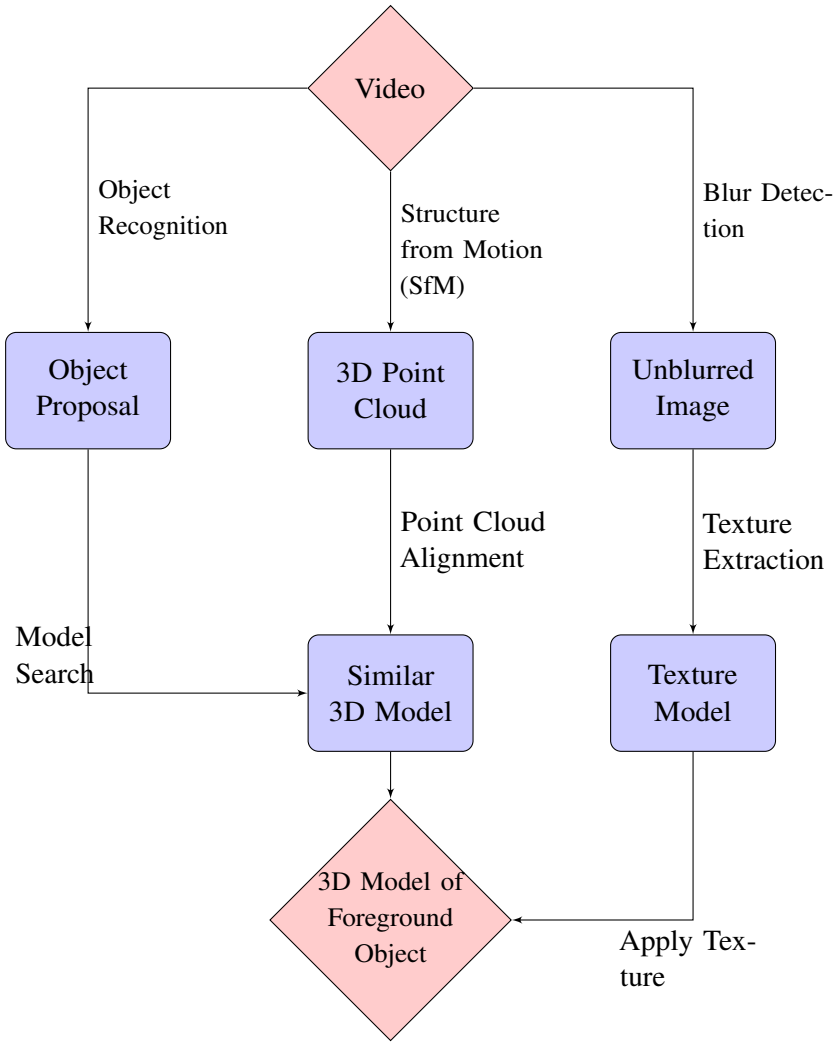


Figure 1.2: A flowchart of the processes and intermediate results to convert a video into a 3D model.

These first two pipelines together give us a shape template that represents our object. The next step is to figure out what color or texture to overlay on this template.

In the right branch of our pipeline, we work on extracting the lighting invariant texture. Our first step is to use a blur detection algorithm to find a clear image among our frames. Then we can use an intrinsic image decomposition algorithm to remove illumination and find the intrinsic color of the object.

After we complete these three branches of our pipeline, we will have a shape template of our object as well as its lighting invariant texture. Applying this texture to the template gives us a complete, relightable model.

Chapter 2

Video to Point Cloud

2.1 Motivation

This section will give us a shape representation that we can use in the later steps of the pipeline described by this paper, as explain in more detail the existing method for video to 3D model conversion. Existing Structure from Motion(SfM) algorithms are able to generate a 3D point cloud from the frames a 2D video. The 3D point cloud generated from this method will be an incomplete representation of the object, but it allows us to have a rough model of the 3D shape of an object, which can then be used in other methods to build a more complete model.

The point cloud generated can then be used to generate a mesh. This is done by estimating the surface normals and generating surfaces based on these estimates.

2.2 Background

3D Reconstruction from Multiple images
In the classic paper Building Rome in a Day [1], the authors introduce the concept of using a large collection of images of the same object, taken with different lighting and at different angles, to reconstruct a 3D point cloud of an object. This allows us to simply search for a term such as “rome” on in image search, and use the 2D image results of the search to reconstruct the structure of an object.

Structure from Motion(SfM)
A similar area of research to 3D reconstruction from multiple images is the Structure from Motion. This method takes in sequential frames, such as those extracted from video to estimate the 3D structure of objects. Since frames are sequential, the tracking of important features such as corners and motion indicators allows the algorithm to more effectively calculate the proportional distances between difference parts of the image and construct a 3D point cloud.

2.3 Results

To test the effectiveness of Structure from Motion as a step of my 3D model reconstruction pipeline, I recorded a video of a chair. I circled the chair to capture different angles and viewpoints of it in order for the algorithm to calculate a more robust point cloud. Some frames from the video are shown below in figure 2.1.

From the video, I sampled frames evenly to get 80 sequential frames. I then ran the sparse point matching algorithm in Photoscan to reconstruct the point cloud in figure 2.2.

To achieve a more complete and detailed 3D point cloud, I additionally used a dense point cloud reconstruction in Photoscan. This method uses the sparse point cloud and images to match as many pixels between images as possible, giving us the much more detailed model we see in figure 2.3. For further processing purposes, I hand segmented the point cloud of the chair from the scene. Views of the segmented dense point cloud can be seen in figure 2.4.

