A Material Segmentation and Classification System

by

Jennifer L. Wong

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author .....
Department of Electrical Engineering and Computer Science
May 24, 2013

Certified by .....
Edward H. Adelson
Professor
Thesis Supervisor

Certified by .....
James R. Bruce
Research Software Engineer, PhD
Thesis Supervisor

Accepted by .....
Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

A Material Segmentation and Classification System

by

Jennifer L. Wong

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2013, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I developed a material segmentation and classification system that takes in images of an object and identifies the material composition of the object's surface. The 3D surface is first segmented into regions that likely contain the same material, using color as a heuristic measure. The material classification of each region is then based on the cosine lobe model. The cosine lobe model is our adopted reflectance model, which allows for a simple approximation of a material's reflectance properties, which then serves as the material's unique signature.

Thesis Supervisor: Edward H. Adelson
Title: Professor

Thesis Supervisor: James R. Bruce
Title: Research Software Engineer, PhD

Acknowledgments

I thank Professor Adelson for advising my thesis and providing me with useful feedback. I also thank James Bruce and the Google Research group for their ongoing support, valuable insights, and resources that allowed me to complete this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Classifying materials – wood, plastic, fabric, etc. – has always been a popular problem in the field of computer vision. It is also a challenging problem, because the appearance of materials can change significantly under different viewing and lighting conditions. Fully representing a material requires a multi-dimensional function and a large collection of data points. Regardless, much research was done to take on this challenge, because material classification is useful for many scientific applications.

In this paper, I will describe the development of a material segmentation and classification system. One particular application that we aim to work toward is the material classification of apparel and other consumer products. Knowledge about a product's material can be very useful, especially for online shopping where a customer is not physically there to inspect the product. Another application that the project will contribute to is a computer graphics problem – the ability to generate accurate shaders for all common materials. In other words, rendering programs would be able to produce an accurate, realistic rendering of any material under any light condition; the only required user input would be the specification of the desired material. While this application does not involve classification, it does require a way to distinguish between materials, which is a discussed topic for this project.

We approach the problem of material classification by using computer vision and machine learning techniques to infer material types from images of objects. This involves gathering a training set that covers a sufficient variety of materials. This

is a plausible task, because we are targeting manufactured objects, which tend to draw from a small set of common materials. We also simplify the representation of a material, by adopting a simple reflectance model that approximates the multi-dimensional function mentioned previously.

Specifically, the three goals of my thesis project are the following:

1. Determine specifications for a training set of materials, where each material is distinguished by its unique reflectance properties.

2. Design and implement a system capable of segmenting a test object into distinct, classifiable material regions.

3. Use the training set to perform per-region material classification.

The third goal will be discussed in the theoretical sense, and will be tested on real data in the future.

1.1 Overview

In this paper, we will thoroughly describe the specific steps of our material classification system. Chapter 2 includes background information and previous work related to the ideas in the project. Chapter 3 summarizes our overall approach to building the classification system. Chapter 4 and Chapter 5 describe the preliminary research we did, before working with 3D data. The ideas established in these two chapters apply in a straight-forward manner to the 3D case, and are therefore worth discussing. Chapters 6 and 7 describe how we represent reflectance and how we fit data to our reflectance model. These procedures aim to create unique identifiers for materials and to provide a framework for classification. Chapter 8 describes the process of point cloud segmentation. Chapter 9 describes the final classification step, which is the step that all our previous ideas have been building toward. Finally, Chapter 10 discusses how these ideas can be applied to a working experimental system in the future.

Chapter 2

Background and Previous Work

There is a great amount of research accomplished in the fields of image segmentation, BRDF data collection, and reflectance models. In this chapter, we will discuss some of the work and its relevance to our particular project goals.

2.1 Various Image Segmentation Algorithms

One of the first steps in our material classification system is to segment our data into regions that are likely to be of the same material. Grouping related pixels together can greatly decrease the number of components that must be considered in subsequent processing steps. If we use similarity in color as a heuristic, this task becomes an image segmentation problem.

2.1.1 Mean-shift Segmentation

Mean-shift segmentation [3] is a popular clustering-based algorithm. Most of the computation is done in the $(x, y, f(x, y))$ domain, where $f(x, y)$ is the pixel value at location (x, y) in the image segmentation case. The segmentation procedure can be described, on a high level, as the following steps:

1. Randomly select a region of interest in the $(x, y, f(x, y))$ domain, of a predetermined size.

2. Compute the centroid of the data belonging to this region.

3. Shift the region so that its center is at the location of the centroid.

4. Repeat Steps 1 - 3 until the desired amount of segmentation is achieved.

This procedure will naturally cause neighboring pixels of similar color to cluster together. However, the mean-shift segmentation method is computationally intensive. Iteratively computing the centroid, or the local maximum of density over a particular region, requires much processing time. Therefore, for this project, we have chosen to use Felzenszwalb's graph-based segmentation method, as described in the next section.

2.1.2 Segmentation by Felzenszwalb and Huttenlocher

The image segmentation algorithm presented by Felzenszwalb and Huttenlocher [4] is widely used for its efficiency and simplicity. The graph-based method begins by representing each pixel in the image as a single-entity component. Our initial edge weights are equal to the color difference, or variability, between adjacent pixels (Figure 2-1). At each step, the variability between every two adjacent components is compared to each component's inner variability in RGB space. If the inter-component variability is lower, the two components will merge as one. Through this iterative merging process, the number of components decreases while the average component size increases. In the end, the different colors and/or textures of the image are segmented appropriately into different components. A preference for smaller or larger components can be adjusted through the parameter k. A larger value of k indicates a preference for creating larger components.

A good quality of Felzenszwalb's algorithm is that it follows a global merging criterion. The decision to merge is not based on local gradients across pixels, but rather on gradients across components. Each component possesses some amount of inner variability, and this amount is compared to the variability between this component and a second, adjacent component. To understand this, consider the

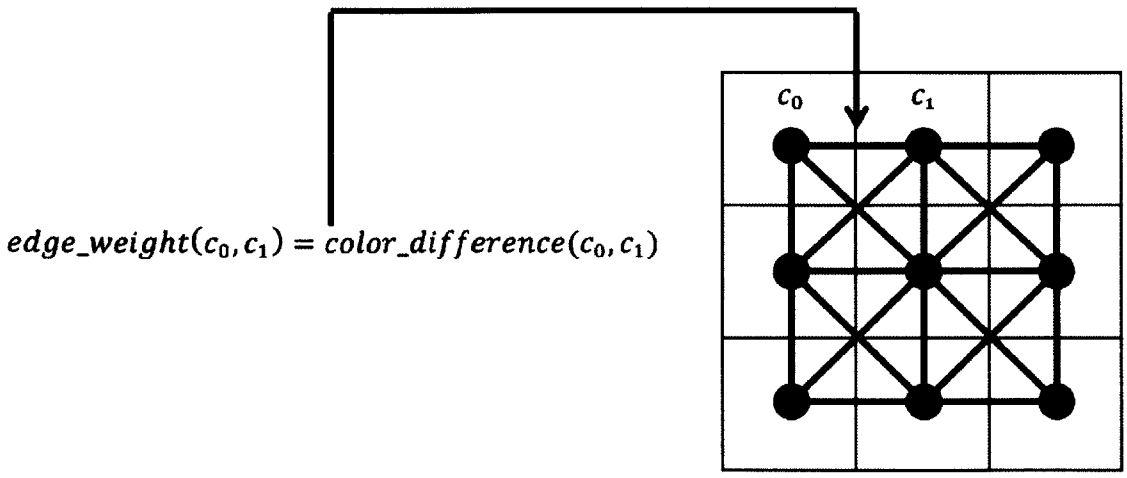

Figure 2-1: The initial edge weight for c_0 and c_1 in single-image Felzenszwalb segmentation.

synthetic image in Figure 2-2(a). The bottom half of this synthetic image contains a gradient from white to black. One might expect that naive segmentation algorithms would keep the gradient area divided into different components. However, as shown in Figure 2-2(b), Felzenszwalb's algorithm merges the entire gradient area into a single component. For the common case, this is the result that we want. With its global properties, Felzenszwalb's algorithm can achieve this result, because the variability within the white-to-black gradient is very gradual and evenly spread. For the same reason, the gradient area is not merged with the white area in the synthetic image. The variability between the two areas is larger than the variability within the gradient area, so they are determined to be two distinct components.

Figure 2-3 shows the segmentation of a real captured image. The randomly generated colors in the result clearly mark the individual regions. It is evident that the left side of the sneaker has been incorrectly merged with the background. While it is true that the input image should have been more thoroughly lighted, this example represents the common case where a single image does not contain all the information necessary to detect all edges. To tackle this problem, our approach uses multiple input images for the process. As demonstrated in Chapter 4, this increases the robustness of the segmentation.

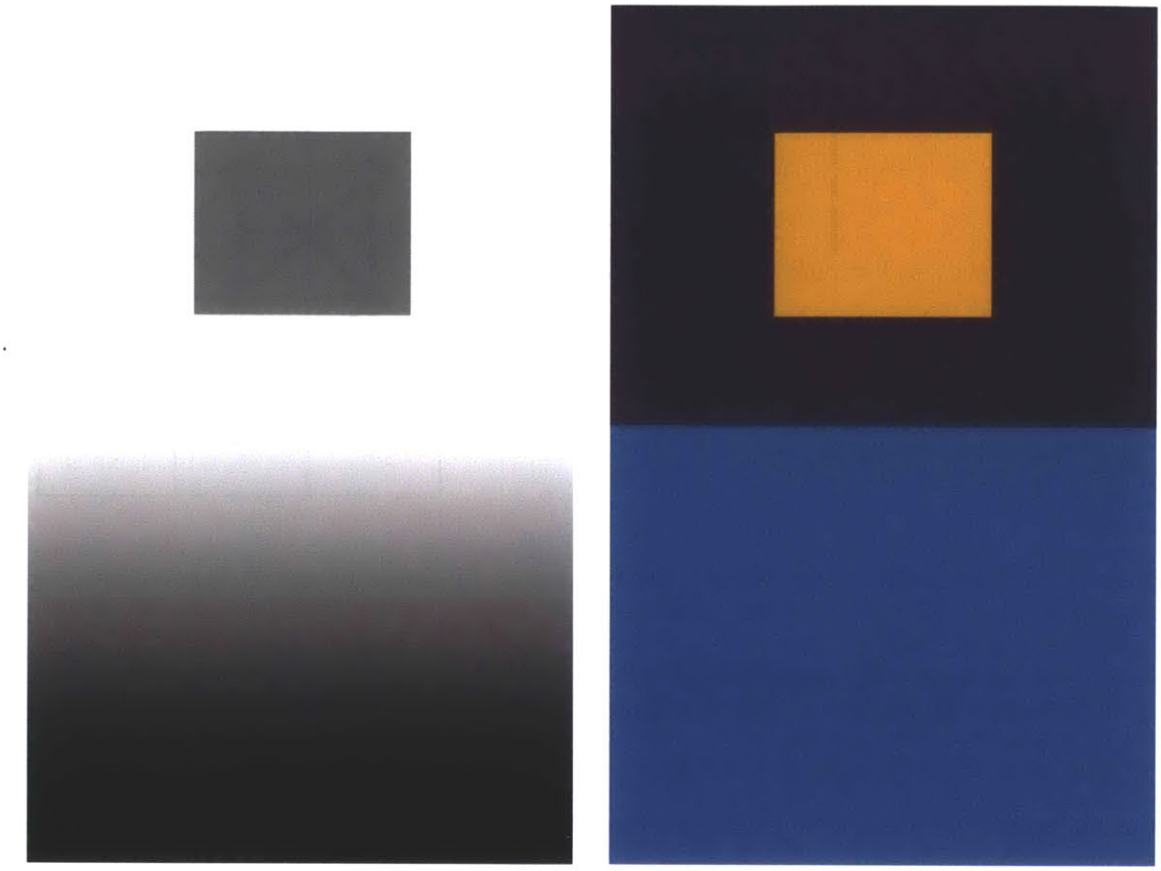

Figure 2-2: (a) A synthetic image whose top half contains a gray box against a white background and whose bottom half contains a white-to-black gradient. (b) The Felzenszwalb segmentation of the synthetic image. The three resulting components are shown in different colors.

