Scalable High-Quality 3D Scanning

By
Karthik Sankaran Narayan

A Dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy
in
Computer Science
in the
Graduate Division
of the
University of California, Berkeley

Fall 2016

Abstract

Scalable High-Quality 3D Scanning

by

Karthik Sankaran Narayan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel & Professor Jitendra Malik, Co-chairs

Over the past decade, vendors across fields ranging from entertainment to retail to architecture have increasingly been shifting everyday consumable media from the 2D plane to the 3D world. Applications in these fields such as augmented/virtual reality, online shopping, and indoor scene reconstruction all have one crucial problem that needs to be addressed: the task of acquiring high-quality 3D models. Currently, designing novel 3D content involves a substantial human component; indeed, most 3D models were created via modeling software, e.g., 3DS Max, Maya, by a human with vast expertise. Although several research communities have explored automated 3D scanning over the course of several decades, the lack of an out-of-the-box solution has precluded this field from permeating the industry. Prior approaches to 3D scanning can largely be divided into image-based reconstruction (IBR) and active reconstruction (AR). While IBR methods produce reconstructions by taking in as input a collection of calibrated RGB images, AR methods actively project patterns onto observed scenes during reconstruction. IBR and AR techniques each have advantages and pitfalls. State of the art methods in 3D scanning typically fall purely within IBR or AR.

In this thesis, we outline a novel scanning approach which merges IBR and AR methods. First, we discuss how to physically construct and calibrate a 3D scanner using commodity hardware. We use this 3D scanner in the collection and release of the BigBIRD dataset, which serves as a key benchmark for the algorithms in this thesis and comprises of 600 12 MP images, 600 registered RGB-D point clouds, and several other processed data for each of 125 objects. Next, illustrating that the advantages and pitfalls within IBR and AR complement each other, we present a novel shape reconstruction algorithm that capitalizes on the strengths of each approach, yielding less than 2 mm of RMS error during reconstruction. We then present a color reconstruction algorithm, with which we produce high quality 3D color meshes of scanned objects; we demonstrate that this color reconstruction algorithm outperforms the state of the art. Finally, with any growing dataset, large-scale data visualization becomes increasingly important. Although the 3D scanning world is not yet at the large scale level, it is only a matter of time before it is. In addition, there already exists an eclectic array of fields, including computer vision, particle physics, and botany, where large-scale visualization techniques can benefit research. We present a novel and highly flexible GPU-based nonlinear dimensionality reduction technique capable of visualizing datasets with tens of millions instances and millions of features. Our algorithm's flexibility and speed yield an implementation that is an order of magnitude faster than state of the art implementations of stochastic neighbor embedding algorithms.

To Amma, Amma, Appa, Perimma, Gokul, and Bhuvana

Contents

Acknowledgments

Man, if someone told me four years ago that I was going to be writing this page that you're reading now, I'd have called them crazy. It's unfair to call this my dissertation, when so many people were a part of it. There really isn't enough room to mention everyone and everything.

Pieter, thank you so much for being such a wonderful advisor. Whether I was excited to discuss a research idea that was completely orthogonal to what I was working on, or whether I was stuck somewhere, Pieter was there to brainstorm. Without a shadow of a doubt, Pieter is the hardest working person that I've met – between teaching at Berkeley, advising his Ph.D. students, postdocs, and undergraduates, serving as cofounder of the next biggest education startup, Pieter always made time to talk about research, career plans, and life in general.

Throughout my academic career, I have always had excellent mentors. I'm highly thankful for the various research discussions I've had with Jitendra Malik, who really made me think of the bigger picture as opposed to drudging around in the pixels. Speaking with Marty Banks was always enlightening – I'd always come out of our chats learning something new: whether it was how recent advances in computer vision and human vision related, or whether it was discovering a new film to watch (the most recent being The Big Short). I could always walk into Alyosha Efros's office: he's a great mentor who was easily the most available. I learned a lot about computer graphics, art, and random hikes around Berkeley from Alyosha.

I'd like to thank the National Science Foundation for the NSF Graduate Fellowship as well as the American Society for Engineering Education for the NDSEG Fellowship. The generous stipends provided by these fellowships made it possible to focus on research and never have to worry about funding.

During the Ph.D. program, I was given the privilege to know and work with some of the most talented researchers that I've ever met: Pulkit Agrawal, Tudor Achim, Animesh Garg, Dylan Hadfield-Menell, Sandy Huang, Tim Hunter, Abhishek Kar, Weicheng Kuo, Alex Lee, Jeremy Maitin-Shepard, Teodor Moldovan, Zoe McCarthy, Ali Punjani, John Schulman, James Sha, Evan Shelhamer, Arjun Singh, Shubham Tulsiani, Ziang Xie, and many others.

I'd like to thank Pradeep Natarajan and Jon Graham, who encouraged me to chase numbers in research.

Charles Isbell, David Roberts, Mike Stilman, and Vijay Vazirani – thank you for introducing me to research.

Duncan, Abby, Brad, Marc, Catie, Melissa – thanks for the games of Avalon, Resistance, and delicious cookouts.

Sowmya, Rajesh – thanks for the intense games of Taboo, the incredible road trips, and our many more vacations to come.

Berk, Mikemac, Sailesh – thanks for being such fantastic friends who put up with my abhorrent bowling skills.

Gokul – thanks for being a great brother, for lightening up the mood, showing me different perspectives, and teaching me how to take life a little less seriously.

Jason – thanks for being my best man. Those years where we played games and watched random TV shows through the night were an era that I'll never forget. Here's to continuing that era through the years.

Bhuvana – thanks for being my best friend, showing me that life exists outside work, and for being a candle when times seemed really dark. Here's to the many years to come.

Mom, Dad – I couldn't have made it this far without you guys. Mom, thanks for listening to me talk about the bugs I squashed at work on my walks back home and my wacky, erratic new research ideas. Dad, thanks for showing me the art and craft of trading and introducing me to the world of finance. Thanks for everything.

1
Introduction

We used to live in a 3D world[*] whose inhabitants consumed content primarily in 2D. With innovations in computing and graphics, towards the turn of the 21st century, vendors in fields ranging from entertainment (augmented/virtual reality, movies, games) to retail (online shopping and advertising) to architecture (scene reconstruction) began shifting towards producing more 3D content. The task of acquiring novel 3D models lies at the intersection of these applications; specifically, improvements in 3D model acquisition speed and quality would have consequences in all of these fields.

It would be incorrect to claim that 3D models were never used in any field prior to the 21st century. Indeed, most every day products that we consume were initially a 3D model at one point. 3D modeling constitutes the bread and butter for most animation and graphics studios. However, in most cases, human 3D modelers with years of expertise typically create these models using specialized tools such as Autodesk's 3DS Max or Maya. Even with experience, manually creating high-quality 3D models involves a painstaking amount of work. Although research communities in computer vision have investigated the problem of automated 3D scanning over several decades, 3D modelers continue to manually construct objects due to the lack of an out-of-the-box solution.

A vast literature exists on the topic of 3D scanning, which can largely be divided into (1) image-based reconstruction (IBR) and (2) active reconstruction (AR). IBR techniques take in a collection of images as input and output either a point cloud or 3D mesh representing the object or scene to scan. Initially, researchers run a calibration procedure to orient the images, which are then used to perform the reconstruction task. Active reconstruction methods typically actively project a known pattern onto a scene, whose projection can be used to recover scene depth. Merging a collection of depth measurements recovers the object of scene. Laser scanners and structured light-based

[*]plus time and possibly many other dimensions, but we won't get into the physics of things

sensors fall within AR.

Unfortunately, the above approaches all have major pitfalls. Multiview stereo-based approaches rely on the presence of highly textured surfaces to recover depth; as such, non-textured and transparent/translucent regions pose a major problem. Visual hull-based techniques cannot recover surface concavities. Although researchers have explored hybrids of multiview stereo and visual hull-based methods, recovering concavities still poses a major problem in non-textured regions. Laser scanners and structured light-based sensors have major difficulties in recovering transparent/translucent regions as well as objects with significant amounts of black; in the former case, the projected light simply passes through the surface while in the latter case, the projected light is absorbed by the surface. However, perhaps the largest largest governing all approaches discussed so far involves camera calibration; without high-fidelity camera calibration, even the most sophisticated algorithms fail to recover high-resolution details.

In this thesis, we explore an approach that resolves many of the issues encountered by previous approaches. We first discuss how to construct a scanning rig, which we name the Big Berkeley Instance Recognition Dataset (BigBIRD) system, from commodity sensors (a small array of RGB DSLR cameras and Kinect sensors). Our high-fidelity calibration algorithm allows us to reconstruct objects with low RMS error (on the order of 1-2 mm). Our algorithms are able to reconstruct ultra high-resolution features such as barcode lines and dates of construction on bottles, features under 1 mm in size. As our reconstruction algorithm leverages both image-based and active reconstruction based techniques, we are able to recover concavities and transparent/translucent regions reliably well. Our scanning setup reliably scans objects that can fit within a cubic meter of space, although it can scan larger objects with more space. It cannot scan full scenes, however. This thesis is organized as follows:

Chapter 1. We detail the construction and calibration of the BigBIRD system[90]. Additionally, we present a high-quality, large-scale dataset of 3D object instances, with accurate calibration information for every image. We anticipate that "solving" this dataset will effectively remove many perception-related problems for mobile, sensing-based robots. The contributions of this work consist of: (1) BigBIRD, a dataset of 100 objects (and growing), composed of, for each object, 600 3D point clouds and 600 high-resolution (12 MP) images spanning all views, (2) a method for jointly calibrating a multi-camera system, (3) details of our data collection system, which collects all required data for a single object in under 6 minutes with minimal human effort, and (4) multiple software components, used to automate multi-sensor calibration and the data collection process. This work was published in ICRA 2014[90].

Chapter 2. We describe reconstruction algorithms to synthesize the data collected per object from the BigBIRD system into 3D models[76]. More generally, we consider the problem of building high-quality 3D object models from commodity RGB and depth sensors. Applications of such a database include instance and object recognition, robot grasping, virtual reality, graphics, and online shopping. Unfortunately, modern reconstruction approaches have difficulties in reconstructing objects with major transparencies (e.g., KinectFusion[80]) and/or concavities (e.g., visual hull). This paper presents a method to fuse visual hull information from off-the-shelf RGB cameras and KinectFusion cues from commodity depth sensors to produce models that are substantially better than either approach on its own. Extensive experiments on the recently published BigBIRD dataset[90] demonstrate that our reconstructions recover more accurate shape and detail than competing approaches, particularly on challenging objects with transparencies and/or concavities. Quantitative evaluations indicate that our approach consistently outperforms competing methods and achieves under 2 mm RMS error. This work was published in ICRA 2015[76].

Chapter 3. We describe an optimization-based vertex coloring algorithm to extract high-fidelity color maps for the 3D models constructed in Chapter 2[74]. Applications of a database of high-quality colored meshes include object recognition in robot vision, virtual reality, graphics, and online shopping. Most modern approaches that color a 3D object model from a collection of RGB images face problems in (1) producing realistic colors for non-Lambertian surfaces and (2) seamlessly integrating colors from multiple views. Our approach efficiently solves a non-linear least squares optimization problem to jointly estimate the RGB camera poses and color model. We discover that incorporating 2D texture cues, vertex color smoothing, and texture-adaptive camera viewpoint selection into the optimization problem produces qualitatively more coherent color models than those produced by competing methods. We further introduce practical strategies to accelerate optimization. We provide extensive empirical results on the BigBIRD dataset[76,90]: results from a user study with 133 participants indicate that on all 16 objects considered, our method outperforms competing approaches[74]. This work was published in IROS 2015[74].

Chapter 4. With any growing dataset, data visualization becomes increasingly important. Non-linear dimensionality reduction techniques have become particularly popular in data visualization. We discuss a novel data visualization algorithm that enables us to visualize up to tens of millions of data instances, an order of magnitude larger than competing algorithms. Although the 3D scanning world is not at this scale, it is only a matter of time before it is; furthermore, we will soon see that there exist several fields that would benefit from such large-scale visualization. Although recent work in non-linear dimensionality reduction investigates multiple choices of divergence measure during optimization[112,15], little work discusses the direct effects that divergence measures have on visualization. We investigate this relationship, theoretically and through an empirical analysis over 10 datasets. Our work shows how the α and β parameters of the generalized alpha-beta divergence can be chosen to discover hidden macro-structures (categories, e.g. birds) or micro-structures (fine-grained classes, e.g. toucans). Our method, which generalizes t-SNE[99], allows us to discover such structure without extensive grid searches over (α, β) due to our theoretical analysis: such structure is apparent with particular choices of (α, β) that generalize across datasets. We also discuss efficient parallel CPU and GPU schemes which are non-trivial due to the tree-structures employed in optimization and the large datasets that do not fully fit into GPU memory. Our method runs 20x faster than the fastest published code[103]. We conclude with detailed case studies on the following very large datasets: ILSVRC

2012, a standard computer vision dataset with 1.2M images; SUSY, a particle physics dataset with 5M instances; and HIGGS, another particle physics dataset with 11M instances. This represents the largest published visualization attained by SNE methods. This work was published in ICML 2015[75].