The dense point cloud can be used to generate a 3D mesh, which is what we are trying to create in this project. However, the mesh it creates has many gaps and holes, and many surfaces look lump, as we can see in figure 2.5.

Figure 2.1: Frames from the video used to reconstruct 3D point cloud.

Figure 2.2: Views of the sparse point cloud reconstruction.

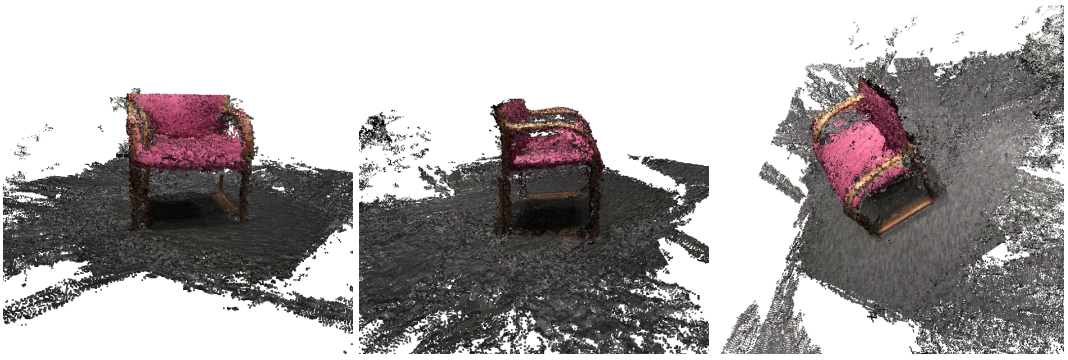


Figure 2.3: Views from the dense point cloud construction.

Figure 2.4: Views from the dense point cloud reconstruction.

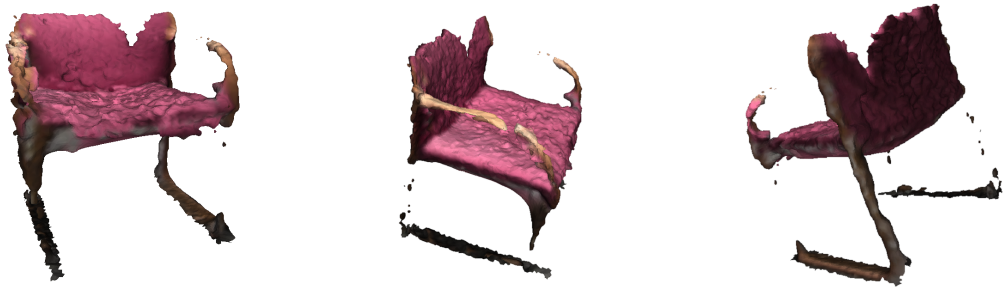


Figure 2.5: 3D mesh constructed from video via Structure from Motion

Chapter 3

Searching for a Similar Model

3.1 Motivation

In Chapter 2, we saw that the method of Structure from Motion algorithms to create a 3D model leads to models that are incomplete or lumpy. These models give us a good idea of the shape of the object, but would require a lot of human editing before they could be used in animation or scene generation. To avoid this problem, we take advantage of the large databases of 3D models to find a similar model to be used as a template. This template can then be tweaked with small modifications to look like the object in a video.

3.2 Background

3D Mesh Databases

There now exists a prevalence of 3D models available online. One such data base is the ShapeNet [2] database. The database has over 50,000 3D models organized by the type of object, mostly of common manmade objects such as airplanes, chairs, cups, tables, towers, and trains. This reminds us of the task of object recognition– if we can determine what an object is, it will greatly reduces the number of meshes we must search from.

Image Object Recognition

The classic computer vision problem of object recognition would be the first step in narrowing our model search. By recognizing the object in the foreground of a frame of a video, we can narrow down our model search greatly by only looking at the part of the database for our recognized object. For example, by recognizing that our example video is focused on a chair, we would only need to search chair models.

2D to 3D Alignment Problem Using CAD Models

The problem of 2D to 3D alignment has been explored using an exemplar-based 3D category representation that can model chairs of different styles and viewpoints. [3] By learning visual elements of 3D models through the rendering of different viewpoints, the authors are able to match a 3D model of a chair

Figure 3.1: Views from a hand chosen 3D chair model similar to the chair captured in video.

Figure 3.2: Views of 3D point cloud generated by the Poisson Disk sampling of the model.

with one seen in a 2D image. However, this problem of 2D to 3D matching is a much harder problem, and the 3D shape information gained from a video would allow more accurate matches.

3D Native Methods for Matching

By using the point cloud or mesh generated from video, we can take advantage of 3D methods for object recognition or model matching. Current methods rely on CNNs to classify 3D information, either with the use of multi-view or volumetric representations. [4] This method could be used not just for object classification, but also for style classification. The style classification could also be done with 3D template matching, where a template could be created from models of the same style. [5]

3.3 Results

Due to time constraints, this paper does not explore the methods described above. For the purposes of working on other steps of the pipeline, a similar model was handpicked(figure 3.1) to be the template. Because this model is to be used to matched with the point cloud from the video, the 3D mesh is sampled using the Poisson disk sampling process to generate the point cloud seen in figure 3.2.

Chapter 4

Point Cloud Alignment

4.1 Motivation

Given two similarly shaped point clouds, aligning their position and orientation would allow us to compare them easily and match their features. For our problem of 3D model creation using video, we could use this to determine transformations that would morph the point cloud of the selected template to the shape of the point cloud generated by the video. After determining these transformations, we could apply the transformations to the selected template itself to tweak its details into the shape of the object in the video.