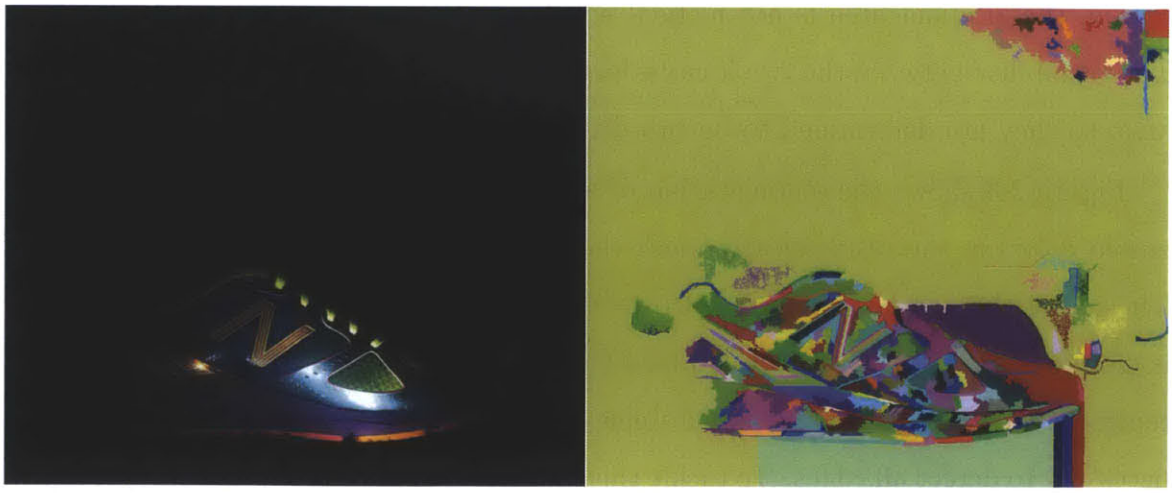

Figure 2-3: Felzenszwalb segmentation of a colorful shoe. Left: Input image. Right: Segmentation result for $k = 10$.

2.2 BRDF Representation and Data Collection

In this project, we will distinguish between materials by their unique reflectance properties. Therefore, we need ways to collect this reflectance data, as well as to present this data in a simple, convenient form. This section describes some related previous work.

2.2.1 Extraction of Shape and BRDFs by Goldman et al.

Goldman et al. [5] developed a method that retrieves both shape and per-pixel BRDF samples from a set of photographs taken from the same camera angle but under different illuminating angles. The BRDF is a quantitative measure of a material's reflectance properties, and will be further discussed later in this paper. The method by Goldman et al. is based on the assumption that many real world surfaces consist of relatively few distinct materials, which they call *fundamental materials*. Then, every surface can be modeled as a weighted mixture of such materials. This produces a simple reflectance model, which also allows for a simpler shape retrieval process that requires only a small amount of scanning.

While Goldman et al. introduce great ideas regarding BRDF retrieval, their reflectance model involves several complications. For example, user input is required to provide specific weight constraints for each fundamental material. Additionally, there are cases where a surface can be described by more than one linear combination of fundamental materials. In these cases, it is difficult to uniquely identify the surface. Finally, because accurate shape is retrievable due to the recent advances in 3D scanning, we would prefer a method that focuses solely on the BRDF aspect.

2.2.2 A Data-Driven Reflectance Model by Matusik et al.

Matusik et al. [8] develops a reflectance model that relies on a minimal number of assumptions and is heavily defined by real data. After gathering 100 spheres of different materials, a series of images of each sphere is captured, with a stationary camera and an orbiting light source. Then, each pixel of the sphere will serve as

a separate BRDF sample for the corresponding material. This process produces a dataset that contains the densely sampled BRDF for each of the 100 materials.

This dataset is called the MERL BRDF database, and appears to be the most complete BRDF dataset available. Unfortunately, it does not cover many materials that we would like to represent and detect. Additionally, we prefer a reflectance model that can both represent new training materials and provide a framework for classifying test materials, without the need to densely sample the BRDF.

One such reflectance model is Lafortune et al.'s cosine lobe model [1], which we have implemented for this project. Chapter 6 describes this model in full detail.

Chapter 3

Approach

I was given the wonderful opportunity to do my thesis work with Google Research. With the help from my team and supervisor, I developed a three-step approach to building a material classification and segmentation system. We will briefly describe each step in this chapter, and expand with further detail in the later chapters.

3.1 Data Collection

3.1.1 The Material Coverage in the Training Set

Our training set contains various fabrics, including polyester, wool, leather, cotton, faux suede, and velvet. We acquired this set of fabrics from Room&Board and `www.fabric.com`, both of which provided samples or swatches at little or no cost.

In addition to fabrics, we also collected various tile samples from Home Depot and `www.cooltiles.com`. These samples include glass, slate, vinyl, sandstone, metal, wood, and ceramic.

3.1.2 The Representation of Materials

We capture approximately 50 images of each material sample, covering eight viewing angles and six lighting angles. From these images, we can generate a 3D point cloud that is complete with surface normal information.

Given the known world locations of our camera and lights, it is then possible to compute the local lighting angles and viewing angle at each point in the point cloud. In addition, our 50 images contain color information for each point, observed under certain sets of viewing and lighting conditions. This is a sufficient amount of information for us to fit into a reflectance model. In particular, we are using the cosine lobe model, which allows us to represent a material's reflective properties with only thirteen numerical values. These thirteen values can serve as the material's unique signature, or feature vector, in our training set. We will go into more detail in Chapter 6.

3.1.3 Point Cloud Generation

Point cloud generation is not one of the steps that are emphasized in this work, as I was not directly involved with its development. However, it is still a very important component of the project, because point clouds are a major part of the data we collect. Furthermore, many ideas were developed under the assumption that a point cloud of the object could be gathered. Therefore, I will briefly describe the procedure for point cloud generation.

As described by Scharstein et al. [10], 3D surface acquisition can be achieved by a system consisting of one projector and one or more cameras. All devices are angled toward the object of interest, with overlapping fields of view. Then, a series of gray code patterns [9] (Figure 3-1) are projected onto the object, providing a unique encoding of the projected image column and row. The cameras observe which gray code occurs at each pixel. For a given gray code value, the pixel location will vary with the camera viewpoint, allowing stereo correspondences to be established. In this way, a calibrated projector and one or more cameras can be used to estimate depth via triangulation.

To further improve accuracy, a repeating "phase" pattern can be used in combination with gray codes. Scharstein et al. used a sequence of sinusoidal patterns, while Gühring [7] demonstrated a high-accuracy method that uses hard-edged stripes with sub-pixel center detection.

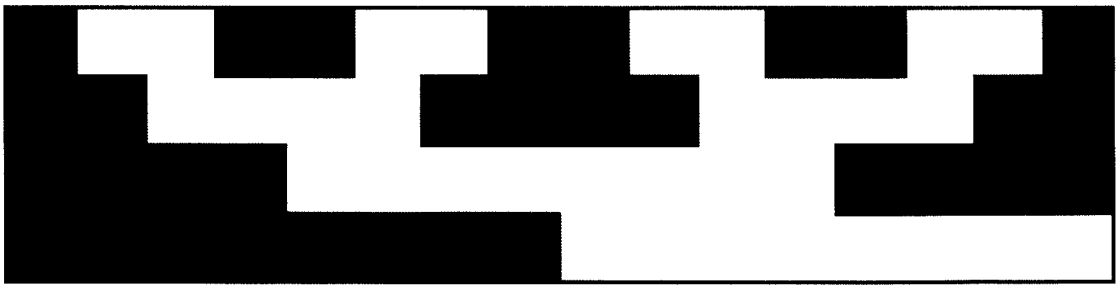

Figure 3-1: The gray code pattern. Each column contains a unique code. This structured light can be projected onto an object to determine correspondences between two cameras.

From the inter-camera correspondences, it is possible to compute the 3D points of the object's surface, which populate the desired point cloud. Neighboring points can then be combined to form surface mesh patches.

3.2 A Color-Based 3D Segmentation

Now that we have a training set, we need to develop a procedure for processing an unknown test object and classifying its material content. Note that in the common case, different parts of a test object can be composed of different materials. In order to classify these materials, we can follow a bottom-up approach, a top-down approach, or a combination of the two.

The bottom-up approach can involve per-point material classification. After classifying each point in the point cloud as a material, we can then smooth the material regions by eliminating obvious discontinuities. However, the problem in this approach is that there is simply not enough information in a single point to make an accurate judgment about its material content. The little information there is will also be quite susceptible to noise.

Therefore, we apply a top-down approach, where we first determine a segmentation of the point cloud into regions that *likely* contain the same material. Per-region material classification will be much more reliable, as there is much more data to work with.

We identify these regions by performing color-based segmentation. The main reasoning behind using color information is that if a continuous region of points share a common color and/or texture, this region is likely to be made of the same material.

We will use a modification of Felzenszwalb's segmentation algorithm that takes in multiple color observations per point. In Chapter 4, we demonstrate the effectiveness of this modified algorithm in the 2D case where the inputs are multiple images taken of the object under a constant viewing angle and varying lighting angles. Then, in Chapter 8, we will apply the same ideas to the 3D case, where we must segment a point cloud.

3.3 Per-Region Material Classification

The final step is to classify each segmented region as a particular material. It is useful to interpret each region as a subset of points in the point cloud. We will fit each region to the cosine lobe model, similar to how we gathered data for our training set. The fitting process will again produce thirteen parameter values, which will ideally uniquely represent the material of the region. We can then perform a nearest-neighbor classification – we can go through all the known material entries in our training set, and locate the material that possesses the most similar parameter values. Of course, the confidence value of the classification is dependent on the number of points in the region, because a greater amount of (good) data leads to greater accuracy. The confidence value is also dependent on the amount of distance between the parameters of the test region and those of the nearest neighbor.

Chapter 4

Image Stack Segmentation

4.1 Introduction

As mentioned in Section 2.1.2, using a single image of the object for segmentation is usually insufficient for an accurate separation of visibly different regions. We extend Felzenszwalb's algorithm by modifying it to accept multiple input images. We call this modified approach *image stack segmentation.* If the input images are appropriately selected, they will provide significantly more information regarding the true edges of the object. In this chapter, we will experiment with different methods that utilize this extra information to effectively improve segmentation results.

4.2 Setup

Our setup for image stack segmentation involves (1) a turntable, (2) a stationary camera pointed toward the center of the table, (3) a small tray suspended above the center of the table, and (4) two light sources affixed to the edge of the table. One light source is positioned below the tray and is angled at approximately 30 degrees above the horizontal. The other is positioned above the tray, and is angled at 30 degrees below.

Figure 4-1: A subset of the image stack collected for the sneaker. Only the angle of illumination is varied for each image.

4.3 Image Acquisition

We aim to expose all true edges of the object, by capturing multiple images under varying lighting angles. We begin by positioning an object on the tray and turning on one of the light sources. Images are captured by the stationary camera as the turntable spins and the light source consequently rotates about the object. The same procedure is done for the other light source. Around 50 images are captured per rotation per light. Figure 4-1 shows a subset of the image stack collected for the sneaker.

It is helpful to discuss the properties of this image stack. First, both top and bottom light sources are used in order to fully expose the edges. Second, each image in the stack is mostly dark, but contains great detail in a certain portion of the object. Such images are achievable by using point lights. Assuming that we have ways to distinguish dark areas from the object foreground and that we have enough images per rotation, it is possible to produce a very accurate segmentation of the object.