2

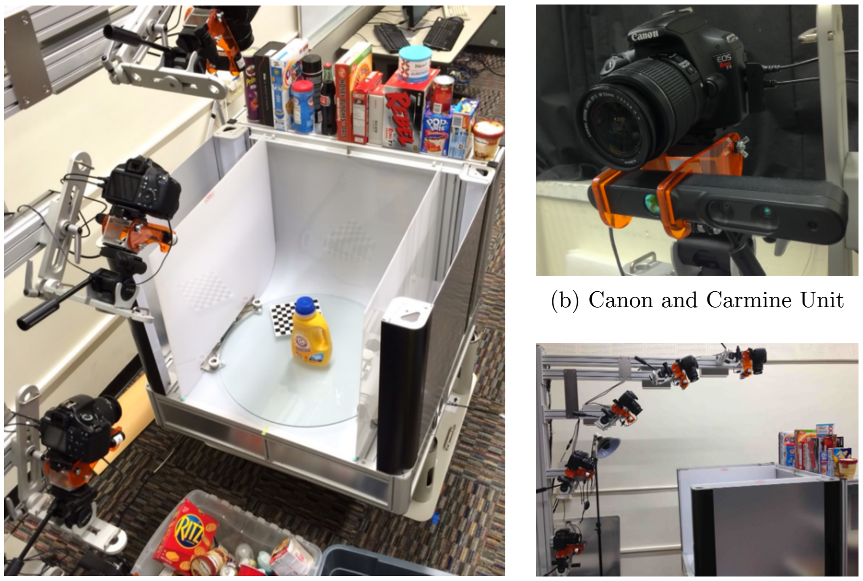
BigBIRD: A Large-Scale 3D Database of Object Instances

We first discuss the construction and calibration of the BigBIRD rig. We also discuss how we collect data from the rig. Further chapters detail algorithms to extract high-fidelity shape (Chapter 2) and color (Chapter 3) models from the collected data per object. This work was published in ICRA 2014[90].

2.1 Introduction and Related Work

Object recognition, the task of identifying a given object in an image, remains an unsolved problem in computer vision. Researchers typically dichotomize object recognition into (1) category-level recognition, where various concrete objects are assigned a single label (e.g. “Pepsi can” and “Coke can” are both assigned the label “soda can”) and (2) instance-recognition, where each concrete object is assigned a separate label (e.g. “Pepsi can” and “Coke can” are given separate labels). The computer vision and robotics approaches to the recognition problem differ fundamentally in that (1) robots in fixed environments typically need to interact with on the order of a few hundred objects and (2) robotic perception algorithms need to successfully localize and detect 3D object poses in addition to identifying the correct object. We believe that instance recognition suits many robotic tasks well, as joint object detection and pose estimation are the primary components of the instance recognition problem.

Despite the advent of commodity RGB-D sensors, which provide both depth and color channels, instance recognition systems still cannot reliably detect hundreds of objects[108,96,101]. We believe that the primary issue currently hampering progress towards reliable and robust instance recognition is the lack of a large-scale dataset containing high-quality 3D object data; this is because collecting such a dataset requires constructing a reliable and high-quality 3D scanning system, which is a significant undertaking.


(a) Ortery Photobench, Perspective

(b) Canon and Carmine Unit

(c) Ortery Photobench, Side

Figure 2.1: Our data-collection system. We place the object near the center of the turntable, and our software takes care of the rest. Note the chessboard on the turntable to merge clouds as the turntable moves. Sample objects from the dataset can be seen sitting on top of the Ortery Photobench.

2.1.1 Datasets

The last decade has witnessed rapid advances in computer vision largely due to fundamental image datasets, such as MNIST, Caltech-101, PASCAL, Labeled Faces in the Wild, PASCAL, ImageNet, and most recently, VQA and COCO[67,33,32,46,30,28? ?]. Unfortunately, the solution to most current 2D vision datasets would not constitute a solution to instance recognition as they currently target image retrieval tasks from arbitrary images drawn from the web. In particular, while some of these tasks emphasize detection, they do not directly address the problem of pose estimation, a component crucial to attaining high performance in instance recognition and robotic tasks. While there exist several 3D vision datasets, most datasets either (1) have few objects, (2) have low-quality objects, (3) provide only single views of objects or scenes, (4) don't contain calibration and pose information, or (5) provide low-resolution RGB data[sip,50,63,88,95,27]. While addressing all five aspects would improve the quality of instance recognition systems, aspect (5) would also provide a venue to explore synergies and comparisons between Kinect-style and multi-view stereo approaches to 3D model construction[38,36,42].

Furthermore, although most recent instance recognition systems work with RGB-D data, there exist high-quality instance recognition systems that use only RGB images, such as MOPED, presented by Collet et al.[25]. However, these generally work with higher-quality RGB images than those provided by RGB-D sensors. Unfortunately, this makes it quite difficult to compare RGB-D instance recognition systems with RGB-only systems, as simply applying the RGB-only systems to the images from RGB-D datasets would yield unrepresentative results. Because we provide high-quality RGB images in addition to the RGB-D data, we can enable meaningful comparison of these systems.

The closest work to ours is that of Kasper et al.[52]. They have a similar setup in which a laser scanner collects 3D data and a stereo pair collects data from 360 points from the viewing hemisphere. They also provide object meshes and calibrated RGB data. However, their 3D data collection setup is only semi-automated and their image collection setup takes an additional 20 minutes. Although they provide a relatively large number of objects (roughly 130 at the time of writing), scaling up to thousands may be infeasible at that speed. Our approach is fully automated after placing the object in the system, and data collection takes less than 5 minutes per object.

2.1.2 Data Collection

The chief obstacle to collecting a high-quality large-scale object dataset involves constructing a reliable 3D scanning system that can provide both high-quality depth and color information. Most commercial 3D scanners either provide only range sensor and low-resolution color information and/or are very expensive. Recent work demonstrates that KinectFusion variants can provide high-quality 3D reconstructions[47,79,107,119]. However, some of these approaches don't provide calibrated RGB images, which are required by many instance recognition platforms, and those that do only provide low-resolution RGB images from the Kinect sensor. Further, the data collection process requires a human to slowly move a Kinect around the full object; even with an automated turntable, a single Kinect attached to an arm cannot image non-convex objects and translucent/transparent objects due to the inherent limitations of Kinect-style RGB-D sensors.

Using multiple Kinects and high-resolution DSLR cameras along with an automated turntable constitutes one possible approach to jointly reducing human effort while improving RGB-D mesh quality. The presence of multiple types of sensors determines highly accurate intrinsics for each sensor as well as relative transformations between pairs of sensors. Researchers have extensively studied this problem for both single and multiple 2D cameras and have recently explored it for single and multiple RGB-D sensors[44,115,41,116,66,91,104]. Typical approaches involve first calibrating each sensor individually to compute its intrinsics, computing stereo pairs between sensors to estimate each sensor's extrinsics, and then running a joint optimization procedure to refine each sensor's intrinsics and extrinsics. For calibrating RGB-D sensors, many approaches require additional hardware and/or setup from what is required for 2D cameras. For example, Herrera et al.[44] present a method that requires affixing a chessboard to a large, flat plane, whereas typical 2D approaches simply require a chessboard alone. Our method requires a source of infrared light, but no additional hardware setup.

Additionally, interference between IR patterns complicates constructing a data-collection system with multiple RGB-D sensors. Butler et al.[17] propose an approach for mitigating interference from multiple depth sensors. However, their approach requires affixing a vibrating motor to each device, which makes a static calibration procedure impossible and also introduces more complexity into the system. We employ time-multiplexing, another common approach, which involves turning off each camera when it is not taking a picture. Concretely, we turn off the infrared emitter, which is roughly two times faster than turning off the depth stream.

2.1.3 Contributions

To address the issues described above, we present the following contributions:

1. A dataset which addresses the various shortcomings of existing 2D and 3D datasets by providing the following data per object: (1) 600 Kinect-style RGB-D images, (2) 600 high-resolution images, (3) accurate calibration information for every image, (4) segmented objects per image, and (5) full-object meshes. We obtain 600 images by taking shots from 5 polar angles and 120 azimuthal angles, the latter equally spaced by 3°.
2. A method for jointly calibrating multiple RGB-D sensors and cameras.
3. Details of our data collection system, which can collect all required data for a single object in under 6 minutes, where the only human effort required involves placing an object on the turntable and running a single command.
4. Multiple software components, including software for calibrating a single depth sensor, software for jointly calibrating multiple sensors (RGB-D and 2D RGB), and tools to simplify the data collection process.

In addition to helping to solve the instance recognition problem, we believe that our dataset removes many obstacles associated with large-scale 3D data and serves as a unified dataset that bridges problems in graphics, computer vision, and robotics. Our dataset can be used for benchmarks in multiple areas, such as 3D mesh reconstruction (with and without RGB-D), instance recognition, and object categorization. We intend to continually add to our dataset, inviting others to request and/or send us objects for which we have not yet collected data. Test scenes and results will be made available as well.

2.2 System Overview

The sensors in our system comprise of 5 high resolution (12.2 MP) Canon Rebel T3 cameras and five PrimeSense Carmine 1.08 depth sensors. We mount each Carmine to one of the T3s using a mount designed by RGBDToolkit[rgb], as shown in Figure 2.1(b). Each T3 is then mounted to the Ortery MultiArm 3D 3000.

We place each object on the turntable in the Ortery Photobench 260. The Photobench contains a glass turntable, which can be rotated in units of 0.5 degrees. It also has four lights, consisting of 4000 LEDs, located at the bottom, the back wall, the front corners, and the back corners. Using a reverse-engineered driver, we can programmatically control the lighting and rotation of the turntable.

To obtain calibrated data, we place a chessboard on the turntable; the chessboard is always fully visible in at least one of the cameras, specifically the Canon and Carmine directly above the turntable (see Figure 2.1 (c)). We refer to Carmine as the reference camera. After calibrating all of the cameras to find the transformations from each camera to the reference camera, we can provide a good estimate of the pose for every image.

For each object, we capture images with each camera at each turntable position. We rotate the turntable in increments of 3 degrees, yielding a total of 600 point clouds from the Carmines and 600 high-resolution RGB images from the Canon T3s. We then estimate poses for each camera, segment each cloud and generate segmentation masks for each of the 600 views, and produce a merged cloud and mesh model. Automation and speed are crucial to enabling large-scale data collection; a significant amount of engineering is required to make the process as fast as possible.

Our system runs the following steps when collecting data for a single object:

1. Start the depth and color stream for each Carmine. Turn off the infrared emitter for each Carmine.

2. Repeat for each turntable orientation (every 3 degrees, 120 total orientations):

   (a) Start a thread for each Canon T3 that captures an image.

   (b) Start a thread for each Carmine that captures a color image.

   (c) Start a single thread that loops through each Carmine, turning on the infrared emitter, capturing a depth map, and turning off the infrared emitter in sequence.

   (d) Once all of the above threads are done executing in parallel, rotate the turntable by 3 degrees.

Using all Carmines simultaneously causes the projected infrared patterns to interfere, leading to severe degradations in data quality. One option involves stopping the depth stream for each device not taking a depth image, and restarting the depth stream immediately before taking an image. However, stopping and starting a depth stream takes roughly 0.5s, imposing a 5 minute minimum bound on collecting 120 images with each of the 5 cameras. Rather than stopping the entire stream, we modified the OpenNI2 library to allow turning off the infrared emitter while keeping the depth stream open, which takes 0.25s. We present detailed timing breakdowns in Table 2.1.

Step	Time (s)
Startup	
Ortery Photobench Startup	3.5
Carmine Startup (depth and color)	9.3
Capture at each turntable position (done 120 times)	
Capture images – performed in parallel	1.82
Capture Canon T3 images (all 5 in parallel)	1.2
Capture Carmine color (all 5 in parallel)	0.07
Capture Carmine depth (all 5 in sequence)	1.82
Rotate turntable	0.48
Total capture time	276
Shutdown	0.49
Total time for one object	289

Table 2.1: Timing information for the data-collection process. Note that the three image capture threads all run in parallel, which means that the image capture step takes as long as the longest process.

2.3 Calibration

We now discuss how we jointly calibrate the sensors. The 10 sensors are situated in a quarter-circular arc, with each Carmine mounted to a Canon T3, and each Canon T3 mounted to the arm. One of the overhead cameras, referred to as the reference camera, can always see the chessboard affixed to the turntable; specifically, we use the overhead Carmine. In order to recover the pose of all of the other sensors, we must estimate the transformation from each sensor to the reference camera.

Kinect-style RGB-D sensor calibration involves estimating the intrinsic matrix for the infrared (IR) camera, the intrinsic matrix for the RGB camera, and the extrinsic rigid transform from the RGB camera to the infrared camera. Highly accurate calibration is crucial to achieving strong depth-to-color registration. In our system, we not only need to calibrate the intrinsics of each individual RGB-D sensor, but also the extrinsics which yield the relative transformations between each of the 10 sensors, both RGB-D and RGB.

Accurate calibration also enables registering depth maps to different RGB images, including the higher-resolution 1280x1024 image provided by the Carmine (hardware registration only works when the color stream is at the same resolution as the 640x480 depth stream). Although this is a relatively well-studied problem[91,44], obtaining strong results is still nontrivial due to multiple details about the Carmines that are not well documented.

Our method requires an external infrared light and a calibration chessboard. At a high level, we take pictures of the chessboard with the high-resolution RGB camera and the RGB-D sensor's infrared camera and RGB cameras[*], as well as a depth map. We then detect the chessboard corners in all of the images. Note that we turn off the infrared emitter before collecting infrared images, and turn it back on before collecting depth maps.

[*]It is vital that the Carmine and chessboard remain completely still while both images are captured, as it is not possible to simultaneously take a color and infrared image.