4.2 Background

Iterative Closest Point

The Iterative Closest Point(ICP) algorithm [6] is a way to align one point cloud to another using only rigid transforms– translations, rotations, or reflection. This algorithm is useful in helping us orient and shift our point cloud to align with our model. The nonrigid version can be useful in helping modify our model further.

3D Morphable Model for Faces

A recent and exciting development in the area of 3D modeling is the 3D Morphable Model [7] for human faces. While regular 3D meshes consist of vertices and faces, they can be hard to modify in a realistic manner. The 3D Morphable Model is a 3D model of the human face that is derived from transforming the shapes and textures of a set of 3D face models into a vector space representation. Then, given images of a specific person, the model can be tweaked to include the specific details of a person's face seen in the image by taking a linear combination of the example models. This concept can be used on not just human faces, but also more general objects by using a collection 3D models of a certain object to create a morphable model.

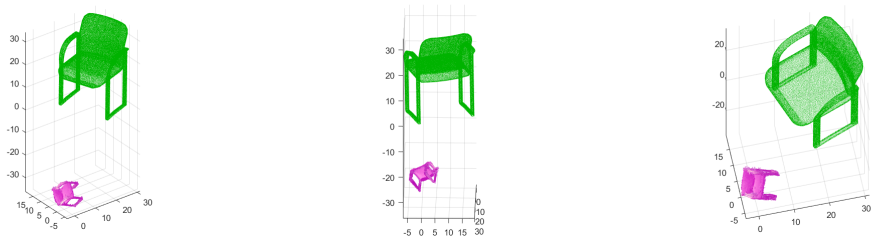


Figure 4.1: Views of the original point clouds. The green point cloud is generated from the 3D Model and the pink point cloud is generated from the video.

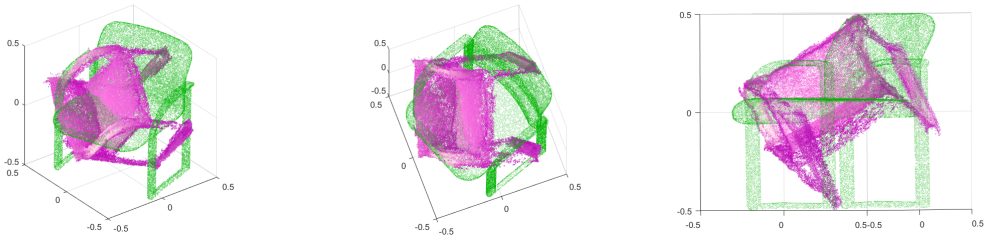


Figure 4.2: Views of the point clouds after normalization to be in the same unit cube.

4.3 Results

Because the ICP algorithm only works if the point clouds are of around the same size and orientation, I developed preprocessing steps to be able to use the Iteratie Closest Point(ICP) algorithm to align my point clouds.

Because there was no constraint on the position or orientation of the template's point cloud or the video's point cloud, the two point clouds were originally very far apart and of different scales(figure 4.1). Since the ICP algorithm can only perform rigid transforms, it did not work well for a case like this where the two point clouds were of vastly different scales. To solve this, I normalized the point clouds to be in the same unit cube(figure 4.2). Since the rotation is a rigid transform that the ICP algorithm is capable of performing, I thought that the algorithm would be able to match these normalized point clouds. However, ICP returned the lackluster results seen in figure 4.3 due to the fact that it is very dependent on the starting orientations of the point clouds.

To find the correct starting orientation, I rotated the template point cloud in all 24 possible orientations that are 90 degrees apart. Some of these orientations are shown in figure 4.4. With each of these 24 starting orientations, I ran the ICP algorithm with the video point cloud. After the ICP converges, it outputs a Root Mean Square Error(RMSE) that is an estimate of how far apart the point clouds are from each other. By assuming that the correct starting orientation would lead to the lowest RMSE, I chose the orientation with the lowest RMSE as the orientation to use. After this, the ICP algorithm was quite effective in aligning the point clouds. We can see the before in figure 4.5 and the after in 4.6. In figure 4.5, the seat and legs are not aligned, but after ICP, the seats and legs of the two different chairs look parallel. This signals that this process was able to effectively align our two point clouds.

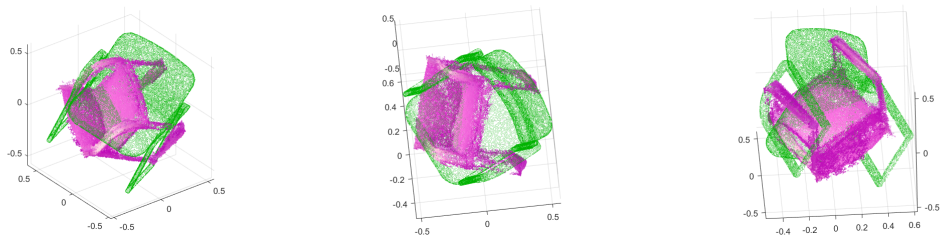


Figure 4.3: Views of the result after running ICP on the normalized point clouds in figure 4.2.