4.4 Combining the Information - Weight Metrics

Because we are now using multiple input images, we need a way to combine all the information and use it effectively in our segmentation process. How do we calculate the initial edge weights if they depend on RGB differences that vary from image to

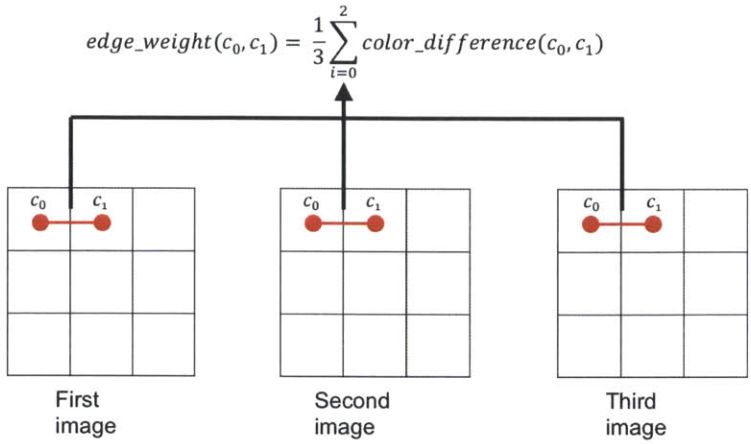

Figure 4-2: The initial edge weight for c_0 and c_1 in image stack segmentation, where the mean metric is used and the stack size is three.

image?

We experimented with three different distance metrics for determining the initial edge weights. Figure 4-2 shows one possible metric that utilizes the mean. Letting c_0 and c_1 be two adjacent pixel locations, our initial edge weight is equal to the average color distance between c_0 and c_1, calculated over the image stack. Of course, other possible distance metrics include those that utilize the median or the maximum.

4.5 Other Adjustments – Edge Pixel Skipping and Filtering

For our initial edge weight calculations, we provide an option to ignore pixels that are very close to 0 or 255. Since we are using point light sources, our input images can contain many dark areas that have little information. In these areas, the true colors are not accurately detected and the inter-component RGB distances are not reliable. Therefore, skipping over these dark areas will provide more accurate weights for the segmentation algorithm. It is a similar case for the pixels close to 255 – we want to ignore these pixels, because they are likely to have been overly saturated by the light sources, and do not reflect the true colors of the object.

Lastly, we experimented with bilateral filtering. This is an image filter that blurs

areas of low contrast but preserves areas of high contrast. Consequently, it can be used to dampen the tiny variations within the same texture while maintaining the true edges of the image. We have considered applying bilateral filtering on our input images as a pre-processing step, as an attempt to eliminate the tiny details that could potentially confuse the segmentation. Although it is a little effective, the improvement is not worth the extra computation time. We have decided that our segmentation results can be refined through other methods.

4.6 Results For Image Stack Segmentation

Results are shown in Figure 4-3. The segmentation uses approximately 100 images, one of which is shown in the first column. The second and third columns show the same segmentation in different colors. In order to generate the colors for Column 2, we go through each region from the segmentation, and retrieve the locations of pixels that belong to the region. We then look up the RGB values of these pixels in one of the input images, and calculate the average color for the region. Column 2 displays the regions in their respective average colors. Column 3, on the other hand, displays them in random colors.

The two visualizations are useful in different ways. The average-color visualization brings out any sign of over-merging. Over-merging refers to the case when areas of visibly different colors are incorrectly merged as one region during the segmentation. Over-merged regions are easily detected because their average colors do not reflect the true colors in the input image.

The random-color visualization is better for under-merging detection. Under-merging occurs when adjacent areas of the same color/texture remain as separate regions throughout the segmentation procedure. Because the region boundaries in the random-color visualization are very defined, it is easy to observe under-merging when comparing against a raw input image.

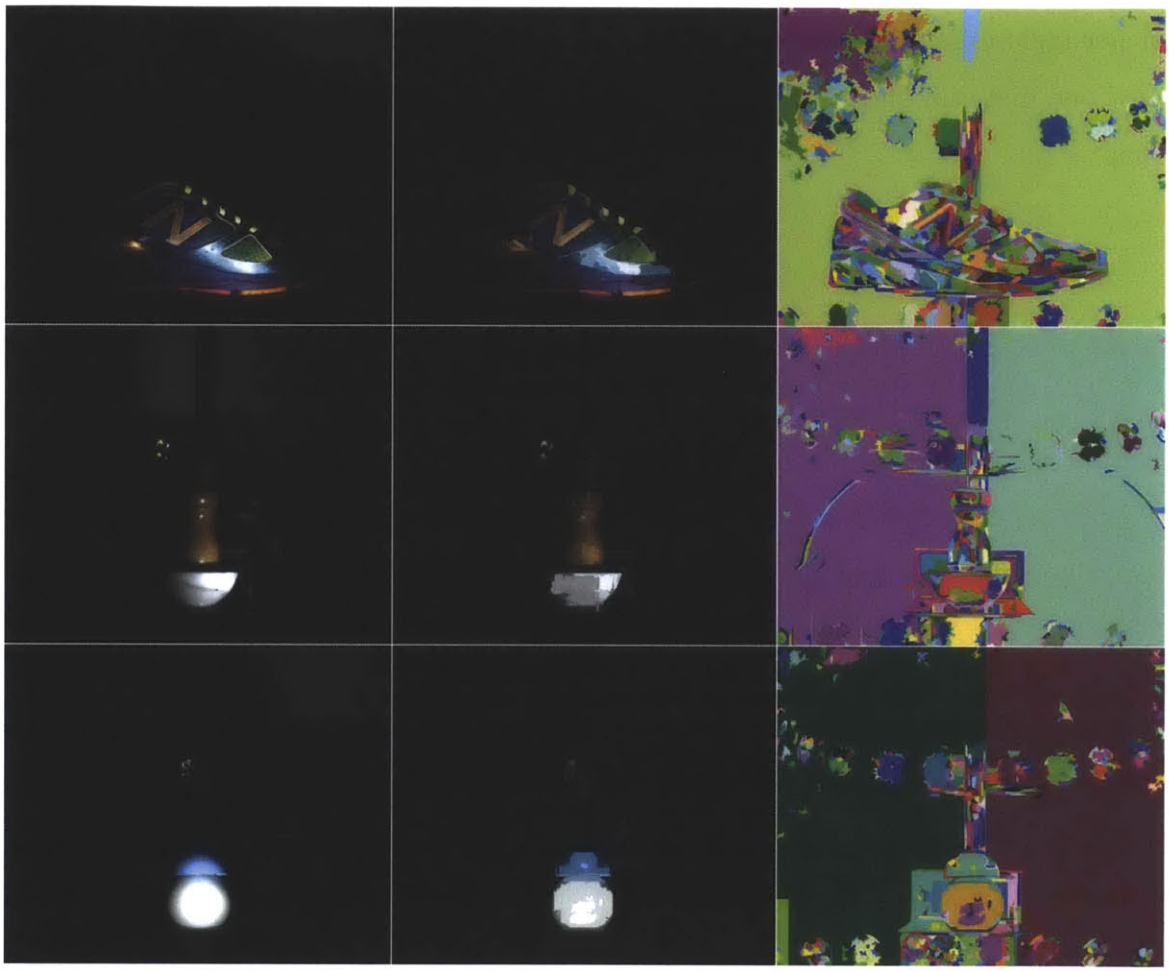

Figure 4-3: Results of image stack segmentation. The rows, from top to bottom, show the segmentations of (1) a sneaker, (2) a wooden pepper shaker on top of a ceramic bowl, and (3) a rubber hemisphere on top of a matte sphere. The columns, from left to right, show (1) one of the input images, (2) segmentation result where regions are in their average colors, and (3) the same segmentation where regions are randomly colored to clearly show boundaries.

4.6.1 Segmentation Issues

One issue is the occurrence of under-merging on both the background and the object. However, it is better to under-merge than to over-merge, because the latter case causes a loss of detail. We must keep in mind that the segmentation is a preparatory step for per-region material classification. We want to ensure that each region contains *at most* one material and texture. That said, we will tackle the under-merging issue in Chapter 5.

Chapter 5

HAC For Further Region Merging

5.1 Introduction

We introduce another procedure that targets the under-merging issue discussed previously. We apply hierarchical agglomerative clustering (HAC) to further merge similar regions together.

HAC is a bottom-up approach for building a hierarchy of clusters. In our case, each segmented region starts as its own unique cluster. Similar to the segmentation algorithm, the goal is to merge clusters by following a particular distance metric. One common metric is Euclidean distance. For example, if we are clustering 2D points and we have three points {(0, 0), (0, 1), (3, 3)}, the first two points will have a smaller Euclidean distance and are therefore more likely to merge than the other pairs of points. The decision to merge depends on the value of the similarity threshold.

With C representing our set of clusters, the process involves the following steps:

1. Go through the pairs of clusters in C, and locate p, the pair with the smallest distance.

2. If $dist(p) < t$, where t is the similarity threshold, merge the two clusters of p into a new, single cluster c_{new}. More specifically, create a new cluster c_{new}, and record a mapping from this cluster to the IDs of the two original clusters.

3. Remove the two original clusters from C, and add c_{new} to C.

4. Repeat steps 1 - 3 until all pairs have distances greater than t.

There are different clustering modes, each with a unique decision rule for merging. *Complete linkage mode* merges clusters c_0 and c_1 based on the maximum distance found between two inner clusters, such that one is contained in c_0 and the other is contained in c_1. This mode provides the strictest form of clustering, because a single pair of high dissimilarity will prevent the two clusters from merging, regardless of how similar the other pairs are. This strictness can be beneficial for our case, where over-merging should be strongly avoided. *Average group linkage mode* merges clusters based on the average distance calculated, and is a more balanced form of clustering. Finally, *centroid mode* uses not the information on the inner clusters of c_0 and c_1, but rather the information on c_0 and c_1 themselves. Specifically, centroid mode works under the assumption that every newly merged cluster is represented in weighted terms of its original two clusters. For example, if clusters are represented as numerical vectors, the merged representative vector can be a weighted average of the original two.

5.2 The Region Clustering Implementation

5.2.1 Inputs For Clustering

As mentioned before, each of our initial clusters will represent a region that has been determined by our segmentation process. The representation will be in the form of a color intensity strip. By the term *color intensity strip*, we are referring to a region's collection of average intensities for the image stack. This is shown in Figure 5-1, where each row corresponds to a different region's intensity strip. A region's average intensity is calculated for each image in the image stack. A typical intensity strip is mostly dark, except for one or two peaks in color. These peaks correspond to the images where the region is most exposed to the light source during the turntable spin.