After collecting data, we first initialize the intrinsic matrices transformations for all fifteen cameras (five Canon T3s, five Carmines with an RGB camera and IR camera each) using OpenCV's camera calibration routines, based on the simple calibration method proposed by Zhang[116]. We also initialize the relative transformations between cameras using OpenCV's solvePnP. We then construct an optimization problem to jointly optimize the intrinsic parameters and extrinsic parameters for all sensors.

2.3.1 Joint Optimization

We use an approach similar to that given by Le and Ng[66]. Their approach requires that all sensors be grouped into 3D systems. A stereo pair of cameras (RGB or IR) yields one kind of 3D system (a stereo system), and a RGB-D sensor's infrared camera and projector yield the other (a RGBD system). Each 3D system has intrinsic parameters, used to produce 3D points, and extrinsic parameters, used to transform 3D points into another system's coordinate frame. We construct and solve the optimization problem using Ceres Solver[Agarwal et al.].

The calibrator optimizes the intrinsic and extrinsic parameters such that 1) each 3D system produces 3D points that match the physical characteristics of the chessboard (e.g. the points are all planar, the points on a given chessboard row are linear, and the distance between generated 3D points match up with the true distance on the chessboard) and 2) all 3D systems agree with each other on the locations of the chessboard corners.

The intrinsic parameters of a RGBD 3D system consist of the intrinsic matrix K and distortion parameters of the sensor's IR camera. The intrinsic parameters of a stereo 3D system consist of the intrinsic matrices and distortion parameters of each camera, along with the rotation and translation from one camera to the other.

The loss function is given by

$$\mathcal{G} = \sum_{s \in \mathcal{S}} \sum_{u \in \mathcal{U}} I(s, u) + \sum_{s_1, s_2 \in \mathcal{S}} E(s_1, s_2, u)$$

where I denotes the intrinsic cost, E denotes the extrinsic cost, $\mathcal{S}$ denotes the set of all 3D systems and U denotes the calibration data (i.e. the chessboard corners).

Let $Q(s, u_i)$ be a function that produces a 3D point for the corner u_i using the intrinsic parameters of system s. For a stereo system, this entails triangulation, and for an RGBD system, this is simply converting image coordinates to world coordinates using the depth value provided by the sensor.

For a 3D system, the intrinsic cost is given by

$$\begin{aligned} I(s,u_i) = &\sum_{u_j \in \mathcal{U}} (||Q(s,u_i) - Q(s,u_j)|| - d_{ij})^2 \\ &+ \sum_{l \in \mathcal{L}} d(Q(s,u_i), l) \\ &+ d(Q(s,u_i), p) \end{aligned}$$

where d_{ij} is the ground-truth 3D distance between points i and j on the chessboard, $\mathcal{L}$ is the set of lines that corner u_i belongs to, p is the plane that corner u_i belongs to, and $d(Q(s,u_i),p)$ measures the distance from the generated 3D point to the plane.

The extrinsic cost is given by

$$E(s_1,s_2,u_i) = ||R_{12}Q(s_2,u_i) + t_{12} - Q(s_2,u_i)||^2$$

where R_{12} and t_{12} represent the rotation and translation needed to transform a point from the coordinate frame of 3D system s_2 to s_1.

The major difference between our approach and that of Le and Ng is that we add one additional term to the cost function for stereo pairs; specifically, we enforce that epipolar constraints are satisfied by adding an additional term to the stereo intrinsic cost function:

$$I(s,u) = ||u_1^T F u_2||^2,$$

where F is the fundamental matrix implied by the current values of the stereo pair's intrinsic parameters, u_1 are the homogeneous coordinates of the calibration datum in the first camera, and u_2 are the homogeneous coordinates of the calibration datum in the second camera.

We obtain the depth intrinsic matrix K_{Depth} from the infrared intrinsic matrix by subtracting off the offset between the depth image and infrared image due to the convolution window used by the internal algorithm. We found the values suggested by Konolige and Mihelich[57] of -4.8 and -3.9 pixels the x and y directions, respectively, worked well. Figure 2.2 shows the results of registering the depth map to the RGB image using our calibration and also using hardware registration.

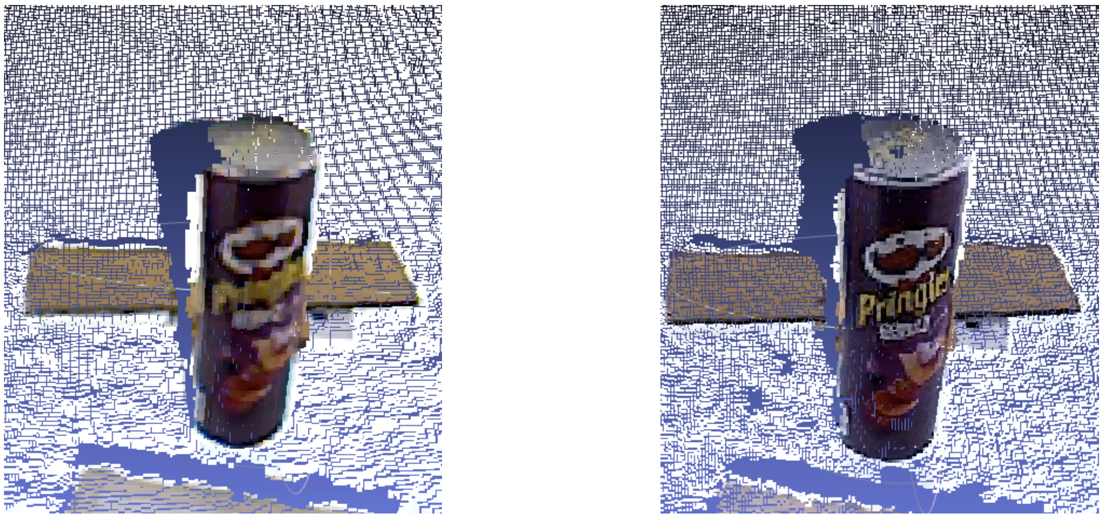


Figure 2.2: Comparison of hardware and software registration. The left image shows a hardware-registered point cloud. Note the bleeding of the cardboard in the background onto the Pringles can and the low resolution of the color data. The right image shows a software-registered point cloud using our calibration. Most of the bleeding of the cardboard onto the can has been fixed, and we can use higher-resolution color data.

2.4 3D Model Generation

After calibrating each camera to the reference camera, we proceed with model generation. At a high level, we:

1. Collect data from each Carmine and Canon as the turntable rotates through 120 3° increments.
2. Filter each Carmine depth map to remove depth discontinuities (see the next paragraph).
3. Generate point clouds for each Carmine view using calibration intrinsics.
4. Merge the 5 point clouds for each of the 120 scenes using calibration extrinsics.
5. Segment the object from the merged cloud (Section 2.4.2).
6. Improve the object cloud quality for each of the 120 scenes through plane equalization (Section 2.4.1).
7. Merge the 120 scenes together to form a single cloud using calibration extrinsics.
8. Create a mesh via Poisson Reconstruction[53,24].

After collecting data from each Carmine and Canon sensor, we run a depth discontinuity filtering step as suggested by Whelan et al.[105], since depth map discontinuities tend

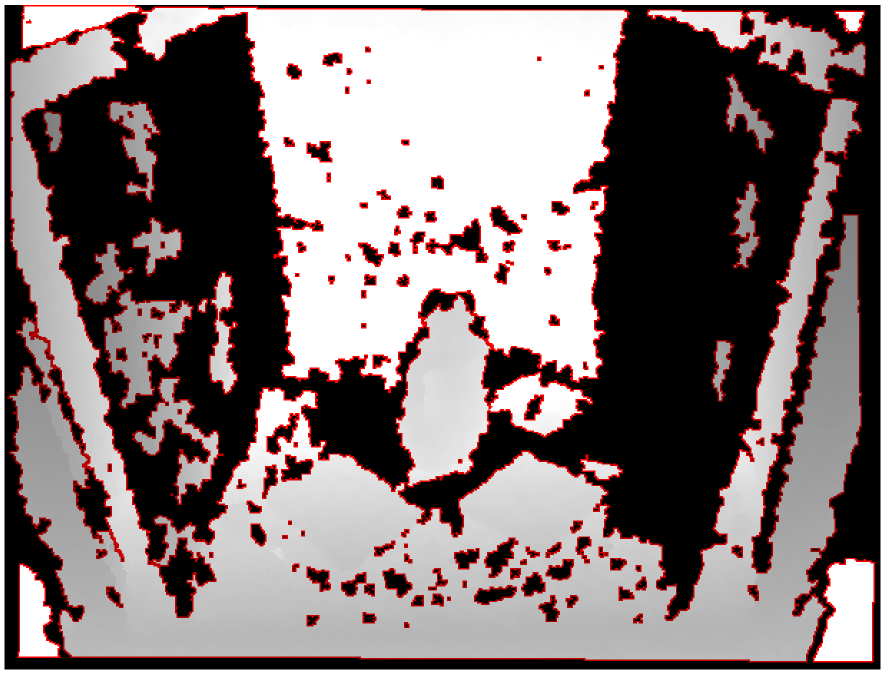


Figure 2.3: Applying depth discontinuity filtering. Pixels marked in red are considered unreliable due to either a discontinuity or neighboring pixels that were not measured by the Carmine depth sensor. Before proceeding, we discard depth measurements associated with the red pixels.

to yield imprecise depth and color measurements. To do so, we associate each 3×3 patch p in the depth image with a value $\max\{(\max p - p_{mid}), (\min p - p_{mid})\}$ where p_{mid} refers to the center pixel's depth. We keep all pixels whose associated patch has a value lesser than some threshold. See Figure 2.3 for an example of the pixels eliminated by depth discontinuity filtering.

2.4.1 Plane Equalization

After obtaining a preliminary 3D mesh, we produce a cleaner cloud through a procedure called plane equalization. As we collect point clouds, recall that we compute the transform from the turntable chessboard to the reference camera via OpenCV's solvePnP. Experimentally, we notice slight depth ambiguities when computing these transforms, evidenced by the non-aligned plane normals and inconsistent depths presented in Figure 2.4. Since we know that the turntable chessboard revolves about a circle roughly horizontal to the ground, we refine each transform's rotational component and translational component by (1) computing a new vector normal to be shared across all chessboards and (2) enforcing the centers of each chessboard to lie on a circle.

Concretely, given a set $\mathcal{T} = \{(R_1, t_1), \dots, (R_n, t_n)\}$ of chessboard poses, we produce a refined set $\mathcal{T}' = \{(R'_1, t'_1), \dots, (R'_n, t'_n)\}$ of chessboard poses. Note that an R_i operates

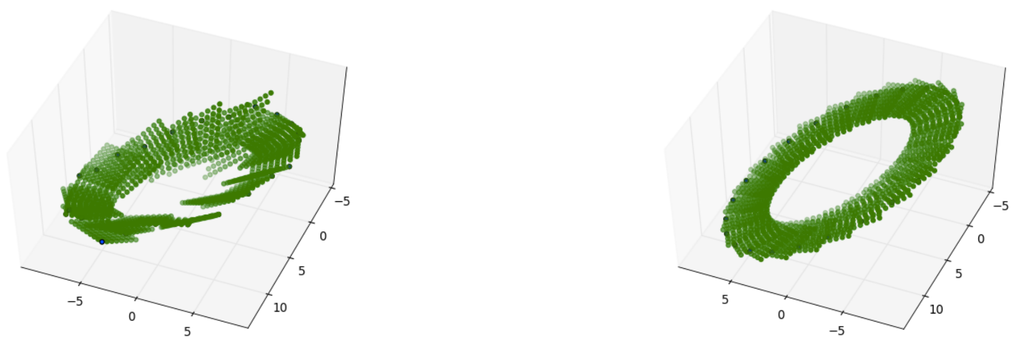


Figure 2.4: The chessboard poses for each turntable location are shown in the frame of the reference camera. On the left, the chessboard poses are determined by solvePnP. On the right, we refine these pose estimates using the plane equalization method described in Section 2.4.1. The refined board poses are significantly cleaner.

on a plane with unit normal $\hat{k}$ yielding a plane with unit normal $R_i[3]$, the third column of R_i. Ultimately, we would like all plane normals to match; to do this, we compute a unit vector $\hat{u}$ so as to minimize $\sum_{i=1}^{n}(\hat{u} \cdot R_i[3])^2$. We solve for $\hat{u}$ exactly by setting it to be the least eigenvector of the covariance of all the $R_i[3]$s. We then compute each R_i' by multiplying each R_i by the transform that takes $R_i[3]$ to $\hat{u}$ via rotation about the axis $R_i[3] \times \hat{u}$. We next compute each t_i' by projecting each t_i onto the least squares circle determined by $\{t_1, \cdots, t_n\}$; this problem can be solved quickly by projecting $\{t_1, \cdots, t_n\}$ onto a plane, computing the least squares circle in the plane's basis, and projecting each point onto the resulting circle. In practice, plane equalization runs in negligible time (< 0.1 s) for $n = 120$ and yields higher quality point clouds (see Figure 2.5).

2.4.2 Object segmentation

As discussed above, for a given turntable angle, we merge the 5 Carmine point clouds into a single cloud using calibration extrinsics. To segment the object from this cloud, we first discard all points outside of the Ortery PhotoBench. We then discard all points below the turntable plane (which was identified in the previous step), and lastly conduct agglomerative clustering to remove tiny clusters of points.