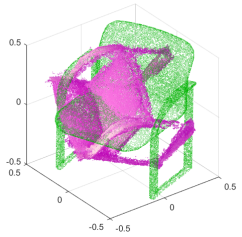
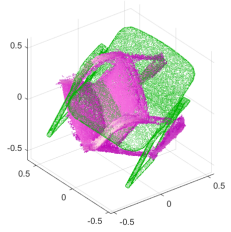
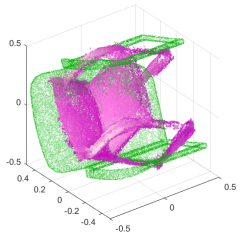
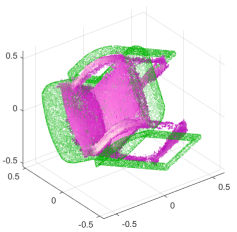
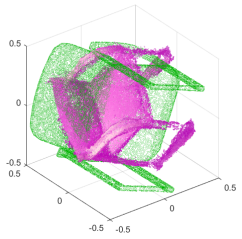
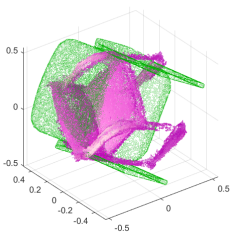
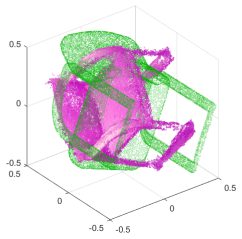
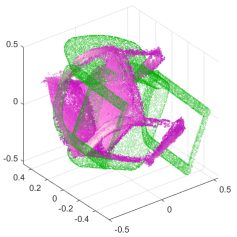
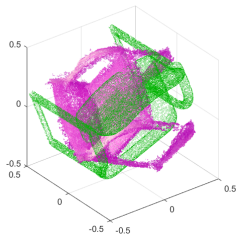
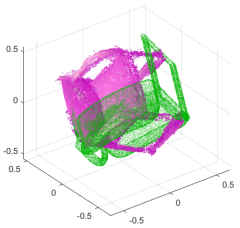
Transform (x, y, z)	Pre-ICP	Post-ICP	RMSE (Post-ICP)
$(0, 0, 0)$			0.1526
$(\frac{\pi}{2}, \frac{\pi}{2}, 0)$			0.0663
$(\frac{3\pi}{2}, \pi, 0)$			0.1406
$(\frac{\pi}{2}, 0, \frac{\pi}{2})$			0.1413
$(\frac{3\pi}{2}, 0, \frac{3\pi}{2})$			0.1295

Figure 4.4: Five of the 24 orientations tried to determine the best starting orientation. The Transform column gives the rotations around the x, y, and z axes to achieve this orientation, the Pre-ICP column shows the model(green) and video(pink) point clouds before running ICP, and the Post-ICP column shows them after running ICP. The RMSE, a indicator of how close the points are after alignment, is given in the last column. We see that the transform in the second row performs the best by far.

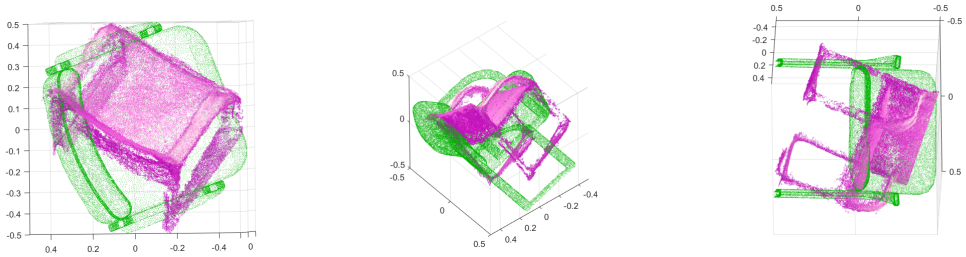


Figure 4.5: Additional views of the automatically chosen model point cloud(green) orientation with the video point cloud(pink)

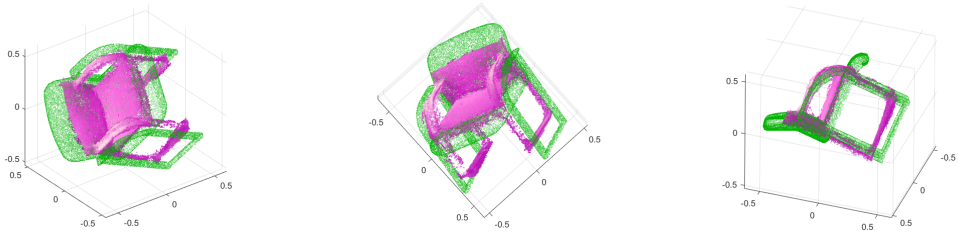


Figure 4.6: Views of the result after running ICP on the preprocessed point clouds in figure 4.5.

Chapter 5

Texture Extraction

5.1 Motivation

After we have a similar to model to use, we would like to capture the details of the original image such as color and texture. Since all natural photographs have some amount of lighting as well as color variation based on the orientation of an object, simply taking a crop of the texture from the image is usually ineffective. For example, if we look at the frames of the video of our chair in figure 2.1, we can see that the back of the chair appears to be much darker than the front of the chair. We know that the front and the back of the chair is made of the same color cloth– it is only the lighting conditions that cause it to look darker in the image. These variations in the color of the chair in the image are caused by the chair's positioning and lighting conditions. However, if we want to created a model of the chair, we would want to know the chair cloth's original color invariant of these conditions– the intrinsic color. This reason holds when we want to create a 3D model of an object. An ideal 3D model should be adaptable to different lighting conditions. For this to be true, we wish for it to have neutral lighting– so that lighting can be added later dependent on the environment that the model is to be put in.

5.2 Preprocessing: Finding a Clear Image

5.2.1 Motivation

To have an accurate sample of the color invariant texture, it is best to start with a clear image. Since our frames are from a video, the clarity of some of the frames may be compromised by the camera motion. Thus we need a way automatically find a clear frame from our set of video frames.

5.2.2 Algorithm

To find a clear frame, I used the method described in "Three-Stage Motion Deblurring from a Video." [8] This method is summarized below.

We can use image statistics as a measure of how blurred and image is. For example, if we convolve the horizontal gradient operator $d_1 = [1, -1]$ with an image we can detect horizontal blur. Since edges will be sharper in a clear image than in a generally blurry image, the values in the convolved sharp image will be relatively higher than the values in the convolved blurry image.