Along with the intensity strips, the clustering algorithm also takes in region sizes. The sizes influence the cluster weights, as explained in the next sections. Average

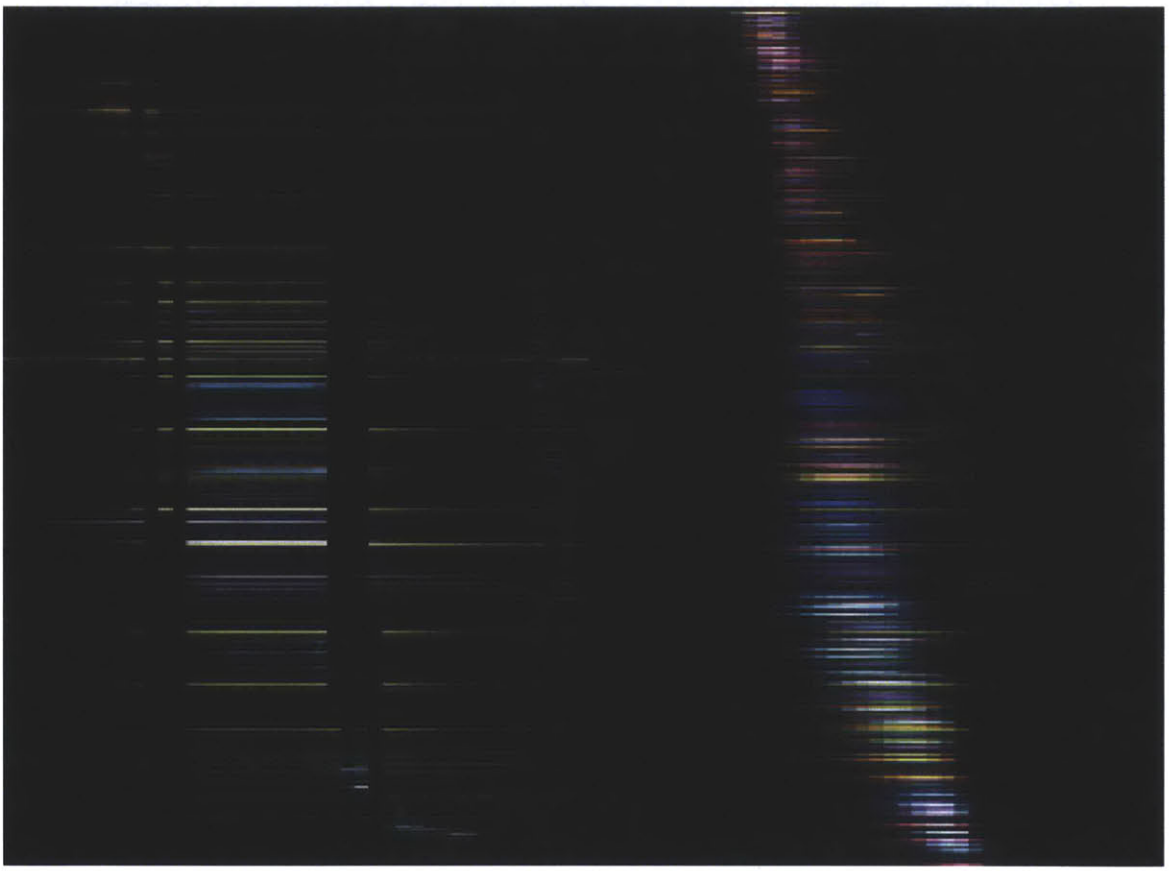

Figure 5-1: A plot of region intensities across the image stack/sequence for the sneaker. The rows map to different segmented regions, while the columns map to different frames of the light spin. The peaks of color for each region correspond to the frames where the region is most directly exposed to light. The left peak is caused by the higher light source and the right peak is caused by the lower light source.

Figure 5-2: The mean image for the sneaker. The color for each pixel in the mean image is an average over all images in the image stack. Region colors are extracted from the mean image and used for visualizing clustering results.

region colors are the third input, and are gathered for visualization purposes. The difference between these average colors and the average colors from the intensity strip is that the former ones are based on the mean image. The mean image consists of pixels averaged across the entire image stack (Figure 5-2).

5.2.2 Cluster Representation

Each initial cluster stores a region's intensity strip in the form of an RGB vector. The length of the vector is consistent, and corresponds to the number of images in the image stack.

5.2.3 Merged Cluster Representation

When two clusters c_0 and c_1 merge, how is the merging represented? In our implementation, the newly merged cluster will have a feature vector equal to the weighted average of the feature vectors of c_0 and c_1. The weighting will be based on their region sizes, as shown in (5.1). The expressions s_0 and s_1 are the region sizes of c_0 and c_1, respectively.

$$fv_{new} = \frac{s_0}{s_0 + s_1} fv_0 + \frac{s_1}{s_0 + s_1} fv_1 \tag{5.1}$$

5.2.4 Distance Metrics

We provide two distance metrics for computing similarity.

Sum of squared differences Given two clusters, we look up their vector representations and calculate their squared RGB distance:

$$ssd(fv_0, fv_1) = \sum_{i=0}^{n} (fv_0[i] - fv_1[i])^2 \tag{5.2}$$

Expression (5.2) consists of a summation of individual differences for every image index. We now calculate the mean image difference, which will be the distance metric.

$$dist(fv_0, fv_1) = \sqrt{\frac{1}{n} SSD(fv_0, fv_1)} \tag{5.3}$$

Cosine similarity Treating clusters as high-dimensional vectors, cosine similarity is the measure of the cosine of the angle between them. A high cosine value indicates a high level of similarity between the clusters. The distance metric is then the expression in (5.5).

$$sim(fv_0, fv_1) = \frac{fv_0 fv_1}{\|fv_0\| \|fv_1\|} \tag{5.4}$$

$$dist(fv_0, fv_1) = 1 - sim(fv_0, fv_1) \tag{5.5}$$

Figure 5-3: A plot of the same region intensities, now organized. Similar regions are now clustered together.

5.2.5 Region Clustering Results

Figure 5-3 shows the same region intensity plot as Figure 5-1, but in a different arrangement. This arrangement reflects a post-clustering order, where regions ending up in the same cluster are displayed together. As a result, the colors in this plot appear more organized than the scattered colors in the previous plot.

The right side of Figure 5-4 is a distance matrix. For any row i and column j, the intensity at (i, j) reflects the RGB distance between regions i and j. A higher intensity indicates a greater distance. Along the diagonal are squares that represent the formed clusters. A cluster's size, i.e. the number of regions it contains, is equal to the width and height of its square. A cluster's collection of region colors is also shown within the square. This visualization provides a good tool for evaluating the clustering accuracy. Incorrect clustering is detected when a square contains inconsistent colors.

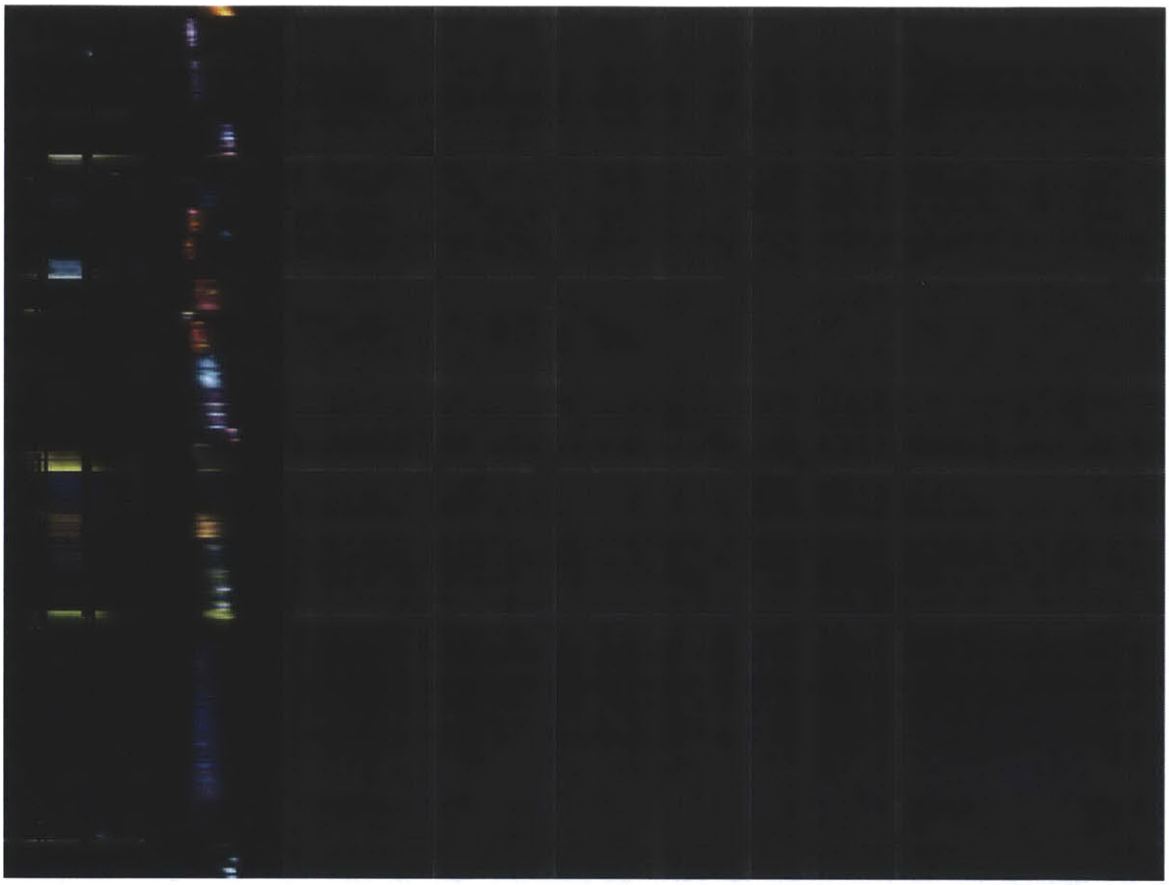

Figure 5-4: The plot on the left is identical to the one in Figure 5-3. It is shown again here in order to provide a visual mapping from region to cluster. On the right is a distance matrix whose diagonal squares represent the formed clusters, in both size and average color. One can see the region-to-cluster correlation by scanning any row across these two images. This clustering is the result of using cosine similarity in average group linkage mode, with a similarity threshold of 0.85.

Figure 5-5: A clustering result for the wooden salt shaker and ceramic bowl, using the sum-of-squared-differences metric in average group linkage mode with $t = 0.87$.

Figure 5-6: Another clustering result for the salt shaker and ceramic bowl, now using the SSD metric in centroid mode. This is also $t = 0.87$. Observe that centroid mode produces less consistent clusters, such as the big brown/gray cluster shown in the right half of the distance matrix.