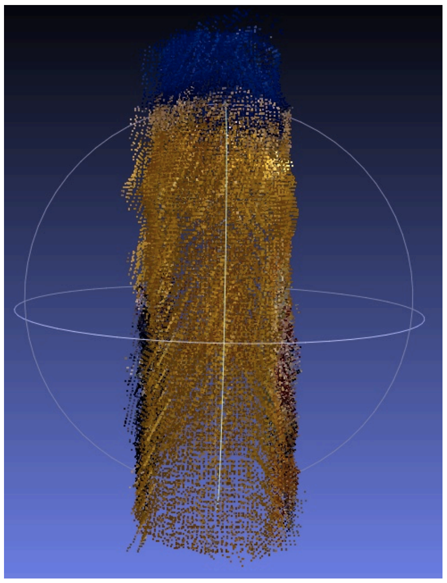
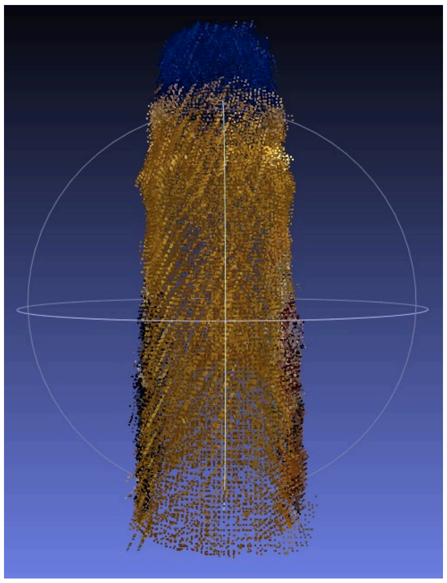



Figure 2.5: Constructed point clouds for one object. On the left, the cloud is constructed using the raw solvePnP poses; the cloud has multiple shifted copies of the object due to misalignment. On the right, the cloud is constructed with the output of the plane equalization procedure; the cloud is much cleaner and well-aligned.

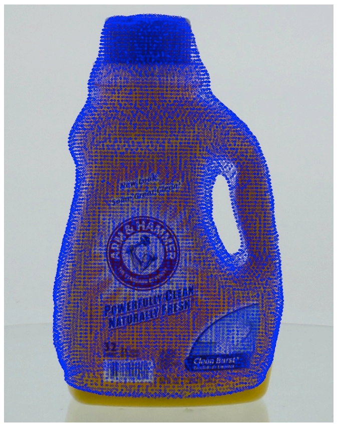


Figure 2.6: In the left image, the 3D mesh is projected onto one of the Canon images. In the right image, we show an example object for which Kinect-style RGB-D sensors yield poor-quality point clouds.

2.4.3 Accuracy

Although we use a naive approach for building 3D models, their accuracy is better than the models used by Xie et al.[108] to obtain state-of-the-art RGBD instance recognition results. In the left image of Figure 2.6, we give a rough idea of the accuracy of our 3D models by projecting a representative mesh onto an image from one of the Canon cameras (which is not used to build the mesh), showing that the system is well calibrated and produces reasonable meshes. We expect that more sophisticated algorithms can produce higher-fidelity 3D models.

2.4.4 Limitations

Our approach relies solely on point cloud data from the Carmines when building the 3D mesh models. However, Kinect-style RGB-D sensors are known to perform poorly for certain objects, including transparent and highly-reflective objects, such as the bottle shown in the right image of Figure 2.6. For these objects, the 3D models may be missing or of poor quality. However, by incorporating methods that also use RGB data, we anticipate being able to provide high-quality 3D models for many of these objects in the future.

2.5 Dataset Usage

The BigBIRD dataset and scanning rig has been employed as a solution towards several related computer vision problems, including object instance and category recognition[? ? ?] , robotics[89?] , scene understanding[?] , and 3D scanning[76?] . The dataset and all code used to generate it, can be obtained at our website (http://rll.eecs.berkeley.edu/bigbird).

2.5.1 Obtaining the Dataset

Due to the large size (and many uses) of the dataset (each object has roughly three gigabytes of data), it is impractical to provide a single downloadable file for the entire dataset, and inconvenient to have a single downloadable file per object. On our website, we provide an automated way to download the data for various use-cases. Instructions for downloading the data are provided on the website. The settings can be configured to download whichever subset of the following components are desired:

1. High-resolution (12MP) images (.jpg)
2. Low-resolution Carmine images (.jpg)
3. Raw point clouds (.pcd)
4. Depth maps (.h5)
5. Segmented point clouds (.pcd)
6. Segmentation masks (.pbm)

7. 3D mesh model (.ply)

2.6 Conclusions

We believe this dataset will significantly accelerate progress in robotic perception, especially the instance recognition problem. We also believe it can lead to benchmarks for a variety of areas from computer graphics, computer vision, and robotics, including 3D object reconstruction, recognition, and grasping.

All of our code and data, including calibration data, object instance data, and test scenes are available at the following URL: http://rll.eecs.berkeley.edu/bigbird. We plan to continue adding to our dataset; we invite others to request and/or send us objects which we have not yet scanned.

3
Range Sensor and Silhouette Fusion for High-Quality 3D Scanning

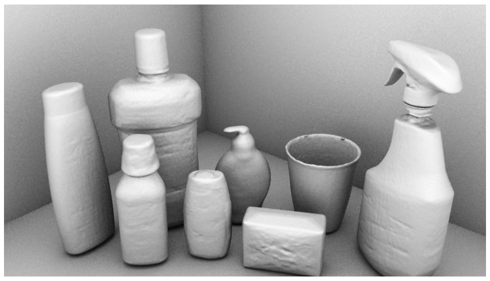
We now present a 3D reconstruction algorithm that produces high-fidelity 3D models given a collection of calibrated RGB and depth images. We use the data collected from the BigBIRD rig, described in Chapter 1, to benchmark our results. This work was published in ICRA 2015[76].

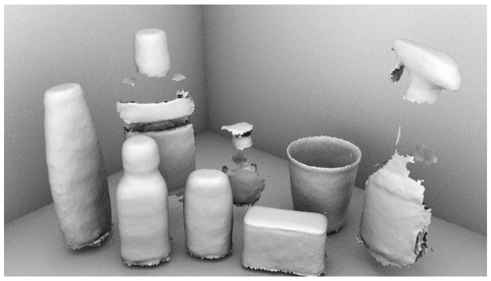
3.1 Introduction and Related Work

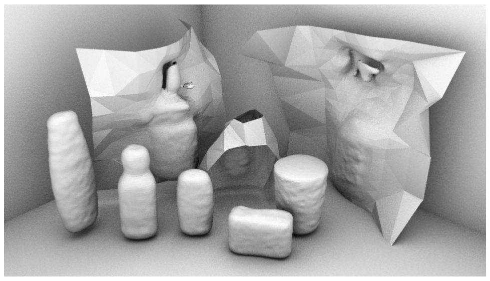
Variants on the recently proposed KinectFusion algorithm have become particularly popular in reconstruction due to the method's ability to reconstruct objects in real-time while recovering surface details[80,107]. As a depth sensor receives streaming depth images, KinectFusion (1) calibrates the current depth map using frame-to-model iterative closest point (ICP), (2) updates a truncated signed distance function (TSDF) that stores averaged depth readings, and (3) constructs a mesh using the marching cubes algorithm[69]. Recently published variants of KinectFusion discuss methods to account for the nonlinear distortions introduced by consumer-grade depth sensors[Zhou & Koltun,119,92].

While KinectFusion primarily uses depth cues in reconstruction, popular stereo techniques employ color cues in reconstruction. In particular, multiview stereo approaches currently obtain state-of-the-art results in reconstruction purely from multiple calibrated RGB images[39,20,49,61,73]. Most such methods produce a 3D point cloud, which is then used to compute a mesh representing the scene. There exist several approaches to compute this point cloud, such as plane-sweeping[26], stereo-matching[94], and patch growing[87].

In reconstructing a single object, one popular approach involves constructing the object's visual hull and iteratively deforming the hull towards multiview-stereo-based point clouds to extract fine details[39,20,37]. In particular, the visual hull attempts to


(a) Our method


(b) KinectFusion [22]


(c) Approach in [25] (Poisson reconstruction)

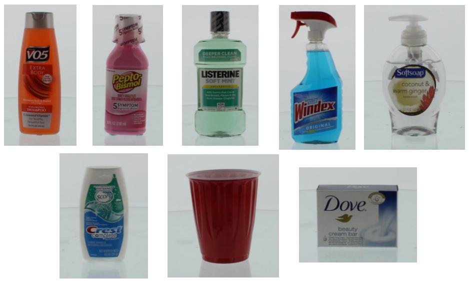

(d) Color images of objects in (a)-(c)

Figure 3.1: Collections of scanned objects reconstructed from (a) the method in this chapter, (b) KinectFusion[80], and (c) the previous approach in[90], and (d) colored versions of these objects. Back row, left to right: VO5 volumizing shapoo, Listerine, Softsoap Hand Soap (Coconut and Warm Ginger), red cup, Windex. Front row, left to right: Pepto Bismol, Crest Complete Minty Fresh Toothpaste, Dove Soap. Note that this image does not represent a scanned scene, but a collection of individually scanned objects to conserve space.

reconstruct an object purely from silhouettes captured from multiple views[10]. Noticing that each object silhouette backprojects to a cone, the visual hull intersects these cones to form a description of the real object's shape. Because the visual hull of an object is the envelope of all its possible circumscribed cones, the object must fully lie within its visual hull (see[86] for a proof and Figure 3.2 for a visualization in 2D).

Although multi-view stereo, visual hull, and KinectFusion-style approaches perform well in specific settings, they have pitfalls. While multi-view stereo approaches perform very well with highly-textured objects, the poor clouds resulting from lack of texture can lead to sparse and inaccurate clouds[39,20,49]. While the visual hull can recover the rough shape of an object even with objects with little texture, it fails to recover concavities in an object, which cannot be represented via calibrated silhouette data[86]. While KinectFusion-style approaches can perform well in the "concavity and low-texture" regime, they fail when working with objects that are translucent, transparent, or highly specular; the visual hull can provide better shape estimates here[80,92].

Little work has been published on combining each method's strengths. Steinbrucker et. al.[93] discuss an approach to jointly use RGB with depth data to improve camera calibration associated with KinectFusion, resulting in higher quality scene scans. Xu et. al.[109] present an approach that closely refines an object's depth data boundaries purely

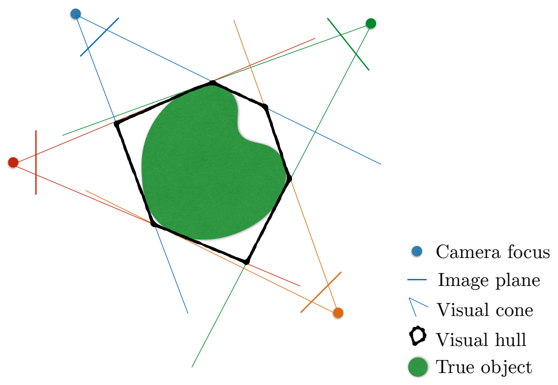


Figure 3.2: The red, green, blue, and orange have different views of the solid green object, and thus different silhouettes. Intersecting the cones formed by the silhouettes forms the visual hull, an approximation to the true object's surface. Although the visual hull provides better surface approximations as the number of cameras increases, the visual hull cannot recover the concavity in the true object.

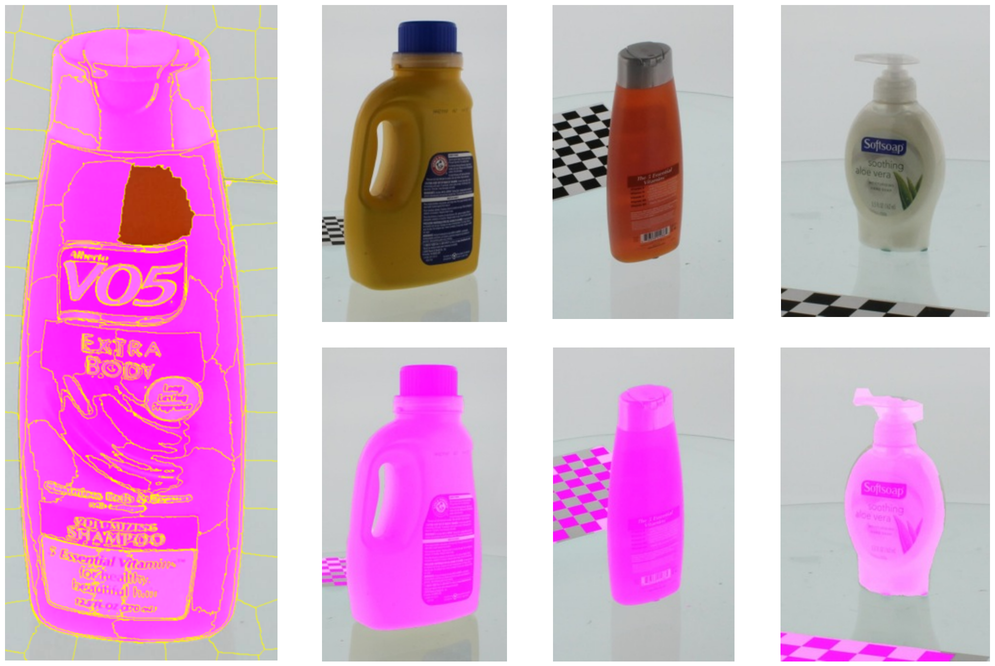
using silhouette information. As these approaches rely heavily on existing depth data from the Kinect, they do not work well in reconstructing objects that are translucent, transparent, or highly specular. Another work discusses a method to improve Kinect depth reading fidelity by fusing stereo information from both the IR and RGB sensors; although this approach reconstructs bits and pieces of translucent, transparent, and specular objects, the improved data alone is not sufficient to create high quality 3D models due to the large variances in depth estimates[22].