However, a horizontal gradient operator will only be able to detect horizontal motion blur. Since our camera motion may be in any direction, we want to prepare for motion blur in any direction. Thus we will use an additional three directions for a total of four gradient operators: $d_1 = [1, -1]$, $d_2 = [1, -1]^\intercal$, $d_3 = [-1, 0; 0, 1]$, $d_4 = [0, -1; 1, 0]$. These are the horizontal, vertical, and Roberts gradient operators, respectively. The Roberts gradient operators can be thought of diagonal gradient operators that help detect blur in both diagonal directions.

For each image I_t in the set of images we are looking through, we convolve each gradient operator d_i with that image to get image gradient $d_i * I_t$. To get the distribution of these image gradients, we sort the values of each image gradient into n bins, to create discrete distribution $P(d_i * I_t)$. This distribution will have n quantities on the x-axis of the distribution, and P_k will be the kth quantity.

The unblurred frame I_s will then be the frame that maximizes the total value of these logarithmic distributions:

$$I_s = \arg\max_{I_t} \sum_{i=1}^{4} \sum_{k=1}^{n} \log P_k(d_i * I_t)$$

In this formula, the logarithmic operation helps to magnify the differences between the distributions being compared. This method allows to extract relatively unblurred frames.

5.2.3 Results

After running the algorithm on the 80 frames from my video, I took the top 3 and bottom 3 results to analyze this method's effectiveness. In figure 5.1, it is not obvious that the method is effective because both the top row and bottom row of images seem to look clear. This is because the frames were high resolution, making it hard to notice the blur unless we look closely. To better see the results, we zoom in on the same location in each frame in figure 5.2. Here it is easier to see that the photos in the top row are clearer– we can clearly see the stripes in the carpet, whereas the pattern of the carpet becomes a blur in the photos of the bottom row.

5.3 Intrinsic Image Decomposition

5.3.1 Background

After finding a clear image, we now take an in depth look at how we can remove illumination from an image. The well-studied problem of intrinsic image decomposition addresses this problem. This problem is based on the idea that an image I is composed of reflectance R and shading S components such that $I = R \times S$, where $\times$ denotes the pixelwise multiplication of reflectance of shading. The reflectance component R is the intrinsic color of a surface, which is invariant to the lighting and orientation of the object. The shading component S signifies the amount of light reflected from a surface, and is usually largely dependent on lighting conditions and orientation.

Figure 5.1: The three clearest images(top) and three blurriest images(bottom) found by the algorithm.

Figure 5.2: Magnified portions of the three clearest images(top) and three blurriest images(bottom) found by the algorithm.

Intrinsic image decomposition, though having been extensively studied, is still a challenging problem. Given an image I with a total of M pixels, we are trying to solve for $2M$ variables, the M variables of reflectance and the M variables of shading. Because this is an ill-posed problem, current algorithms attempting to solve it rely on assumptions about the reflectance and shading of natural images.

5.3.2 IID Algorithm Overview

To work on this problem for my project, I implemented the algorithm in a paper titled "A Closed-Form Solution to Retinex with Nonlocal Texture Constraints." [9] The solution from this paper is able to produce good results with a single 2D image, whereas many other solutions rely on a 3D image captured from a kinect or multiple views from different images. The paper uses the common retinex constraint – the assumption that the changes in shading and reflectance are small between neighboring pixels– as well as a nonlocal texture constraint, that is then solved for using a closed form solution.

The authors formulate the decomposition problem as the minimization of an objective function that

incorporates the constraints as follows:

$$\arg\min_S F(S) = \lambda_l f_l(S) + \lambda_r f_r(S) + \lambda_a f_a(S)$$

where λ_l, λ_r, and λ_a are positive weights (set to 1, 1, and 1000, respectively, in our implementation) of the three different objective functions. Function f_l is a local term for the retinex constraint, f_r is the function of the nonlocal texture constraint, and function f_a is the absolute scale constraint.

As mentioned above, the intrinsic image decomposition problem assumes an image I is the pixelwise multiplication of reflectance R and shading S. This means that given a pixel p, the value I_p of a pixel can be broken down into $I_p = S_p \times R_p$. Taking the logarithm of both sides, we get $\log I_p = \log S_p + \log R_p$. For simplicity of notation, we reuse I_p, S_p, and R_p to represent their log values, giving us $I_p = S_p + R_p$. Since $R_p = I_p - S_p$ and we are given I, each of the three functions in our objective function can be formulated to only be dependent on S by substituting $I_p - S_p$ whenever needed. We go over the formulation of these three constraint functions below.

5.3.3 Constraints

Retinex Constraint

The retinex constraint uses the assumption that differences in shading and reflectance between neighboring pixels will be small. We formulate this with a constraint that minimizes the shading and reflectance between adjacent pixels with the following formula:

$$f_l(S) = \sum_{(p,q)\in N} [(S_p - S_q)^2 + \omega_{(p,q)}(R_p - R_q)^2$$

where N denotes the set of all adjacent pixels. Minimizing this function helps us achieve our goal of achieving smoothness in reflectance and shading. We balance reflectance smoothness with the term $\omega_{(p,q)}$, which we define as:

$$\omega_{(p,q)} = \begin{cases} 0 & \text{if } ||\hat{R}_p - \hat{R}_q||_2 > t \\ 100 & \text{otherwise} \end{cases}$$

where $\hat{R}_p$ and $\hat{R}_q$ represent the chromaticities of pixels p and q, and t is a threshold. Chromaticity is the intensity-normalized color of a pixel. This formulation of $\omega_{(p,q)}$ means that if there is a large change in chromaticity, we allow large variations in reflectance by weighing the reflectance term at 0. Otherwise, we assume that the gradient is due to shading by enforcing a heavy penalty for small changes in reflectance.