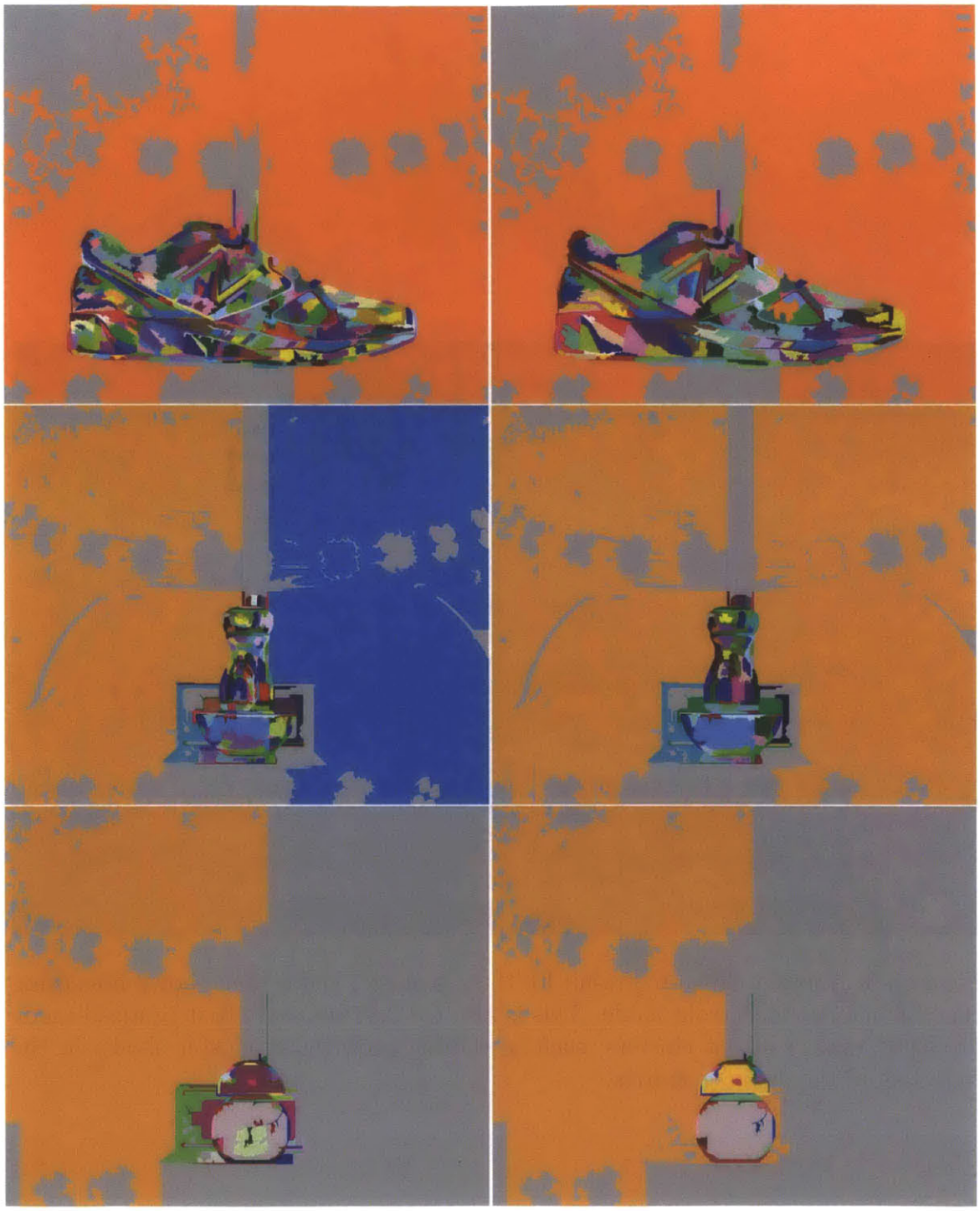

Figure 5-7: A direct comparison between pre-clustering and post-clustering segmentations. Parameters are $k = 10$, $t = 0.90$, and complete linkage mode. The pre-clustering images are the same results from Figure 4-3, but object masks have been applied to remove some of the background regions (now in gray). For the sneaker, the number of regions has reduced from 320 to 127. For the pepper shaker and ceramic bowl, it has reduced from 152 to 66. For the spherical objects, 65 to 27. We have removed almost two-thirds of the regions, with no significant sign of over-merging.

Chapter 6

Representing Reflectance

The reflectance properties of a material can be thoroughly represented by the material's bidirectional reflectance distribution function, or *BRDF*. This function describes how a light is reflected off an opaque surface, given an incoming light direction and an outgoing light direction. Specifically, for a point on the surface, it is the ratio of the reflected light intensity to the incident light intensity.

By densely sampling a range of incoming and outgoing light angles and calculating their corresponding BRDFs, we can build an accurate reflectance model for the material. Rendering programs, such as POV-Ray or Mitsuba Renderer, can then use this model to synthetically render images of the material. The only inputs needed for the rendering process are (1) the positions of the viewer and the light source, (2) the intensity of the light source, and (3) the diffuse color of the material.

However, collecting the full BRDF of a material is very time-consuming, as we would have to densely sample a whole hemisphere's worth of incident and exitant angles. There are currently a few BRDF data sets available to the public, but they cover only a small range of materials. In this paper, we explore a simplified reflectance model that relies on the approximation of a few nonlinear parameters rather than on dense sampling. This simplified model represents the materials fairly accurately, with the additional bonus that it allows us to have very simple training data for future classification.

6.1 A Simplified Reflectance Model

6.1.1 Assumptions

In order to produce a simplified reflectance model, we have made a few assumptions regarding reflectance.

1. Our model assumes isotropic reflection. For the purposes of this paper, isotropic reflection refers to the property that the intensity of light reflected remains unchanged as the exitant angle rotates about the incident angle, or vice versa.

2. For the diffuse component, the intensity of light reflected is dependent only on the $\hat{z}$ (up) components of the exitant angle and of the incident angle. In other words, the diffuse color observed by a stationary viewer will remain the same even as the light source rotates about the observed point's surface normal.

It is important to note that our use of the term *isotropic reflection* is different from some other papers. While our usage is consistent with Lafortune et al. [1], the term has also been used by Matusik et al. [8] as the property where the intensity of reflected light remains unchanged as the exitant angle rotates about the surface normal. Assumption (1) implies classes of symmetry that include those implied by Matusik et al., and is therefore a stronger statement.

6.1.2 Model Specifications

Our model consists of cosine lobes, which are essentially cosine functions raised to a high power:

$$f_r(\mathbf{u}, \mathbf{v}) = \rho_s C_s \cos^n \alpha \tag{6.1}$$

In this representation, $f_r(\mathbf{u}, \mathbf{v})$ is the approximated BRDF for incident angle $\mathbf{u}$ and exitant angle $\mathbf{v}$. The term ρ_s expresses the maximum albedo, or the reflecting power of the lobe, and is between 0 and 1 if the normalization factor C_s is assigned to be $\frac{n+1}{2\pi}$. The term ρ_s and the exponent n determine the size and shape of the lobe.

The function in (6.1) can also be represented as a dot product of $\mathbf{u}_m$ and $\mathbf{v}$, where $\mathbf{u}_m$ is the mirrored incident angle. Then, by applying singular value decomposition, we develop the expression in (6.3). To learn more about this derivation, please see the work by LaFortune et al. [1].

$$f_r(\mathbf{u}, \mathbf{v}) = \rho_s C_s [\mathbf{u}_m \cdot \mathbf{v}]^n \tag{6.2}$$

$$f_r(\mathbf{u}, \mathbf{v}) = \rho_s [C_x u_x v_x + C_y u_y v_y + C_z u_z v_z]^n \tag{6.3}$$

As Lafortune et al. also describe, the BRDF can be sufficiently represented as a combination of three or more cosine lobes. In their paper, they use three lobes to represent the specular component of the observed color, and one lobe to represent the diffuse component. We will follow this setup.

The full expression for our model is shown in (6.6). The BRDF approximations, $f_{r,specular}(\mathbf{u}, \mathbf{v})$ and $f_{r,diffuse}(\mathbf{u}, \mathbf{v})$, act as weights for the specular component and the diffuse component, respectively. The $f_{r,specular}(\mathbf{u}, \mathbf{v})$ term is a summation of three cosine lobes, each in the form of (6.3). The $f_{r,diffuse}(\mathbf{u}, \mathbf{v})$ term contains a scalar cosine function, involving only values in the $\hat{z}$ direction. The ρ terms have been absorbed into the other parameters.

$$f_{r,specular}(\mathbf{u}, \mathbf{v}) = \sum_i [C_{x,i} u_x v_x + C_{y,i} u_y v_y + C_{z,i} u_z v_z]^{n_{specular,i}} \tag{6.4}$$

$$f_{r,diffuse}(\mathbf{u}, \mathbf{v}) = C_d (u_z v_z)^{n_{diff}} \tag{6.5}$$

$$color_{observed} = \frac{f_{r,specular}(\mathbf{u}, \mathbf{v}) color_{specular} + f_{r,diffuse}(\mathbf{u}, \mathbf{v}) color_{diffuse}}{f_{r.specular}(\mathbf{u}, \mathbf{v}) + f_{r,diffuse}(\mathbf{u}, \mathbf{v})} \tag{6.6}$$

Certain constraints can be made on these terms in order to enforce our assumptions from Section 6.1.1. First, applying the constraint $C_x = C_y$ on the $f_{r,specular}(\mathbf{u}, \mathbf{v})$

Known Parameters	Unknown Parameters
$\mathbf{u}$	$C_{x,y,i}$ for all $i \in \{1,2,3\}$
$\mathbf{v}$	C_z for all $i \in \{1,2,3\}$
$color_{observed}$	$n_{specular,i}$ for all $i \in \{1,2,3\}$
	$color_{specular}$
	C_d
	$n_{diffuse}$
	$color_{diffuse}$

Table 6.1: The known and unknown parameters of our reflectance model.

term causes each component in the summation to be symmetric in x and y. Additionally, we know that each component is a function of the cosine of the angle between the incident and exitant angles. Combining these two conditions, we can conclude that $f_{r,specular}(\mathbf{u}, \mathbf{v})$ remains unchanged as the exitant angle rotates about the incident angle, or vice versa. This is equivalent to assumption (1). Furthermore, assumption (2) is reflected in the $f_{r,diffuse}(\mathbf{u}, \mathbf{v})$ term. The expression for $f_{r,diffuse}(\mathbf{u}, \mathbf{v})$ is achieved by applying the constraint $C_x = C_y = 0$, so that only values in the $\hat{z}$ direction matter. Since the coordinate axes are aligned locally to the point's surface normal, tangent, and bitangent, we can further conclude that $f_{r,diffuse}(\mathbf{u}, \mathbf{v})$ remains unchanged as the light source rotates about the point's surface normal. In other words, we conclude that assumption (2) holds.

6.1.3 The Known Parameters of the Model

It is important to state explicitly which parameters are known and which parameters are unknown. This information is given in Table 6.1. Of course, we know the colors that are observed at each point. In addition, we can assume that we know $\mathbf{u}$ and $\mathbf{v}$, which are the local incident and exitant light angles at each point on our object. These vectors can be retrieved from our data collection – our point cloud contains the world coordinates of each point, and we also have the physical coordinates of our camera and light sources.

We will now describe the method in which we transform our light vectors from the world coordinate system to the local coordinate system for each point in our point

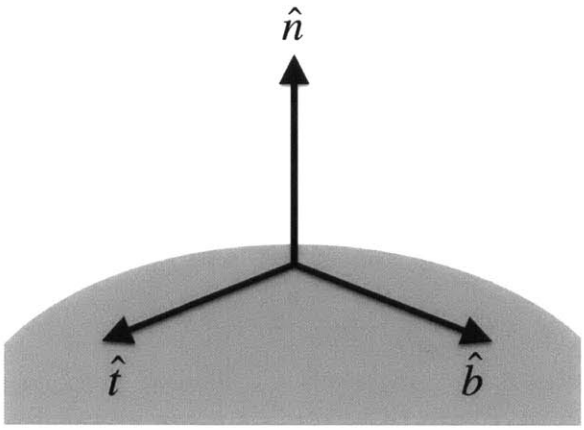

Figure 6-1: The local coordinate system. Its axes are aligned to the normal, tangent, and bitangent vectors.

cloud.

Notation Let $\hat{w}$ be the unit vector that indicates the incident light direction. Let $\hat{n}$ be the point's surface normal, and $\hat{e}$ be along an adjacent edge in our triangle mesh. From knowing these three vectors, which are expressed in world coordinates, we will compute $\hat{v}$, the incident light direction defined in a local coordinate system centered at the point. This local coordinate system will have its $\hat{x}$, $\hat{y}$, $\hat{z}$ axes aligned to the bitangent $\hat{b}$, tangent $\hat{t}$, and $\hat{n}$, respectively (Figure 6-1). These vectors are computed through the following steps:

1. Compute the cross product $\hat{n} \times \hat{e}$. This will be the bitangent $\hat{b}$, a unit vector orthogonal to the surface normal.

2. Compute the cross product $\hat{b} \times \hat{n}$. This will be the tangent $\hat{t}$, a unit vector orthogonal to $\hat{b}$ and $\hat{n}$. The tangent $\hat{t}$ can be viewed as the "adjusted" $\hat{e}$ that resulted from enforcing orthogonality between $\hat{e}$ and $\hat{n}$.