Contributions. This chapter presents a reconstruction method that capitalizes on the strengths of the RGB and depth modalities. Specifically, we take in as input a set of calibrated RGB and depth images and produce a high-resolution object mesh. By fusing silhouette and depth information, our method recovers detailed models for challenging objects that are difficult to reconstruct using either modality alone. We evaluate our results on the BigBIRD dataset, discussed in Chapter 1. Shown in Figure 3.1, our method outperforms competing approaches including the original BigBIRD models[90] and KinectFusion[80].

3.2 Unifying Image and Range Sensor Data

At a high level, our reconstruction pipeline (1) computes object segmentations from the high resolution RGB images (Section 3.2.1), (2) computes a visual hull using the segmented, calibrated RGB images (Section 3.2.2, 3.2.3), (3) computes a KinectFusion model using the calibrated depth data (Section 3.2.3), (4) refines the original raw depth maps using the visual hull and KinectFusion models and merges these refined depth maps into a point cloud (Sections 3.2.4, 3.2.5), and (5) forms an object mesh by fusing this merged point cloud with the visual hull (Section 3.2.6).

Our experiments (Section 4.4) illustrate that our approach capitalizes on the strengths



(a) Interactive Segmentation (b) A&H Detergent (c) VO5 Shampoo (d) Softsoap Handsoap

Figure 3.3: The interactive segmenter, (a), original images, (b-d top row), and automatically computed segmentations learned from manual segmentations (b-d bottom row). After tiling the image with superpixels using SLIC (separated by yellow boundaries in (a)), users can quickly select superpixels belonging to the object (pink regions). Selecting the last orange superpixel in (a) would complete the segmentation process. Although our learned background models misclassify chessboard regions in (b-d) and produces noisy segmentations for translucent objects (e.g., the Softsoap dispenser pump), our reconstruction scheme still recovers very good object shape (Section 4.4). Best viewed in color.

of both the KinectFusion and the visual hull approaches to recover accurate shape models even for objects with concavities and translucent parts.

3.2.1 Computing Object Segmentations

Given an object, we first describe a method to extract silhouettes for each of its 600 high-resolution RGB images, which we use to compute the visual hull. We first manually segment only the first view of each of the 5 Canon DSLR cameras. Specifically, after running Simple Linear Iterative Clustering (SLIC)[4] to tile an image with superpixels, we manually select the superpixels belonging to the object. The interface, with a sample object, is shown in Figure 3.3(a); users select computed SLIC superpixels (marked with yellow boundaries) belonging to the object. This process takes under 2 minutes per object (for all 5 segmentations) and is the only manual step in our pipeline. Note that the user only has to manually segment 5 objects, and not all 600; we show

how to automatically compute the rest of the segmentations below. In particular, we use 5 segmentations, one per DSLR camera.

Using only 5 manually segmented images, we can recover segmentations for all views: per manual segmentation, we construct a dataset $\{(\text{pix}_i, y_i)\}_i$ where $y_i = 1$ denotes an object-pixel and $y_i = 0$ denotes a background pixel, and pix_i denotes the color of the pixel in LAB space. We run k-means ($k = 20$) on the background pixels, i.e. $\{\text{pix}_i | y_i = 0\}$, initialized with k-means++[7]. For all pixels in the manual segmentation, we store the Euclidean distance to the closest mean; next, we use the manual labels to compute a threshold T such that pixels closer than T to a cluster center are classified as "background" while the rest are classified as "foreground." Specifically, we choose T by maximizing the number of correctly predicted pixels; because the number of pixels is on the order of a few million, we can optimally select T by considering all possible distances that the pixels take on.

For the remaining 119 images captured from each of the 5 cameras, we then use the corresponding background model to classify each pixel as object or background. Finally, we mark superpixels as object-superpixels if they contain greater than 30% coverage of object-pixels, and background-superpixels otherwise. Figures (b-d) show examples of marked object superpixels with a magenta tint; we deliberately use a low threshold of 30% to recover object superpixels belonging to white/translucent/transparent objects. Though this introduces false positive object superpixels (e.g. see marked chessboard superpixels in (b-d)), the visual hull carves these regions away, as the same false-positive superpixel rarely appears in many camera views. Armed with these silhouettes and the calibration information per image provided in the BigBIRD dataset, we can then construct the visual hull.

3.2.2 Computing Visual Hull Models

Many variations on computing visual hulls exist[34,65,71,62,35]. In this chapter, we consider the method discussed in[34], which offers good tradeoffs between speed and accuracy: we (1) define a function $F(x) = 1$, if $x \in \mathbb{R}^3$ falls in all silhouettes, and 0 otherwise, (2) compute an initial point x_0 such that $F(x_0) = 1$, i.e. an initial point that lies on the visual hull, and (3) run an implicit surface polygonizer to recover the visual hull mesh, using x_0 as an initialization. We can compute x_0 by considering the 3D back-projections of the bounding boxes per silhouette; repeatedly sampling points within the intersection of these bounding boxes eventually yields a valid x_0 (see[20] for how to compute this intersection). For step (3), we use the publicly available Blumenthal polygonizer with marching tetrahedra[13,98].

This visual hull approach leads to excessive object carving arising from object pixels being labeled as background pixels during segmentation (see Figure 3.4, left objects). Such errors typically arise when the object either has bright, white colors or white specularities near segmentation boundaries. To ameliorate this problem, we account for silhouette noise by instead polygonizing the surface $G(x) = 1$ if $x \in \mathbb{R}^3$ falls in $1 - \epsilon$ of all silhouettes, and 0 otherwise (we set $\epsilon = 0.1$ in our experiments).

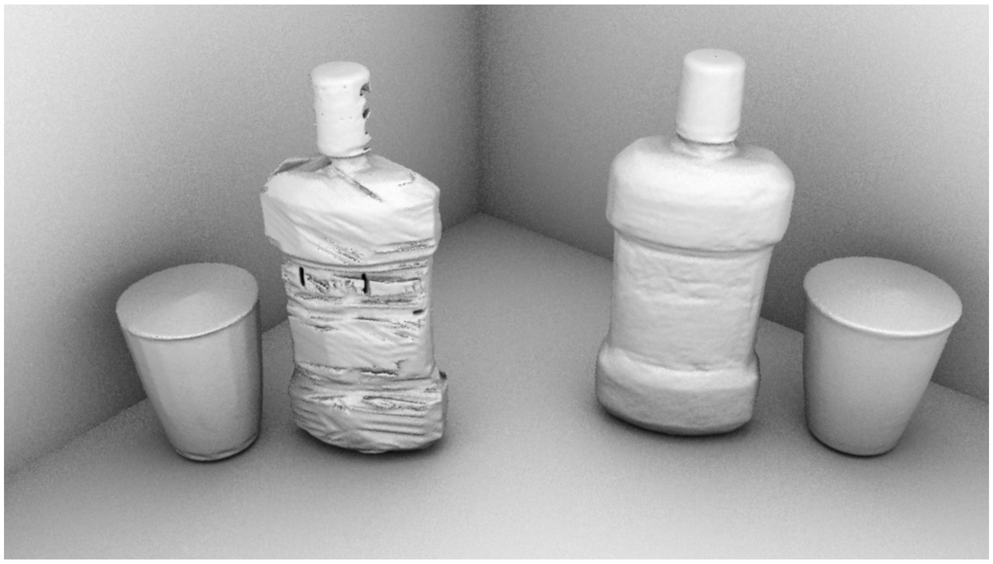
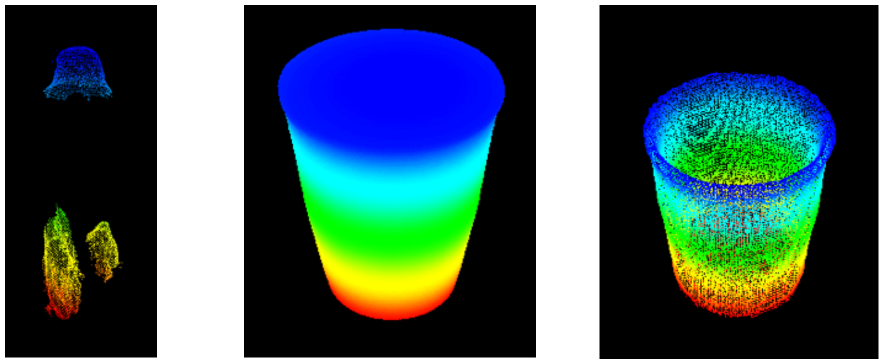


Figure 3.4: Comparisons of the hard visual hull (left two objects), using the method in[34] and the soft visual hull, using the method in Section 3.2.2 (right two objects).

Despite being a crude approximation, the right two objects in Figure 3.4 demonstrate that this approach works well in practice; the hard visual hull recovers finer surface detail for the cup, but heavily over-carves the Listerine bottle due to segmentation errors. The soft visual hull forgives the segmentation errors in the Listerine bottle that cause excessive carving, but smooths the cup's surface details. We use the soft visual hull, as it generally conforms better to an object's shape. As expected, both methods fail to recover the cup's concavity.

3.2.3 Computing Calibrated KinectFusion Models

We can jointly use the depth camera data to recover object concavities; since individual depth maps are inherently noisy, we fuse the depth maps into a single mesh using a variant of the KinectFusion algorithm. The KinectFusion algorithm assigns poses to each incoming camera frame via frame-to-model, point-to-plane ICP[80]; because KinectFusion requires slow camera pose movements, and the BigBIRD dataset has 5 cameras in relatively far away locations (see Figure 2.1c), we use the camera poses that the BigBIRD dataset provides per Carmine rather than frame-to-model ICP. Separate experiments indicated that the provided poses provide more reliable poses over those provided by frame-to-model ICP (particularly in cases where depth maps have gaps due to transparencies). The provided poses also allow us to use depth information from all 5 cameras. We employ a CPU implementation of KinectFusion that represents the TSDF structure using an octree; specifically, we adopt the implementation used in[119].



(a) Palmolive detergent, KinectFusion mesh vertices

(b) Red cup, visual hull mesh vertices

(c) Red cup, KinectFusion mesh vertices

Figure 3.5: Challenges associated with the depth modalities we attempt to blend. Point clouds recovered from KinectFusion meshes can often have large gaps of missing space[80]. These gaps could be misleading due to missing depth readings arising from object transparencies (see (a), featuring the KinectFusion mesh vertices of the translucent Palmolive dishwashing soap). Gaps could also be legitimate, recovering object concavities such as the cup's concavity in (c). Although the visual hull can fill in illegitimate gaps, it can also introduce hallucinated points which cover important concavities, such as the cup's concavity in (b). Section 3.2.4 and 3.2.5 discuss methods to capitalize on each modality's strengths and reason through the inaccuracies that each modality introduces.

3.2.4 Refining the Original Depth Maps

To ultimately fuse the visual hull and KinectFusion models, we aim to construct a dense cloud whose points lie on the surface of the object and deform the visual hull towards this cloud. In particular, this dense cloud's points will be a subset of the union of the visual hull and KinectFusion mesh vertices. Selecting this subset is nontrivial, since the visual hull introduces hallucinated vertices, e.g., the top of the red cup (Figure 3.5b). Further, KinectFusion models can contain large empty spaces in regions with few depth readings (Figure 3.5a). Exacerbating the problem, gaps in the KinectFusion model could either be due to legitimate gaps such as object concavities (Figure 3.5c) or illegitimate gaps due to object transparencies (Figure 3.5a). Ultimately, we construct the desired cloud by refining the raw depth maps.

We summarize our algorithm to fuse these depth cues in Algorithm 1. We now explain how Algorithm 1 assigns a refined depth to a single pixel (i, j) given z-buffered depths for the visual hull and KinectFusion maps vh and kf and a raw depth map raw, for a single camera c in angle a. The following 4 cases correspond to the 4 cases described in Algorithm 1.

Case 1: vh does not project onto (i, j). We assume that the recovered mesh strictly lies within the visual hull, so if $vh[i, j] = \infty$, we use a refined depth of ∞. The remainder of the cases assume that vh projects onto (i, j).

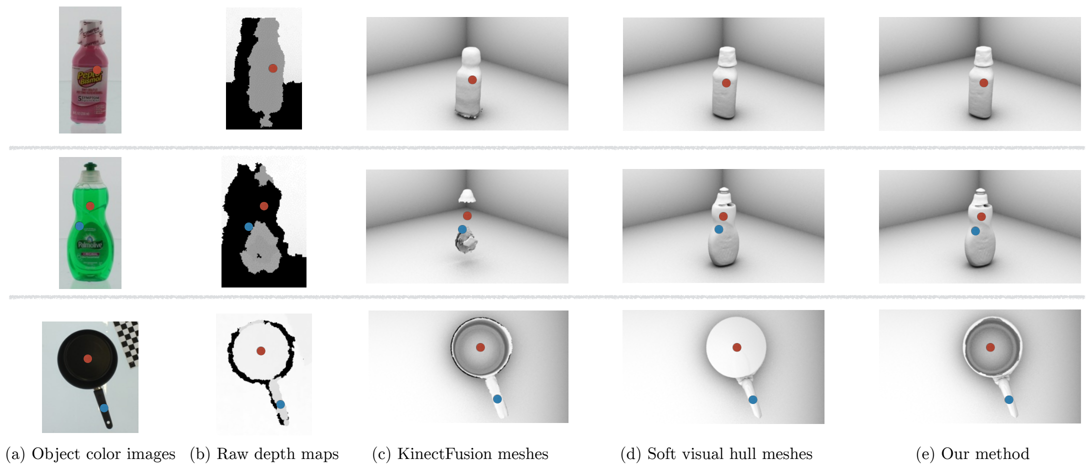


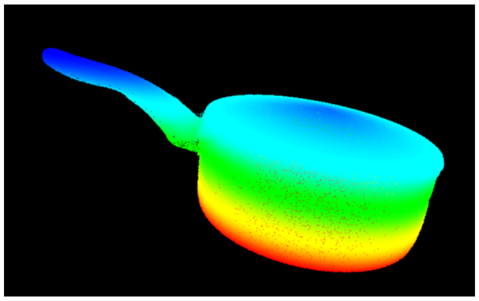
Figure 3.6: Three concrete cases of the intuition behind merging visual hull and KinectFusion depth information. Section 3.2.4 explains how Algorithm 1 operates on (1) the red corresponding points in the first row (Pepto Bismol, a simple, opaque object), (2) the red and blue corresponding points in the second row (Palmolive dishwashing container, an object with major translucencies), and (3) the red and blue corresponding points in the third row (black pot, an object, an object with a major concavity). Our method takes the best pieces of the KinectFusion and soft visual hull meshes; this is particularly evident in the third row, where our method recovers the pot's concavity from the KinectFusion mesh and the refined handle from the soft visual hull.