To get an equation that is only dependent on S, we substitute R_p with $I_p - S_p$ in the original equation.

$$\begin{aligned}
f_l(S) &= \sum_{(p,q)\in N} [(S_p - S_q)^2 + \omega_{(p,q)}((I_p - S_p) - (I_q - S_q))^2] \\
&= \sum_{(p,q)\in N} [(S_p - S_q)^2 + \omega_{(p,q)}((I_p - I_q) - (S_p - S_q))^2] \\
&= \sum_{(p,q)\in N} [(S_p - S_q)^2 + \omega_{(p,q)}((I_p - I_q)^2 - 2(I_p - I_q)(S_p - S_q) + (S_p - S_q)^2)] \\
&= \sum_{(p,q)\in N} [(1 + \omega_{(p,q)})(S_p - S_q)^2 - 2\omega_{(p,q)}(I_p - I_q)(S_p - S_q) + \omega_{(p,q)}(I_p - I_q)^2]
\end{aligned}$$

Nonlocal Texture Constraint

Because the intrinsic image decomposition problem is ill-posed, with many more unknowns than equations, we seek strategies that will help reduce the number of unknowns. The strategy used in this paper is the analysis of texture. Based on the theory of Markov Random Fields, pixels with the same surrounding neighborhoods can be expected to have the same value. In this algorithm, we make use of this knowledge by assuming pixels to have the same reflectance if they have similar chromaticity texture matrices.

By grouping our pixels with similar texture to have the same texture, we often able to greatly reduce the number of unknowns in our problem. If we originally had an image with M total pixels, we would have had M unknowns for reflectance– one for each pixels. If we group our pixels into N groups, where each group has similar neighboring texture, the assumption that each of these pixels has the same reflectance means that we now have N unknowns for reflectance. Because there are typically only a few different textures in a natural image, usually $N << M$. Thus this process greatly reduces our number of unknowns from $2M$ to $M + N$, often leading to much more accurate solutions to the decomposition problem.

For pixels in the same group, we penalize differences in their reflectances with the following forumulation:

$$f_r(S) = \sum_{G_r^i \in \Gamma_r} \sum_{p,q \in G_r^i} (R_p - R_q)^2$$

where Γ_r is our set of pixel groups, and G_r^i is the ith pixel group.

We now rewrite this formula in terms of S with the substitution $R = I - S$:

$$\begin{aligned} f_r(S) &= \sum_{G_r^i \in \Gamma_r} \sum_{p,q \in G_r^i} ((I_p - S_p) - (I_q - S_q))^2 \\ &= \sum_{G_r^i \in \Gamma_r} \sum_{p,q \in G_r^i} ((I_p - I_q) - (S_p - S_q))^2 \\ &= \sum_{G_r^i \in \Gamma_r} \sum_{p,q \in G_r^i} ((I_p - I_q)^2 - 2(I_p - I_q)(S_p - S_q) + (S_p - S_q)^2) \end{aligned}$$

The object function above depends on pixels being grouped together properly, which may not be the case. To account for improper matches that may exist in a group, we introduce the following soft grouping algorithm that assigns a confidence weight signaling our confidence that the pixel is indeed in the correct group.

Nonlocal Texture Constraint: Soft Grouping Algorithm

The paper's grouping algorithm looks at 3×3 windows centered on each pixel of the image. Starting off with all unmatched pixels, we iteratively select an unmatched pixel and compare its window to the windows of each of the other pixels until we find a match. We define a match to be a pixel whose window has a sum of squared differences less than a specified threshold $t_g = 0.01$. We additionally compare windows rotated by 90, 180, and 270 degrees to facilitate matching. After all of our pixels have been grouped in this manner, we go through each of our groups to calculate the confidence that each pixel truly belongs in that group by measure its local similarity to the rest of the group. For each group, we calculate the average 5×5 window and the average 7×7 window. For each pixel p in that group, we compare p's

windows to the larger average windows using the same SSD threshold matching technique that we used earlier. If the largest window p matches is of size $K \times K$ we record this as $m_p = K$. We then use this value as well as the normalized SSD value of the original 3×3 match to calculate the grouping weight for pixel p as c_p:

$$c_p = m_p(1 - \frac{1}{9}SSD)$$

Using these grouping weights calculated from the soft grouping, we redefine our nonlocal texture objective function to be

$$f_r(S) = \sum_{G_r^i \in \Gamma_r} \sum_{p,q \in G_r^i} c_p c_q((I_p - I_q)^2 - 2(I_p - I_q)(S_p - S_q) + (S_p - S_q)^2)$$

Absolute Scale Constraint

The decomposition problem $I = R + S$ has an inherent ambiguity. For any valid decomposition, we can increase R by any value and decrease S by the same value to get another valid decomposition. In order to resolve this, we create an absolute scale for shading and reflectance. We can do this by anchoring down our brightest pixels to have a shading value of 1 with the following formula:

$$f_a(S) = \sum_{p \in G_a} (S_p - 1)^2$$

where G_a is the set of the brightest pixel(s) in the image.