Then, our expression for $\hat{v}$ will be

$$\begin{bmatrix} \hat{v}_x \\ \hat{v}_y \\ \hat{v}_z \end{bmatrix} = \begin{bmatrix} \hat{t}_x & \hat{t}_y & \hat{t}_z \\ \hat{b}_x & \hat{b}_y & \hat{b}_z \\ \hat{n}_x & \hat{n}_y & \hat{n}_z \end{bmatrix} \begin{bmatrix} \hat{w}_x \\ \hat{w}_y \\ \hat{w}_z \end{bmatrix} \tag{6.7}$$

Chapter 7

Fitting Data to a Reflectance Model

7.1 Data Format – Protocol Buffers

A major goal of this project is to design a way to fit captured data to the reflectance model defined in Chapter 6. Our captured data will be in the form of a multi-layered protocol buffer [6]. Figure 7-1 shows the layout of our protocol buffer design. It contains all the 3D points in the point cloud, as well as the interpolated triangle mesh. Although we only use the points for this project, the triangle mesh is also included for potential future use. In addition to these components, we also have our collection of color observations. Each color observation is an observation of a particular 3D point, under a particular set of local camera and light angles.

Looking at the diagram in Figure 7-1, we see that the color observations are first separated by camera view, i.e. turntable orientation. The reason for this division is that for each camera view, there is a single set of 3D point to image coordinate mappings. This set of mappings does not change for differing light positions. If we had instead divided the observations based on light position, then the same set of mappings would be stored in multiple areas of the protocol buffer. Therefore, the camera-view-based division results in the smallest amount of redundancy.

Each camera view further expands into four components. The first component

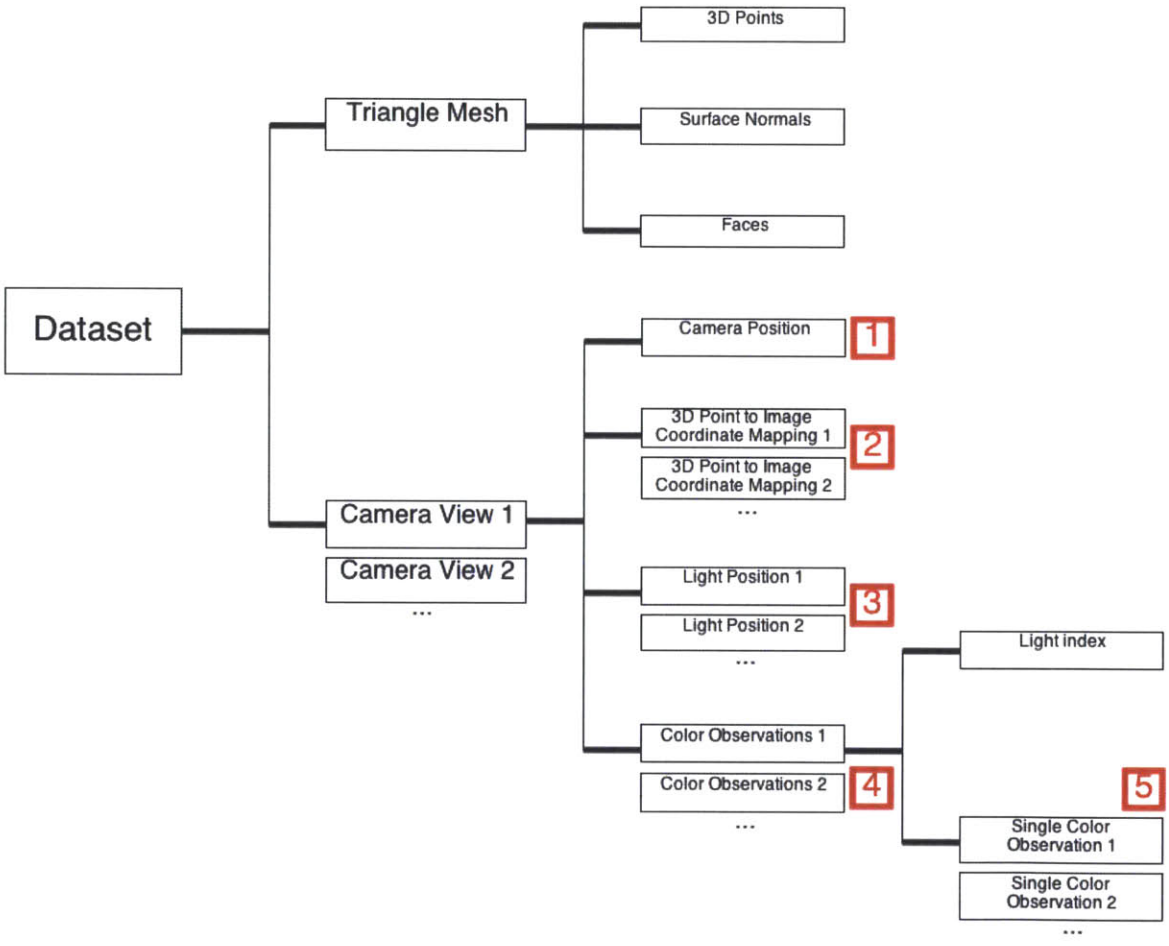

Figure 7-1: The layers of our protocol buffer message.

is the camera position, which is the position of the camera adjusted according to the current turntable orientation. The second component is the list of 3D point to image coordinate mappings, for all 3D points that have appeared in this particular camera view. The third component is the list of light positions. These light positions are also adjusted according to the current turntable orientation. Finally, the fourth component stores all the color observations. It is first divided up by light position, so that this first level has the same number of divisions as the third component. In other words, the areas labeled *3* and *4* in Figure 7-1 are parallel fields that contain the same number of divisions. Then, for each light position, there is a list of the raw color observations, corresponding to colors observed for specific 3D points and specific pairs of camera and light positions. This list, labeled *5* in Figure 7-1, is a parallel field to area *2*. Namely, the first color observation listed in area *5* would be for the first 3D point listed in area *2*, and so on.

The fitting process should be effective because we have enough information – there are several color observations for each point in our point cloud, each corresponding to a different pair of camera and light positions. From the camera and light positions, we can calculate the local incident and exitant angles, which are the $\hat{u}$ and $\hat{v}$ vectors in the reflectance model. This calculation is described in Section 6.1.3. Therefore, the information included in the protocol buffer message is enough to determine all the known parameters of the reflectance model. The next step is then to approximate the values of the *unknown* parameters of the model, through nonlinear optimization.

7.2 Nonlinear Approximation

We can use the Ceres Solver [2] to approximate the unknown parameters that are listed in Table 6.1. The Ceres Solver is open source software that provides a nonlinear least squares minimizer. In our specific case, we aim to compute the parameter values that minimize the sum of squares difference between the given data and the reflectance model.

7.2.1 Inputs for the Ceres Solver

Our input to the Ceres Solver will be the protocol buffer message described earlier in this chapter. The code implementation will parse through this protocol buffer and create a list of observations, each complete with its observed color, local incident angle, and local exitant angle. Each such observation counts as one *residual block*, or one equation in our nonlinear system of equations.

7.2.2 Outputs of the Ceres Solver

The outputs of the Ceres Solver will be the optimal fitted values of our unknown parameters. These results can also be stored as a protocol buffer (Figure 7-2). Therefore, our training set will consist of a list of such protocol buffers. Each of these protocol buffers will represent a certain material's reflectance properties, and will ultimately provide a unique signature for the material.

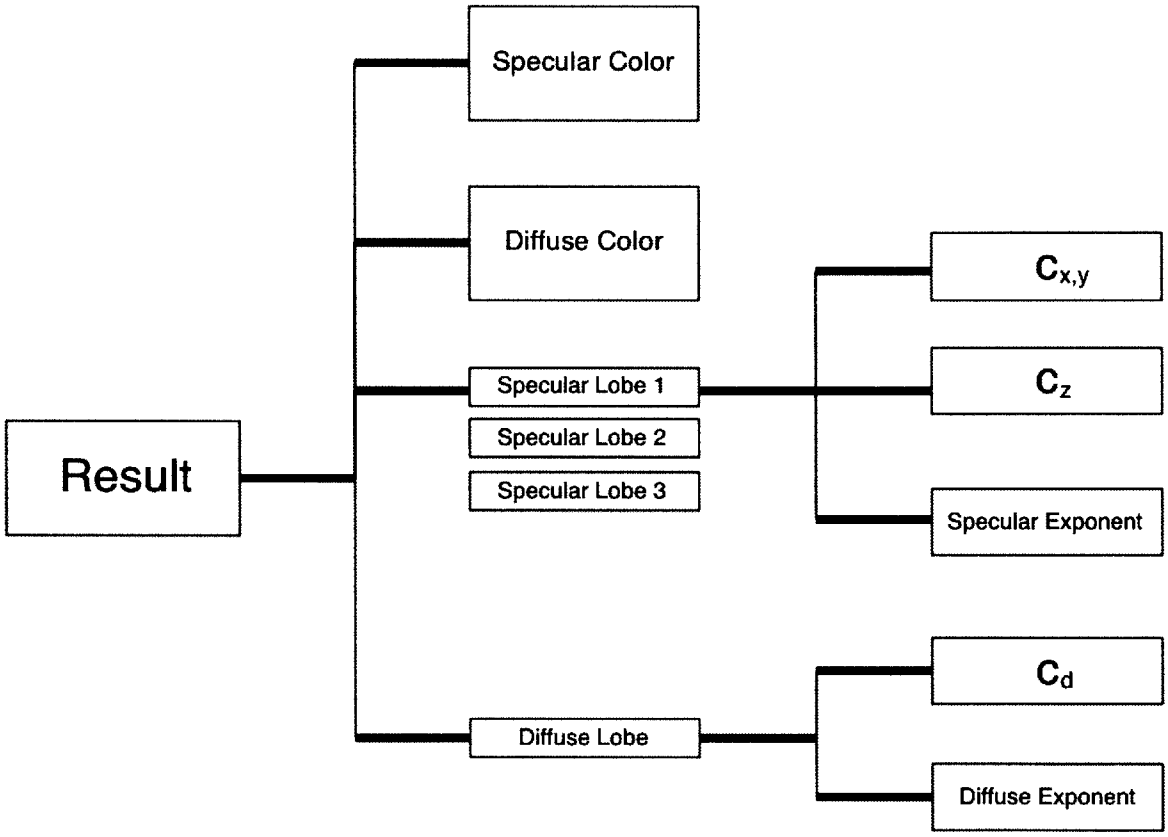

Figure 7-2: The protocol buffer message that stores fitting results. This is also the structure of entries in the training set.

Chapter 8

Point Cloud Segmentation

A typical object consists of multiple material regions, each of which we aim to classify. Since our fitting process works on one material at a time, we need a way to distinguish between these regions prior to data fitting. As mentioned earlier, one method is to perform color-based segmentation, as regions of similar color are likely to be of the same material.

In Chapter 4, we discussed the novel idea of *image stack segmentation*, which takes in multiple observations of an object and combines them to produce an accurate segmentation. This idea is applied to the first of two methods discussed in the next section. As for the second method, there is a simple mapping from the 2D case to the 3D case – pixel locations are instead 3D points in the point cloud, and adjacent pixels are instead vertices of adjacent triangles in the triangular mesh. Therefore, segmentation can be performed in a standard fashion. For both methods, the HAC process for region clustering (Chapter 5) can be applied for further refinement. However, there are still new complications that we need to consider for the point cloud case.

8.1 Various Methods

Unlike the light spin sequence where the only varying parameter was the light angle, we now also have to consider the varying turntable angles. Figures 8-1, 8-2, and 8-3

show texture images captured for all six lighting angles and all eight turntable angles, for a starfruit, an orange, and a robot-shaped toy, respectively. Each point in the point cloud appears for one or more turntable angles. Cases where the point would not appear would be when (1) the point's surface normal is more than ninety degrees apart from the camera angle and when (2) the point is occluded by another part of the object. For each turntable angle for which it does appear, its color is observed under six different light angles.

This six-element sequence qualifies as an image stack for image stack segmentation. Consequently, we can produce a different image segmentation result for each turntable angle for which this point has appeared. Figures 8-4, 8-5, and 8-6 show example segmentations for all turntable angles.