Case 2: Either raw or kf has a missing depth at (i, j). Because vh projects onto (i, j), Algorithm 1 interprets this case as the Carmine missing readings due to transparencies. For example, this happens for the red point in the second row of Figure 5.2, the translucent Palmolive dishwashing liquid, where the soft visual hull is reliable while the KinectFusion and raw depth maps have missing depths. In this case, Algorithm 1 prescribes returning the visual hull's depth.

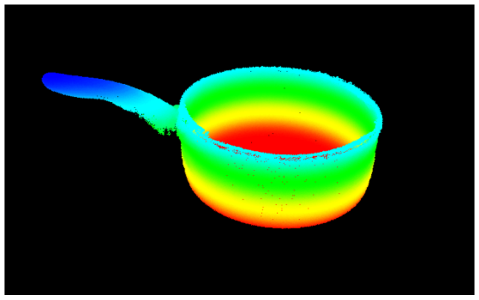
For the blue point in the same row, the raw depth map returns a missing depth while the KinectFusion mesh actually returns a depth reading; note that the KinectFusion depth reading is spurious since the depth reading represents a point that actually lies "inside" the object rather than on the object's surface. Specifically, the KinectFusion mesh returns a false depth reading here, since this point is directly visible to the camera when it isn't supposed to be (namely because transparent surfaces are not reconstructed). According to Algorithm 1, we return the visual hull's depth, as desired.

Case 3: vh and kf return readings closer than 1 mm. In this case, we opt to use vh's depths, since surfaces recovered by the visual hull tend to be more refined. For example, shown in the first row of Figure 5.2, the KinectFusion and soft visual hulls for the Pepto Bismol container are both fairly reliable. Algorithm 1 returns the visual hull reading for the red point.

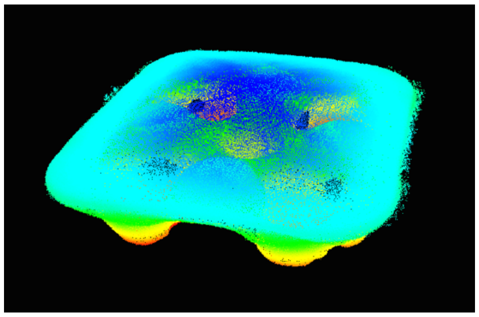
Case 4: vh and kf return readings farther than 1 mm. In this final case, we opt to return the maximum depth between vh and kf, as this likely implies the presence



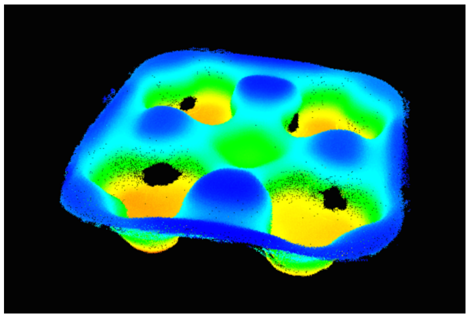
(a) Pot, with hallucinated points



(b) Pot, hallucinated points removed



(c) Cup holder, with hallucinated points



(d) Cup holder, hallucinated points removed

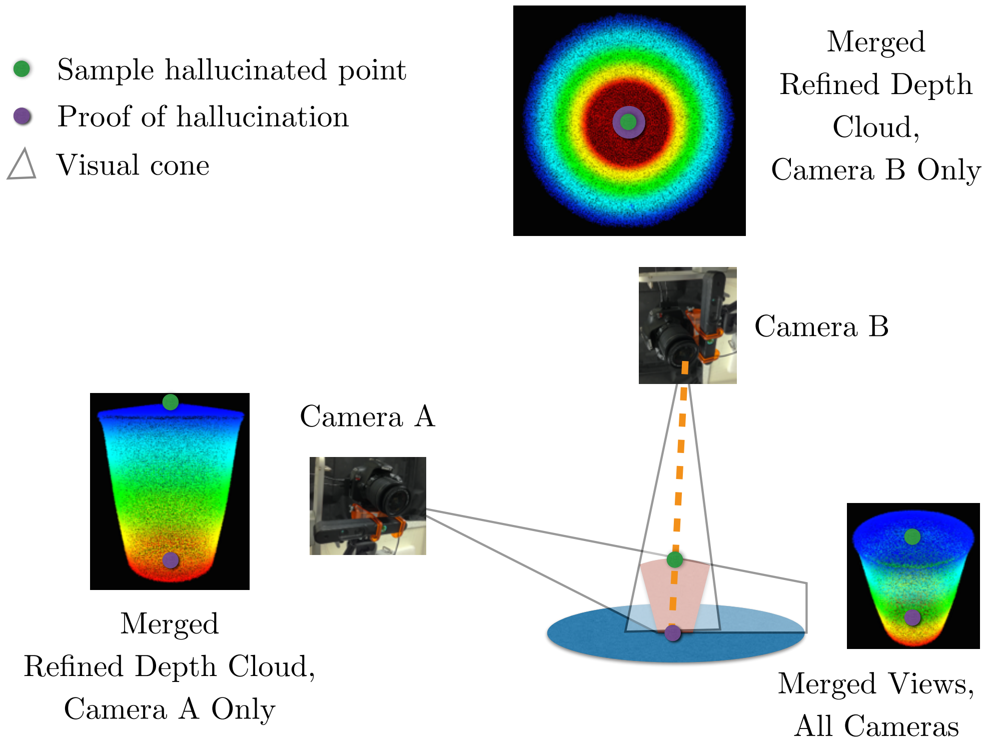
Figure 3.7: Point clouds of a black pot (a) immediately after applying Algorithm 1 and (b) after applying our hallucination removal scheme. We similarly show point clouds of an object with more complex concavities (a paper cup holder) in (c) and (d). See Figure 4.7 for color images of these objects.

of a concavity. We address this case in the third row of Figure 5.2, which presents a black pot with a large concavity. In the red point, both KinectFusion and soft visual hulls return valid depth readings, but these readings differ by more than 1 mm. Taking the maximum depth reading gives us the KinectFusion depth, allowing us to properly recover the concavity.

Algorithm 1 opts to choose vh depths for the blue point, which falls under Case 3. This leads to a more refined handle, showing that Algorithm 1 can pick and choose parts of vh and kf based on reliability.

3.2.5 Eliminating Hallucinated Points

Algorithm 1 has a shortcoming, namely that it always assigns a finite depth to points that fall within vh. Due to segmentation errors and the soft visual hull threshold, it is possible for points to fall within the soft visual hull, but not fall within the true object. Because the Carmine and raw depth maps indicate such points to be outside the object, Algorithm 1 treats these hallucinated points as part of Case 2, generating hallucinated points. As an example, Algorithm 1 frequently hallucinates points where



Sample hallucinated point
Proof of hallucination
Visual cone
Merged Refined Depth Cloud, Camera B Only
Camera B
Camera A
Merged Refined Depth Cloud, Camera A Only
Merged Views, All Cameras

Figure 3.8: Camera A's viewing cone does not fully carve away the visual hull, leaving a sliver of hallucinated points; the green dot shows a sample hallucinated point (see Camera A's refined depth map). Discussed in Section 3.2.5, the existence of the purple dot along the orange ray allows us to detect the green dot as a hallucinated point.

the visual hull fills in concavities (see Figure 3.7). Figure 3.8 presents a visualization for why this happens for a cup: camera A's cone does not fully carve away the top of the cup, leaving the sliver of points covering the concavity.

We eliminate hallucinated points as follows: for each point P generated by the visual hull in the cloud generated by Algorithm 1, project P onto each refined depth map D. If P's depth in D's frame is strictly smaller than the current depth value in D, then discard P.

Figure 3.8 explains why this method works: camera A introduces a hallucinated point at the green dot. We can use camera B to resolve this discrepancy; projecting the green dot onto camera B's image plane shrouds the purple point, which camera B originally sees with Algorithm 1 (see Camera B's refined depth map). Eliminating this green dot resolves the contradiction that arises from the purple dot indicating free space along the orange ray.

Because we assume that the Carmines do not usually hallucinate poins, we iterate only through points generated by the visual hull. Further, as long as the KinectFusion model provides depth readings in concave regions of the object, the hallucinated points

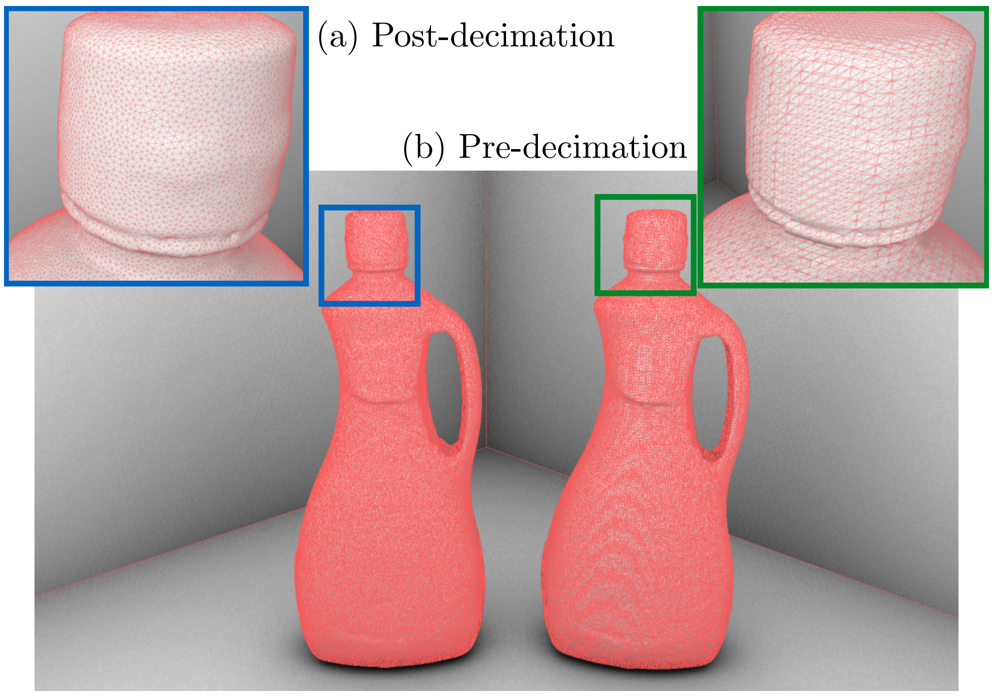


Figure 3.9: Zooms (a) and (b) are meshes obtained after fusing together the visual hull and dense cloud after hallucination removal (Section 3.2.6). (b) shows the mesh triangles before applying our decimation procedure. (a) shows the triangles afterwards. Note the substantially improved triangle quality, without the loss of surface geometry quality.

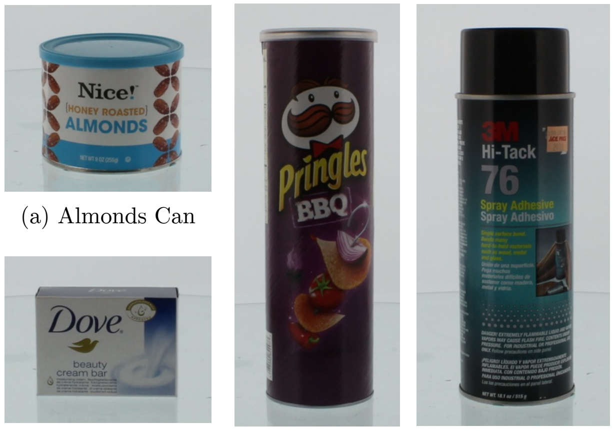
that “cover up” the concavity will be eliminated. See Figure 3.7b, d for merged clouds after hallucination removal of two sample objects. Finally, to remove stray points in this de-hallucinated cloud, we remove all points that do not have at least 5 neighbors within a radius of 1 mm.

3.2.6 Fusing the Unified Cloud and Soft Visual Hull

We now fuse the soft visual hull computed in Section 3.2.2 and “de-hallucinated” cloud computed in Section 3.2.5 into a single model. Similar to the visual hull formulation, for $x \in \mathbb{R}^3$, we define a function $F(x) = 1$ if x projects within all silhouettes and x’s nearest neighbor to a point in the de-hallucinated cloud lies within a maximum distance $r = 1$ mm. After finding an initial x_0 on the surface by repeatedly sampling from the intersection of the back-projected silhouette bounding boxes, we run the Bloomenthal polygonizer to extract a mesh.