5.3.4 Optimization

Now that we have defined the components of our object function $F(S)$, we can proceed in optimizing it to solve the decomposition problem. In each of our constraint functions, we only have quadratic or lower degree terms with respect to S. Thus, $F(S)$ is quadratic with respect to S so we can rewrite it in standard quadratic form as

$$\frac{1}{2}\mathbf{s}^\mathsf{T}\mathbf{A}\mathbf{s} - \mathbf{b}^\mathsf{T}\mathbf{s} + c$$

where $\mathbf{s}$ is a $M \times 1$ vector created by concatenating the shading of all pixels, and $\mathbf{A}$ and $\mathbf{b}$ are a symmetric $M \times M$ matrix and $M \times 1$ vector, respectively, containing the constants from our objective function.

From a quick glance, we can see functions f_l, f_r, and f_a, are all nonnegative. Furthermore, if the input image is not uniform, f_l will be positive definite because $(S_p - S_q)^2$ and $(R_p - R_q)^2$ will only both be zero if $I_p = I_q$. Thus, for all non-uniform images, f_l will be positive definite, meaning the matrix $\mathbf{A}$ is positive-definite.

We now know $\mathbf{A}$ is a symmetric positive-definite(SPD) matrix. This means that our quadratic function will have a global minimum, which is the solution of the linear equation

$$\mathbf{A}\mathbf{s} = \mathbf{b}$$

This gives us a closed form solution to our image decomposition problem, allowing us to solve for the shading vector $\mathbf{s}$ using a conjugate gradient algorithm. After we find the shading $\mathbf{s}$, we also have the reflectance R from the fact that $I = R \times S$.

5.3.5 Automatic Thresholding

This method contains one main parameter – the chromaticity threshold t in the retinex constraint. While our other parameters seem to work well for all images, t must vary according to each image if we want good results. Thus we use an automatic thresholding method based on entropy minimization. We run our algorithm with each of the 12 uniformly sampled values between 0.00001 and 0.005 as t. Then, we calculate the reflectance image with the lowest quadratic entropy, which we define as:

$$V = \sum_{i=0}^{255} p^2(i),$$

where $p(i)$ is defined as

$$p(i) = \frac{1}{N} \sum_{j=i-N/2}^{i+N/2} G(j, s^2)h(j)$$

where $h(j)$ denotes the value of each histogram bin, with j ranging between 0 and 255.

$p(i)$ can be thought of as the estimated probability that a pixel has reflectance value i, and is estimated with a Parzen window estimator using Gaussian kernels G with mean j and variance s^2. The parameter N controls how big a window we want to look at, and we set $N = 30$ in my implementation.

5.3.6 Results

Evaluation of Implementation

In order to evaluate the effectiveness of my implementation, I tested it on the same data set that was used in the paper I implemented. However, because I ran it on a personal computer, I faced memory constraints for larger images. To evaluate the effectiveness of my implementation despite this, I shrunk the evaluation images.

As seen in figure 5.3, my implementation is as efffective as the author's results in capturing the shading and reflectance of the test images. My results miss fine lines in the reflectance due to the fact that it was run on a smaller image. Some other differences are like caused by differences in optional parameters.

Results on Chair Example

After validating that my implementation was correct, I ran the decomposition algorithm on my chair example. Since I was still limited by size constraints, I cropped and shrunk part of a frame to get the leftmost image in figure 5.4. The decomposition was effective, as evidenced by the matching reflectance of the cloth backrest and seat of the chair, and the matching reflectance of the wooden legs and armrest. However, due to the size constraints, we get a lot of splotching as seen in the grey bits in the armrest and legs. To see what the decomposition of the armrest and backrest would look like if I was using the full resolution image, I decomposed crops of the armrest(center image in figure 5.4) and backrest(right image in figure 5.4). The results of these decompositions had pretty constant reflectance. This means that it was able to remove the lighting and orientation, which is what we wanted. These sections of lighting invariant color are useful for our problem of 3D model creation because we can use these patches as tiles to overlay on our model.

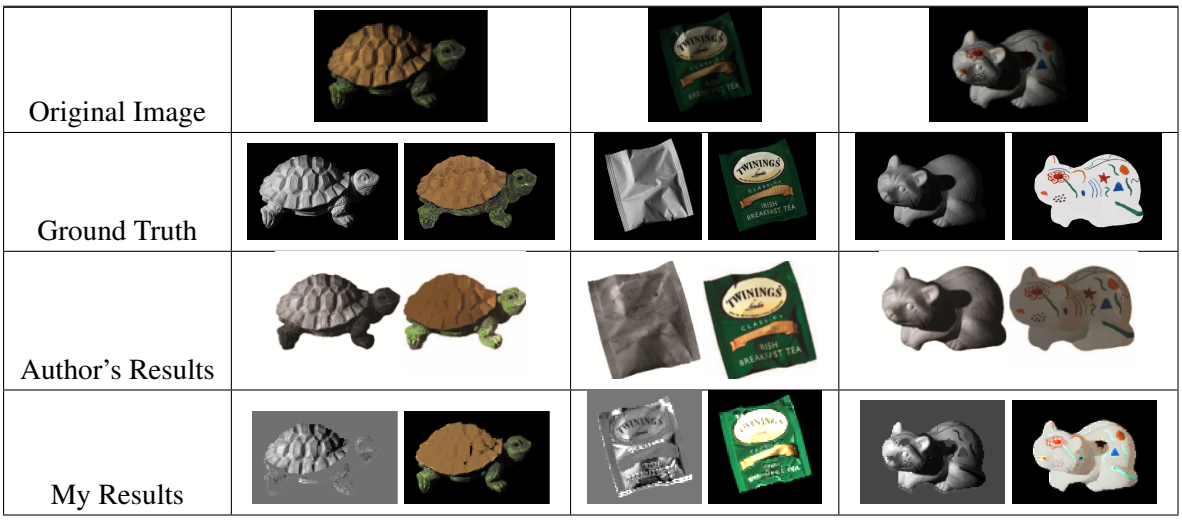


Figure 5.3: Evaluation of my implementation. The ground truth shading and reflectance for three images is compare to the shading and reflectance found from the author's implementation and my implementation of this decomposition algorithm. In each box, the shading is represented by the black and white image on the left, and the reflectance is represented by the color image on the right.