How do we then combine all these results to produce the global segmentation of the point cloud? Before we discuss some possible methods, it is helpful to clarify what information is available to us.

1. Each 3D point belongs to an average of three or four image segments. Each of these segments corresponds to a certain turntable angle.

2. Each image segment has a computable average color.

For further clarification, we call a segmented region of pixels an *image segment*, and a segmented region of 3D points a *point cloud segment*.

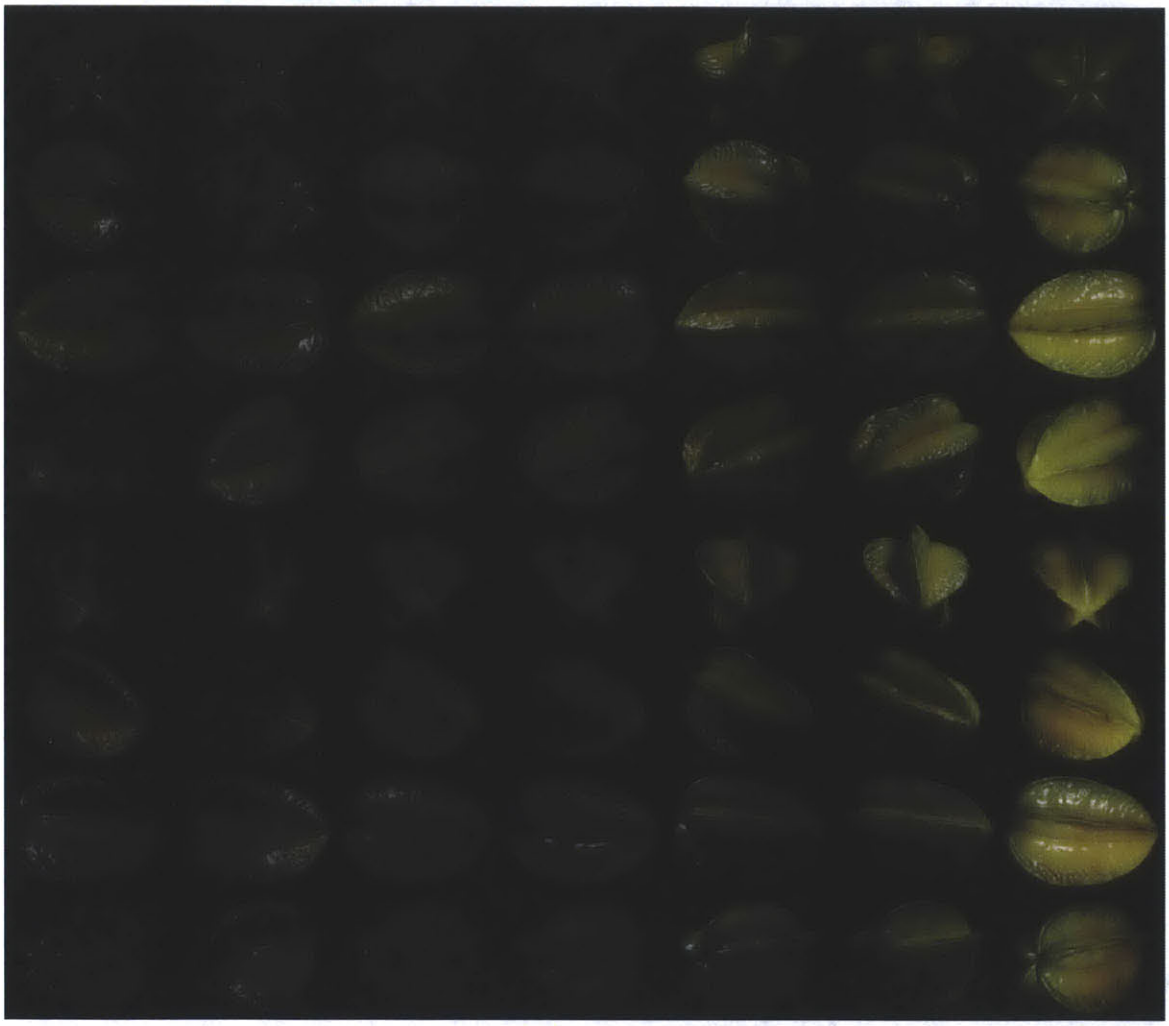

Figure 8-1: Texture images of a starfruit, covering all six lighting angles and all eight turntable angles.

Figure 8-2: Texture images of an orange, covering all six lighting angles and all eight turntable angles.

Figure 8-3: Texture images of a robot toy, covering all six lighting angles and all eight turntable angles.

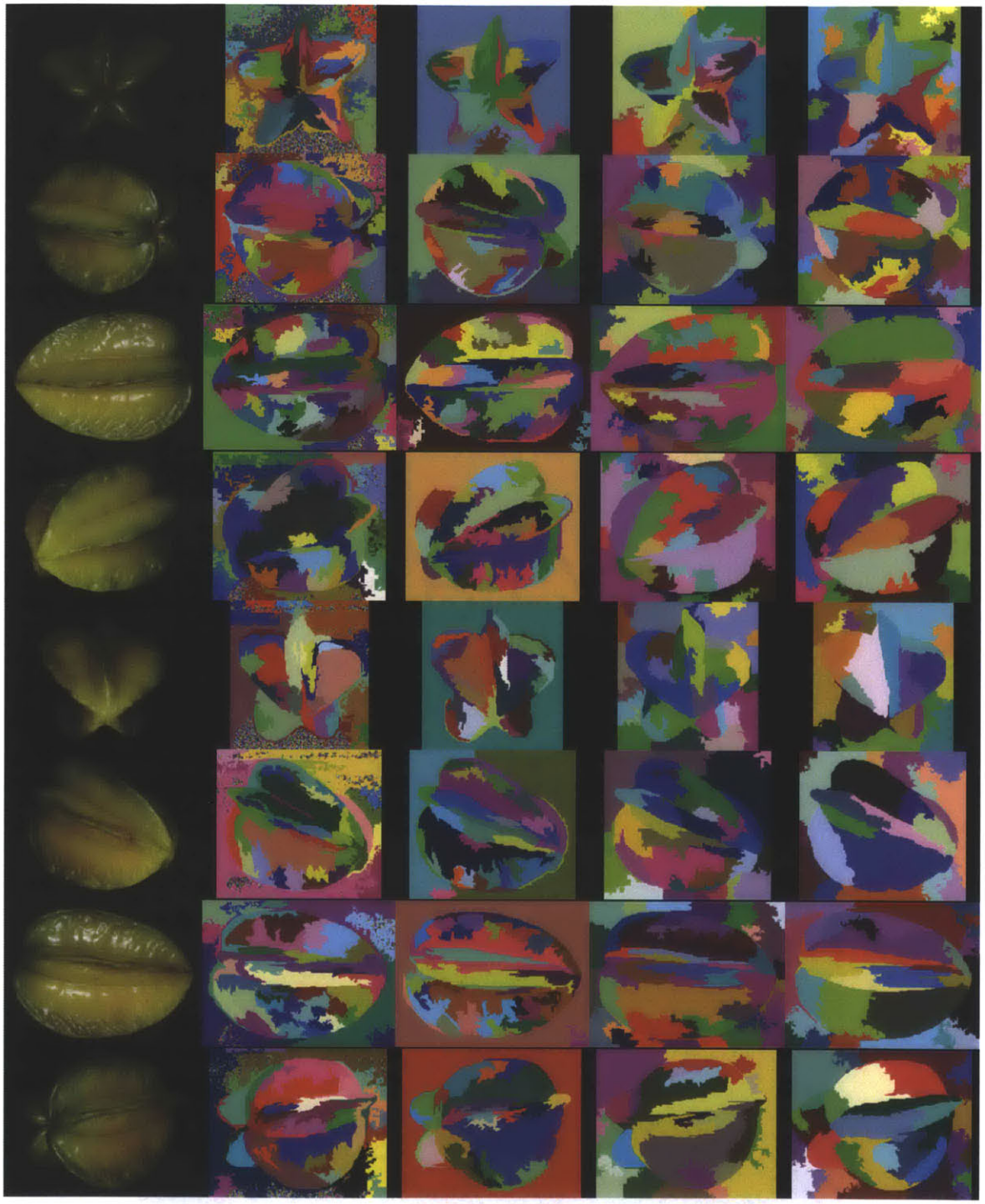

Figure 8-4: Segmentations for all orientations of a starfruit. First column: Original image. Second column: Single-image segmentation. Third, fourth, fifth columns: Image-stack segmentation with the mean metric, median metric, and max metric, respectively. Note that overall, image-stack segmentation detects true edges a little better than single image segmentation.

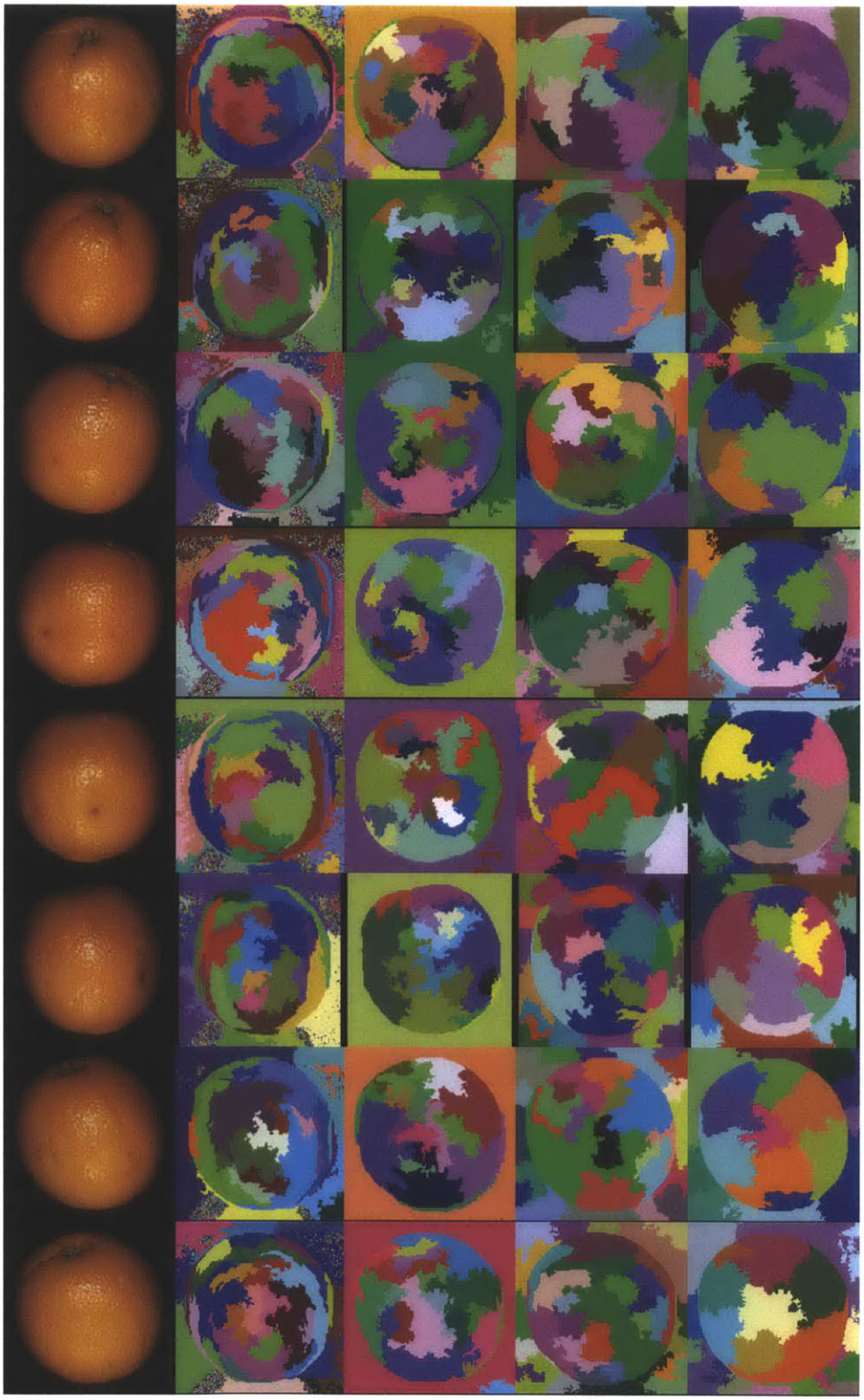

Figure 8-5: Segmentations for all orientations of an orange. First column: Original image. Second column: Single-image segmentation. Third, fourth, fifth columns: Image-stack segmentation with the mean metric, median metric, and max metric, respectively.