In practice, the polygonizer generates many triangles with poor aspect ratio, e.g. slivers.[*] Triangles with aspect ratios close to 1 are desirable, since this leads to cleaner

[*]The aspect ratio of a triangle is defined as the ratio of the circumradius to twice the inradius;



(a) Almonds Can

(b) Dove Soap Box (c) Pringles Can (d) 3M Spray

Figure 3.10: The Almonds Can, Dove Soap Box, Pringles Can, and 3M spray, which we use for quantitative measurements. We model the Almonds, Pringles, and 3M Cans using 3 cylinders: the cap, the container, and the bottom. We model the Dove Soap Box using a rectangular prism.

meshes that are more easily editable in 3D software due to their connectivity properties; this also leads to cleaner texture maps[20]. To improve triangle aspect ratios while losing little surface detail, we alternate between (1) applying $\sqrt{3}$-subdivision without smoothing[55] and (2) applying an edge decimation procedure. Figure 3.9 shows a sample mesh before and after two iterations of this subdivision-decimate process.

3.3 Experiments

Figure 4.7 presents reconstructions of 19 distinct objects that fall in 3 categories: (1) simple and easy to reconstruct (relatively opaque, without concavities), (2) objects with at least one major concavity, and (3) objects with major translucencies or transparencies. Each row presents a different object category while each column presents reconstructions obtained from a different algorithm: the Poisson reconstruction method from[90] (PR), the hard visual hull method from[34] (VH), KinectFusion[80] (KF), and our approach. Images do not represent scanned scenes, but a collection of individually scanned objects to conserve space.

3.3.1 Simple Objects

Our method outperforms the other three competing methods in reconstructing simple objects. PR and KF tend to oversmooth surface details; further, neither method produces satisfactory reconstructions of the crayon. While PR produces closed meshes, KF often does not, evidenced by the noisy polygons towards the bottom of the objects.

e.g., the aspect ratio of an equilateral triangle is 1.

Although VH recovers the crayon, it overcarves the VO5 shampoo and Pepto Bismol, and fails to reconstruct the other objects due to excessive carving.

Our approach preserves the best aspects of each method: we recover surface details without excessive carving, evidenced by the properly reconstructed crayon and cap details for the Pepto Bismol, spray adhesive, and shampoo, while retaining closed meshes.

3.3.2 Objects with Concavities

In reconstructing concave objects, PR and VH miss surface details and hallucinate polygons. PR produces hallucinated polygons that cover concavities for the red cup, paper cup holder, carrying tote, and black pot as well as extraneous polygons around the paper plate. Due to excessive carving, VH fails to recover the paper cup holder, mangles the pot's surface, and fails to recover the pot's handle. KF and our method both recover all concavities. However, our method produces more refined models: our pot's handle is more clearly defined, and the edges of our objects are not as rough due to our closed meshes.

3.3.3 Objects with Translucencies

PR produces many hallucinations in reconstructing translucent objects in an effort to reconstruct areas with few depth points. Unfortunately, this yields nearly unrecognizable reconstructions. Again, VH overcarves objects, entirely missing 3 objects. For the remaining 3, VH misses the top halves of the Palmolive and Windex bottles and overcarves the Listerine bottle. KF recovers bits and pieces of all objects, but still misses large regions due to the translucencies.

Our method recovers the majority of objects, including tiny surface details: we recover the opening to the Palmolive bottle, indentations on the Coca-Cola bottle where the label is present, and bottle caps of the Coca-Cola and Dragon Fruit juice bottles, all details on the order of 1 mm or less. Further, our method properly recovers objects that are white, with large clear regions, shown by the Softsoap Hand soap and Bai5 Sumatra Dragon Fruit juice.

Our approach is not perfect however, as it fails to recover regions of the Windex's pipe and Palmolive bottle. Our automatic segmentations fail to recover the superpixels corresponding to these regions, as our segmentation method classifies the white/translucent colors of the pipes as part of the background.

3.3.4 Quantitative Measurements

Quantitatively measuring errors associated with the BigBIRD dataset is nontrivial, as there exists no ground truth data. We inspect 4 objects that can be decomposed into a few primitives: a Pringles can, an almonds container, a Dove soap box, and a 3M spray (Figure 3.10). We model the Pringles, almonds, and 3M spray containers using 3 cylinders stacked on each other and the Dove soap box using a rectangular prism. We

Primitive Fitting RMS Errors (mm)				
	PR[90]	SVH	KF[80]	Our Method
Pringles	0.566	0.541	0.850	0.563
Dove Soap	0.995	0.981	1.123	0.948
Almond Can	0.339	0.303	0.662	0.294
3M Spray	2.018	1.971	2.189	1.958

Table 3.1: RMS Errors from fitting ground truth models to reconstructions from Poisson reconstruction (PR), the soft visual hull (SVH), KinectFusion (KF), and our method.

determined the appropriate dimensions per primitive using calipers that are accurate to 0.1 mm. After fitting the appropriate ground truth model to each reconstructed object via point-to-plane ICP, we obtain the RMS errors in Table I; we compare our method to the soft rather than the hard visual hull, as the latter heavily overcarves the Dove box and Almond Can. Thanks to our well-calibrated cameras, most RMS errors are under 2 mm. In the case of the Dove Soap, Almond Can, and 3M spray our method produces reconstructions with the least RMSE; with the Pringles, our method comes in a close second place.

3.4 Conclusion

This chapter reasons through the advantages and shortcomings of the KinectFusion and visual hull techniques, and arrives at a highly effective method to fuse these models together. Individually, the KinectFusion algorithm does poorly in reconstructing objects with major translucencies but reconstructs concavities, while the visual hull does poorly in reconstructing concavities but reconstructs regions with major translucencies. Our method exploits the complementary nature of KinectFusion and visual hull to produce a unified algorithm that properly recovers objects with concavities and translucencies.

Algorithm 1 Refining the Original Depth Maps

```
$C \leftarrow$ list of depth cameras whose depth maps to refine
$A \leftarrow$ list of turntable angles per depth camera
$VH \leftarrow$ the soft visual hull mesh
$KF \leftarrow$ the KinectFusion mesh
$cloud \leftarrow$ empty point cloud
for $c$ in $C$ do
    for $a$ in $A$ do
        $raw \leftarrow$ raw depths for camera $c$, angle $a$
        $vh \leftarrow VH$'s z-buffered depths in camera $c$, angle $a$
        $kf \leftarrow KF$'s z-buffered depths in camera $c$, angle $a$
        $nr \leftarrow$ number of rows in $raw$
        $nc \leftarrow$ number of columns in $raw$
        $refined \leftarrow$ empty array with $nr$ rows and $nc$ cols
        for $0 \leq i < rows$ do
            for $0 \leq j < cols$ do
                if $vh[i,j] = \infty$ then
                    $refined[i,j] = \infty$                                ▷ Case 1
                else if $raw[i,j] = \infty$ or $kf[i,j] = \infty$ then
                    $refined[i,j] = vh[i,j]$                               ▷ Case 2
                else if $|vh[i,j] - kf[i,j]| < 1$ mm then
                    $refined[i,j] = vh[i,j]$                               ▷ Case 3
                else                                                       ▷ Case 4
                    $refined[i,j] = \max\{vh[i,j], kf[i,j]\}$
                end if
            end for
        end for
        $newcloud \leftarrow$ point cloud generated from $refined$
        $cloud$.appendPointsNotAtInfinity($newcloud$)
    end for
end for
return $cloud$
```

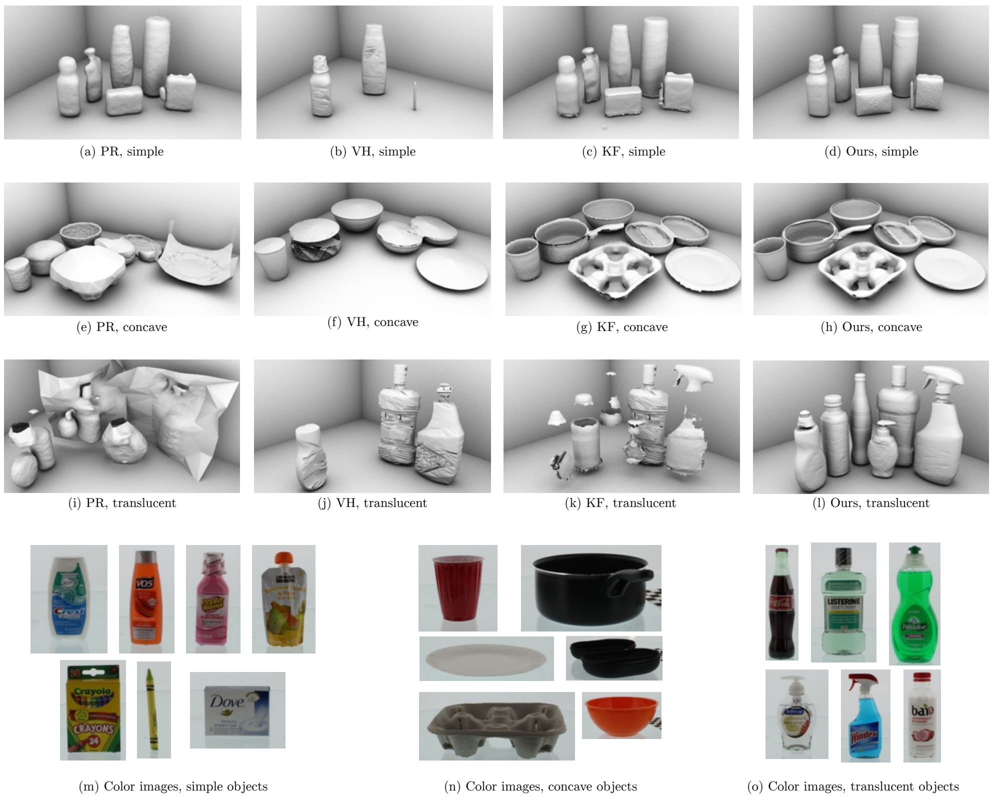


Figure 3.11: Collections of scanned objects reconstructed by competing methods. Rows 1-3 present objects that (1) are “simple”, i.e. have few concavities, and are mostly opaque), (2) have major concavities, and (3) have major transparencies/translucencies, respectively. Columns 1-4 present reconstructions produced by the Poisson reconstruction-based method in the original BigBIRD [90,54] (PR), the visual hull method in[34] (VH), KinectFusion[80] (KF), and our approach, respectively. Color images of simple, concave, and translucent/transparent objects are presented in (m)-(n), respectively. Each image does not present a scanned scene, but a collection of individually scanned objects to conserve space.

4 Optimized Color Models for High-Quality 3D Scanning

Now that we have an effective algorithm for reconstructing high-quality 3D meshes given a collection of well-calibrated RGB and depth images, we consider the problem of estimating high-quality color models of these reconstructed meshes. We continue using the data collected from the BigBIRD rig. This work was published in IROS 2015[74].

4.1 Introduction and Related Work

Recovering accurate color models given a shape model and a collection of color images has been keenly explored. Accurate color models of 3D objects have been shown to play major roles in object and instance recognition, particularly for cluttered scenes[108,97,6,70]. These approaches automatically annotate reconstructed shape models with color and texture features computed from calibrated RGB images. Other than robot vision, applications of high quality color models include virtual reality, computer graphics, and online shopping. In this chapter, we propose a color model reconstruction method that outperforms competing methods.

Representing the reconstructed shape model as a mesh, a collection of triangles on a given vertex set, several widely used methods typically involve variants on volumetric blending, i.e., assigning an averaged color to each mesh vertex[20,40,80,107,78,110,106]. These approaches assume provided calibration information; in tandem with the provided mesh, these approaches establish corresponding points across the images and compute weighted averages for each mesh vertex. While some approaches advocate employing averaging with uniform weights[80], others recommend assigning greater weight to views that observe the vertex more frontally[107]. In an effort to combat specularity problems and the lack of frontal views for vertices, some approaches disregard viewpoint extrinsics and instead give higher weights to views with greater color saturation[43,20].

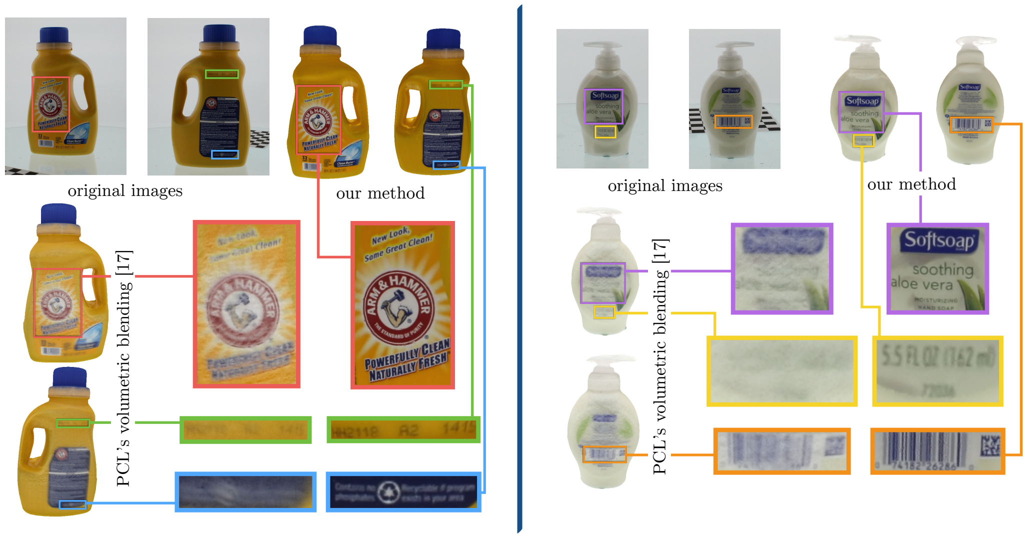

original images
our method
PCL's volumetric blending [17]
original images
our method
PCL's volumetric blending [17]

Figure 4.1: Arm & Hammer Detergent (top) and Softsoap Aloe Vera (bottom), reconstructed using our method and PCL's volumetric blending[80]. The BigBIRD scanning rig (Figure 2.1) captures 600 DSLR RGB images per object[90]; we show two per object here. Our color models can recover very fine textural details: in the Aloe Vera soap, the "5.5 FL OZ" text is 3 mm tall; the numbers under the barcode are just over 1 mm tall. PCL's volumetric blending smooths away these details.