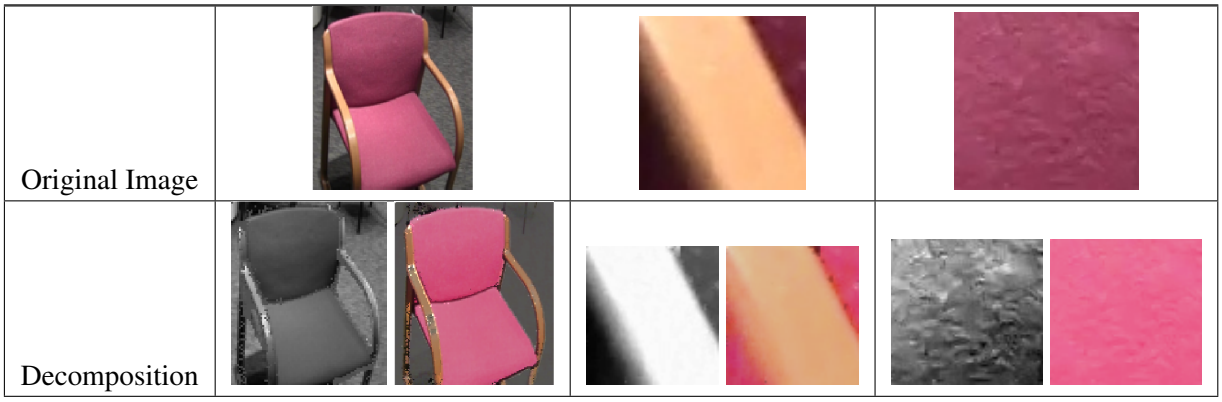


Figure 5.4: Decomposition of crops of a frame of the example video. Left: Image of the full chair. Middle: Cropped image of part of the armrest of the chair. Right: Cropped image of part of the backrest of the chair.

Chapter 6

Summary

6.1 Current Progress

Due to the time constraints of this thesis, this paper only tries and solves the steps of the pipeline highlighted in figure 6.1. In the past 4 chapters, we have show that we are able to take in video, turn it into a point cloud, align that point cloud to one of a similar model, and find the lighting invariant texture of the object.

This means that at this point, we can take in a video of an object and a 3D model similar to that model, and get out an aligned template and a texture model. To complete the rest of the pipeline, we would need to automate the model search process and automate the application of the texture to our template model.

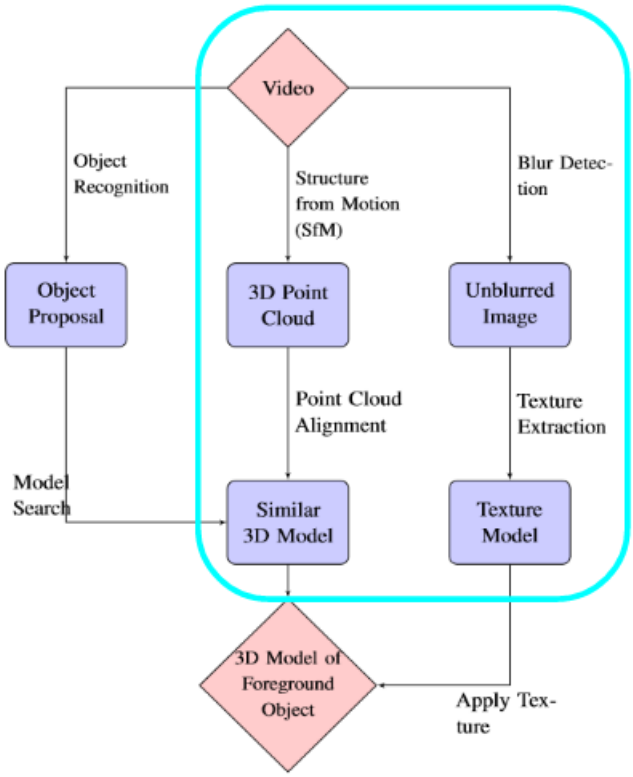


Figure 6.1: Components of the pipeline tried and solved in this paper are highlighted

Bibliography

[1] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. https://grail.cs.washington.edu/rome/rome_paper.pdf.

[2] Shapenet. https://www.shapenet.org/taxonomy-viewer.

[3] Mathieu Aubry, Daniel Maturana, Alexei A. Efros, Bryan Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. http://www.di.ens.fr/willow/research/seeing3Dchairs/.

[4] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2016.

[5] Frederic Jurie and Michel Dhome. Real time 3d template matching. https://hal.archives-ouvertes.fr/inria-00548278/document.

[6] Iterative closest point. https://en.wikipedia.org/wiki/Iterative_closest_point.

[7] Volker Blanz and Thomas Vette. A morphable model for the synthesis of 3d faces. http://gravis.dmi.unibas.ch/publications/Sigg99/morphmod2.pdf.

[8] Chunjian Ren, Wenbin Chen, and I fan Shen. Three-stage motion deblurring from a video. https://link.springer.com/chapter/10.1007/978-3-540-76390-1_6.

[9] Qi Zhao, Ping Tan, Qiang Dai, Li Shen, Enhua Wu, and Stephen Lin. A closed-form solution to retinex with nonlocal texture constraints. http://ieeexplore.ieee.org/document/6175903/.