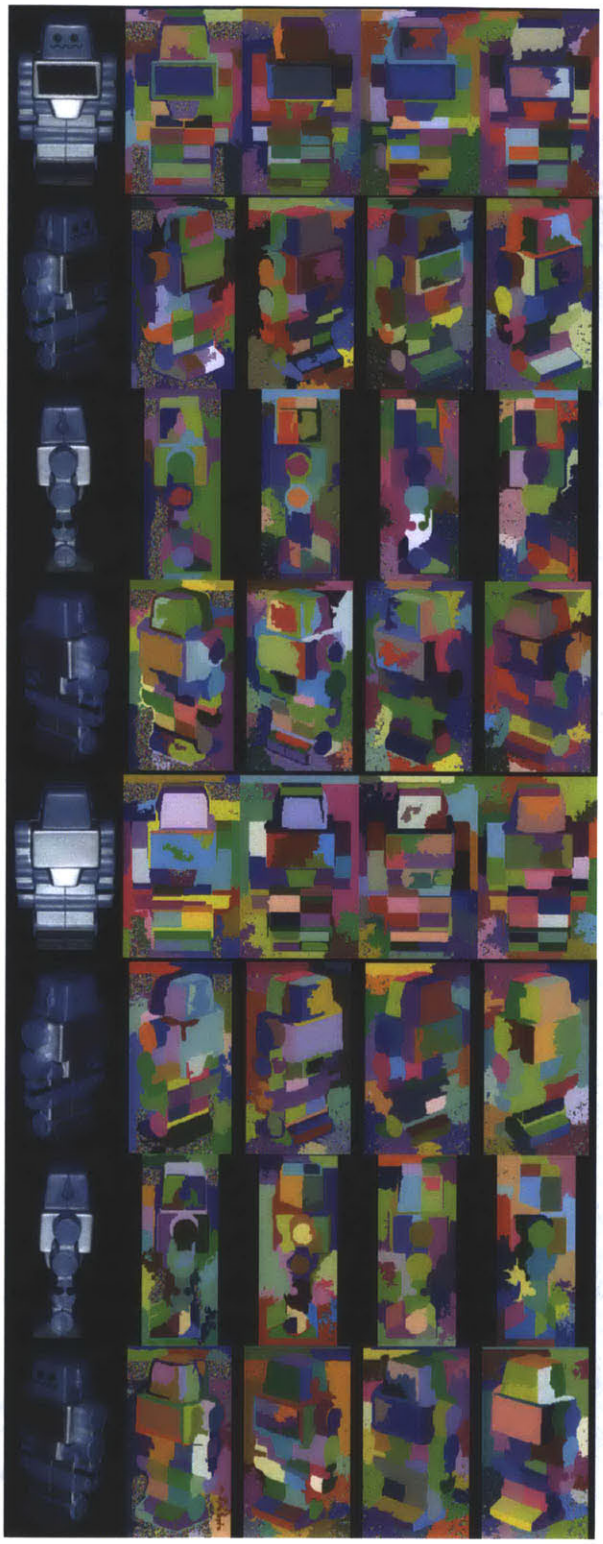

Figure 8-6: Segmentations for all orientations of a robot toy. First column: Original image. Second column: Single-image segmentation. Third, fourth, fifth columns: Image-stack segmentation with the mean metric, median metric, and max metric, respectively.

8.1.1 Method 1

Let P be a queue containing the points in the point cloud. We use the term R_i^p to refer to the ith image segment that point p belongs to. Additionally, the term $c(R_i^p)$ refers to the average region color of R_i^p. Then, Method 1 involves the following procedure:

1. Pop a point p off the queue P.
2. Set $\bigcup_{R^p}$ equal to R_1^p.
3. For $i \in \{2, ..., |R^p|\}$, set $\bigcup_{R^p}$ equal to $\bigcup_{R^p} \cup R_i^p$ if $|c(R_1^p) - c(R_i^p)| \leq t$, where t is an adjustable parameter that determines the strictness of our region merging.
4. Initialize set S to $\emptyset$. The set S will be a growing region of the point cloud.
5. For every 3D point r that lies in $\bigcup_{R^p}$:
   (a) Add r to S. Then, remove r from the queue P.
   (b) Remove r from all image segments in R^r. In other words, remove r from all the image segments it appears in.
6. Add S to our set of point cloud segments.
7. Repeat steps 1 - 5, until P becomes empty.

At the end of this procedure, every point will be assigned to exactly one point cloud segment, resulting in a full segmentation of the point cloud. Steps 3 - 5 use the theory that if any two points a and b share at least one image segment, it is likely that they should belong to the same point cloud segment. Furthermore, if two image segments have overlapping points and very similar average colors, their points should belong to the same point cloud segment.

As explained in Section 4.6.1, the over-merging of regions is worse than the under-merging of regions, since we need to be certain that we are not fitting data for more than one material at a time. In theory, Method 1 should not over-merge, as long as

the individual image segmentations had been strict. The subsequent merging of image segments in Step 3, which is based on average colors, should also not cause a problem under the assumption that a typical object does not contain a very complicated material composition.

8.1.2 Method 2

Method 2 is a simpler method that does not involve stack segmentation. Instead, it depends on mappings from the 2D case to the 3D case. Specifically, the vertices of our segmentation graph are now 3D points instead of pixels, and the edges of our graph are now sides of triangles in the triangular mesh. In addition to arranging these mappings, one new complication is the assignment of colors to our points.

As explained earlier, each point has appeared for one or more turntable angles. For each such turntable angle, it has appeared in six different colors corresponding to the six different light sources. What is a good way, then, to assign a single color to the point? One simple way would be to use the average color, or the color averaged over all the observations of the point. Then, we can apply the standard single-image Felzenszwalb segmentation method.

Method 2 can be more concretely defined as the following steps:

1. Let G be the graph that we will segment.
2. Set the vertices of G to points in the point cloud.
3. For each pair of vertices, add an edge between them if they map to points that share a triangle in the mesh.
4. For each vertex $v \in G$, compute $c(v)$, the color of v averaged over all color observations of v.
5. Run the standard Felzenszwalb segmentation on G, using differences in $c(v)$ as the distance metric.
6. Re-map the vertices back to their corresponding points, to finalize a full segmentation of the point cloud.

In theory, Method 2 would be less error-resilient than Method 1, as the color averaging in Step 4 simplifies our available data by a significant amount. We performed color averaging in Method 1 as well, but it was a region-based averaging. Because pixels that ended up in the same region were assumed to have similar colors already, averaging over these colors did not result in a big loss of information. Therefore, in comparison, Method 1 takes better advantage of the data that we have, at the expense of being more complex.

Chapter 9

The Classification Step

9.1 Procedure

We are finally prepared to classify test materials. The full procedure is as follows:

1. Perform image capture to produce the Dataset protocol buffer for the test object. This protocol buffer includes both point cloud data and color observations.

2. Perform point cloud segmentation to generate $\{S_1, S_2, ..., S_n\}$, where S_i is a region of the point cloud that is likely to be made of the same material.

3. For $i \in \{1, 2, ..., n\}$:

   (a) Feed S_i into the Ceres Solver. The Ceres Solver will fit the data to a reflectance model, and generate $\mathbf{t}$, the fitted parameter values.

   (b) Go through the training set, and determine the entry $\mathbf{g}$ that contains parameter values closest to $\mathbf{t}$.

   (c) Classify region S_i as material m, where m is the material that $\mathbf{g}$ represents.

At the end of this procedure, every point in the point cloud will be classified as some material. Of course, there are more considerations to be made regarding the accuracy of this classification.

9.2 Classification Accuracy

9.2.1 Region Size

The confidence value of the classification of a particular region is dependent on the number of points in the region. If there are only a few points in the region, there probably isn't enough data for the fitting process to produce an accurate reflectance model for the region. Assuming that the majority of the data is reliable, accuracy should increase as the amount of available data increases.

One way to avoid classifications of small regions is to detect the region sizes prior to classification. If the size is lower than a predetermined threshold, simply mark the region as *unknown*.

9.2.2 Distance in the Feature Vector Space

Another sign of classification inaccuracy is a large distance between the fitted parameter values of the test region and those of its nearest neighbor entry in the training set. When a large distance occurs, it is more than possible that the test material is simply not covered by the training set.

A similar problematic case is when two or more entries in the training set are equally close to a test region's fitted values. Which entry represents the true material? Finding a way to resolve such conflicts would be part of the future work for this project.

Chapter 10

Future Work

10.1 Fitting Real Data

If given the opportunity to work further on this project, a definite task would be to run our fitting process on the collection of various fabrics and tiles that we have gathered. We should then test our fitting process on real objects, and evaluate the system's effectiveness in material classification. We want empirical evidence that using the cosine lobe model as a reflectance model does allow us to produce unique and consistent signatures for materials.

We expect the discovery of several overlooked cases, simply because there are many factors that are correlated with classification accuracy. In addition to the cases mentioned in Section 9.2 regarding the implications of region sizes and distances, our data might not be reliable in the first place. For example, our color observations could contain too many shadows.

10.2 Applying HAC

We should also experiment with applying HAC (hierarchical agglomerative clustering) to the point cloud segmentation. As described in Chapter 5, performing additional region merging with HAC greatly decreases the number of segmented regions, without signs of over-merging. There are two advantages to having fewer regions – (1) fewer

runs of the fitting process are required and (2) each region now contains more data points, which increases the robustness of the fitting process. Therefore, in the ideal sense, applying HAC advances the classification in both efficiency and accuracy.

Chapter 11

Contributions

In conclusion, this paper described the development of a material segmentation and classification framework, which includes the following contributions:

1. Developed a robust procedure for multi-image and 3D segmentation.
2. Designed simple and compact data structures, in the form of a protocol buffer, to store raw observations and material feature vectors in the training set.
3. Implemented an optimization procedure that allows the BRDF to be approximated with only thirteen numerical values.
4. Developed a material classification procedure that encompasses all of the contributions listed above.

Bibliography

[1] *Non-Linear Approximation of Reflectance Functions*, 1997.

[2] Sameer Agarwal and Keir Mierle. *Ceres Solver: Tutorial & Reference.* Google Inc.

[3] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis Machine Intell.*, 24(5):603–619, 2002.

[4] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2), September 2004.

[5] Dan B. Goldman, Brian Curless, Aaron Hertzmann, and Steve Seitz. Shape and spatially-varying brdfs from photometric stereo. Technical report, University of Washington and University of Toronto, 2003.

[6] Google. Protocol buffers. `http://code.google.com/apis/protocolbuffers/`.

[7] Jens Gühring. Dense 3-d surface acquisition by structured light using off-the-shelf components. Technical report, University of Stuttgart, 2001.

[8] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, July 2003.

[9] Joaquim Salvi, Jordi Pagès, and Joan Batlle. Pattern codication strategies in structured light systems. *Pattern Recognition*, 37(4):827–849, 2004.

[10] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. *IEEE Computer Vision and Pattern Recognition*, 1, 2003.