Given initial camera poses, a different set of methods explores how to locally optimize camera poses so as to maximize color agreement[82,118,81].

We jointly recover camera poses and a color model by efficiently solving a non-linear least squares optimization problem. While Zhou et. al. detail a similar approach[118], we observe that in practice (Section 4.4), their recovered color models suffer from specularities in RGB images and often contain ghosting and smoothed away textures. Although Hernandez et. al.[20] provide a remedy to eliminating specular highlighting artifacts, their method does poorly in recovering sharp textures.

Contributions. We demonstrate that incorporating 2D texture cues, vertex color smoothing, and texture-adaptive camera viewpoint selection into the optimization problem qualitatively ameliorates these problems. Ultimately, we demonstrate that our method produces qualitatively more coherent color models than those produced by competing methods. Results from a user study with 133 participants indicate that our method outperforms competing approaches, advancing the state of the art.

4.2 Problem Formulation

We take in as input a 3D mesh $\mathbf{M}$, whose representation is a collection of triangles in 3D space. Mesh $\mathbf{M}$ consists of a vertex set $\mathbf{P}$, where each $\mathbf{p} \in \mathbf{P}$ neighbors vertices $N(\mathbf{p}) \subset \mathbf{P}$. We additionally take in a collection of RGB images, $\{I_i\}$, that observe the original object; each image has an associated intrinsics matrix $\mathbf{K}_i \in \mathbb{R}^{3\times 3}$ and initial extrinsics matrix $\mathbf{T}_i^0 \in SE(3)$. For each $\mathbf{p} \in \mathbf{P}$, we wish to estimate $\vec{C}(\mathbf{p})$, vertex $\mathbf{p}$'s color.[*] The collection of vertex colors, which we denote $\mathbf{C} = \{\vec{C}(\mathbf{p})\}$, constitutes the color model we aim to learn.

We estimate the color model $\mathbf{C}$ by minimizing a non-linear least squares objective that refines the original camera extrinsics $\mathbf{T}^0 = \{\mathbf{T}_i^0\}$ so as to maximize each vertex's color agreement. Concretely, denote $V(\mathbf{p}) \subset \{I_i\}$ as the subset of images that observe vertex $\mathbf{p}$ without occlusion, $\mathbf{T}_i$ as the updated extrinsics matrix for image I_i, and $\vec{\Gamma}_i(\mathbf{p}, \mathbf{T}_i)$ as the color obtained by projecting $\mathbf{p}$ onto image I_i using extrinsics $\mathbf{T}_i$ and intrinsics $\mathbf{K}_i$. For each $I_i \in V(\mathbf{p})$, we would like the error residual $\|\vec{C}(\mathbf{p}) - \vec{\Gamma}_i(\mathbf{p}, \mathbf{T}_i)\|^2$ to be small. This reasoning suggests that we minimize the following objective:

$$\mathcal{J}(\mathbf{C}, \mathbf{T}) = \frac{1}{2} \sum_{\mathbf{p}\in\mathbf{P}} \sum_{I_i \in V(\mathbf{p})} \|\vec{C}(\mathbf{p}) - \vec{\Gamma}_i(\mathbf{p}, \mathbf{T}_i)\|^2 \tag{4.1}$$

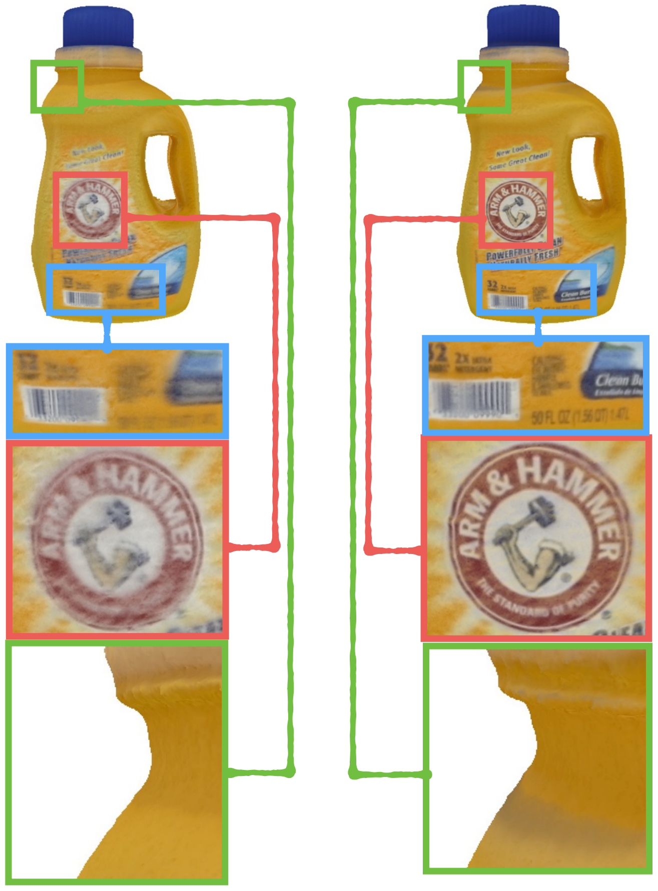
where $\mathbf{T} = \{\mathbf{T}_i\}$. The only variables to minimize are $\mathbf{C}$ and $\mathbf{T}$. We compute $\vec{\Gamma}_i(\mathbf{p}, \mathbf{T}_i)$ by composing extrinsics matrix $\mathbf{T}_i$, (fixed) intrinsics matrix $\mathbf{K}_i$, and a color evaluation from I_i, which can be written as $\vec{\Gamma}_i(\mathbf{u}_i(\mathbf{g}(\mathbf{p}, \mathbf{T}_i)))$. The functions $\mathbf{g}$ and $\mathbf{u}$ are defined as:

$$\mathbf{g}(\mathbf{p}, \mathbf{T}_i) = \mathbf{T}_i \mathbf{p} \tag{4.2}$$

$$\mathbf{u}([g_x, g_y, g_z, g_w]^T) = (c_x + g_x f_x / g_z, c_y + g_y f_y / g_z) \tag{4.3}$$

where f_x, f_y are the focal lengths of I_i and (c_x, c_y) denotes the principal point of I_i; we obtain these values from $\mathbf{K}_i$. The function $\vec{\Gamma}_i([u_x, u_y]^T)$ computes the bilinearly interpolated intensity at coordinates (u_x, u_y) in I_i. In the following sections, we explore objective variants that successively improve reconstructed color models qualitatively. We delve into concrete optimization details in Section 4.3.

[*]All color vectors in this chapter are 4-vectors whose entries all lie between 0 and 1. The first entry is a grayscale intensity while the second through fourth are scaled RGB intensities.



(a) Iteration 0 (b) Iteration 200

Figure 4.2: Mesh coloring (a), before, and (b), after, optimization using our implementation of the approach employed by Zhou et. al.[118]. Although this method improves texture coherence, the textures are still faded (particularly the Arm & Hammer logo). Further, solid yellow regions are still blotchy.

4.2.1 Accounting for Color Constancy and Specularities

In practice, specularities and the non-Lambertian nature of objects prevent us from achieving perfect color agreement, even after refining $\mathbf{T}$. Equation (4.1) asks for $\vec{C}(\mathbf{p})$ to agree with projected colors $\vec{\Gamma_i}(\mathbf{p}, \mathbf{T}_i)$ for all views $I_i \in V(\mathbf{p})$: as such, recovered colors in regions with white specularities may be heavily faded away, because residuals corresponding to images with high specularities will draw $\vec{C}(\mathbf{p})$ towards white. Indeed, we observe this behavior empirically. Zhou et. al. minimize a similar objective to Equation (4.1)[118].[†] As an example, applying this optimization problem to the Arm & Hammer detergent bottle drawn from the BigBIRD dataset[90] (692K vertices, 600

[†]The primary difference is that during optimization, Zhou et. al. set color vectors to only contain grayscale values and set final RGB colors by computing a weighted average per color

camera views), Figure 4.2 reveals that although the textures become clearer, they remain faded.

Rather than asking for color agreement from all views $V(\mathbf{p})$ for vertex $\mathbf{p}$, we consider selecting a subset. Images that present the most accurate colors for $\mathbf{p}$ typically have the most head-on, frontal views: i.e., the cosine of the angle subtending I_i's optical axis and $\mathbf{p}$'s normal is close to -1 (the optical axis and normal vectors point in opposite directions). We consider sorting $V(\mathbf{p})$ by this "foreshortening value" and retaining at most the top N images per $\mathbf{p}$. Figure 4.3 summarizes the results for N ranging from 1 to 600 (total number of views) for the Arm & Hammer detergent bottle.

Interestingly, we discover that N offers a tradeoff between sharp textures and smooth non-textured regions. Shown in Figure 4.3, smaller N leads to rough colors on the handle while larger N smooths out this noise. For example, when $N = 1$, we observe white regions on the detergent handle because the most frontal views incorrectly bleed onto the white background in the best viewing image (see Figure 4.1 for sample viewing images); larger N rectify this problem. Across many objects, we found that setting N beyond 30 does not produce visibly smoother non-textured regions. As originally conjectured, smaller N leads to crisper textures while larger N leads to faded, washed out textures. Looking carefully at $N = 1$ reveals noisy boundaries for the Arm & Hammer logo while $N = 10$ provides cleaner results by averaging away this noise.

We wish to take the best of both worlds by setting N's value depending on whether $\mathbf{p}$ is considered "textured." To do this, we consider assigning a label $t_{\mathbf{p}}$ to each $\mathbf{p}$, where $t_{\mathbf{p}} = 1$ when $\mathbf{p}$ is textured and 0 otherwise. With this, we employ the objective:

$$\mathcal{J}(\mathbf{C}, \mathbf{T}) = \frac{1}{2} \sum_{\mathbf{p} \in \mathbf{P}} \sum_{I_i \in V'(\mathbf{p}; t_{\mathbf{p}})} \|\vec{C}(\mathbf{p}) - \vec{\Gamma}_i(\mathbf{p}, \mathbf{T}_i)\|^2 \tag{4.4}$$

where $V'(\mathbf{p}; t_{\mathbf{p}})$ retains the top $N = 30$ views when $t_{\mathbf{p}} = 1$ and the top $N = 10$ views otherwise. Although interpolation between $N = 10, 30$ based on $t_{\mathbf{p}}$ is possible, we found that this produced less smooth color models. We discuss how to compute $t_{\mathbf{p}}$ in Section 4.3.4; until then, for easier exposition, we assume that these labels have already been computed.

4.2.2 Smoothing Speckled Regions

Although adapting $N = |V'(\mathbf{p}, t_{\mathbf{p}})|$ alleviates faded textures, we expect boundary artifacting due to the difference in the number of cameras employed in adjacent textured and untextured regions. Figure 4.4(a) (please use digital zoom in a PDF reader to view details) shows the colored detergent after optimizing Equation (4.4); the red dotted box reveals anticipated artifacts, while the blue and green boxes reveal slightly blotchy textures that still remain.

channel, where the weight for I_i is the cosine of the angle subtending I_i's optical axis and $\mathbf{p}$'s normal.

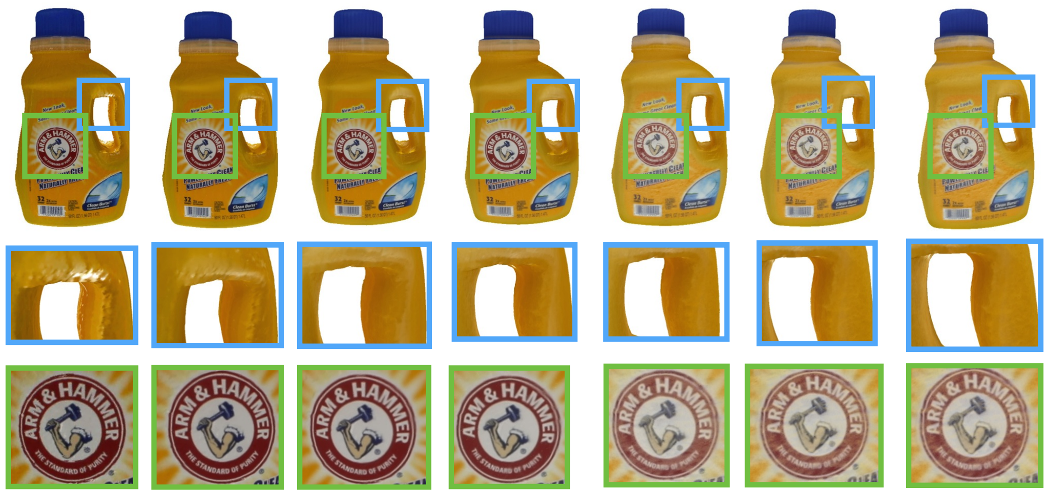
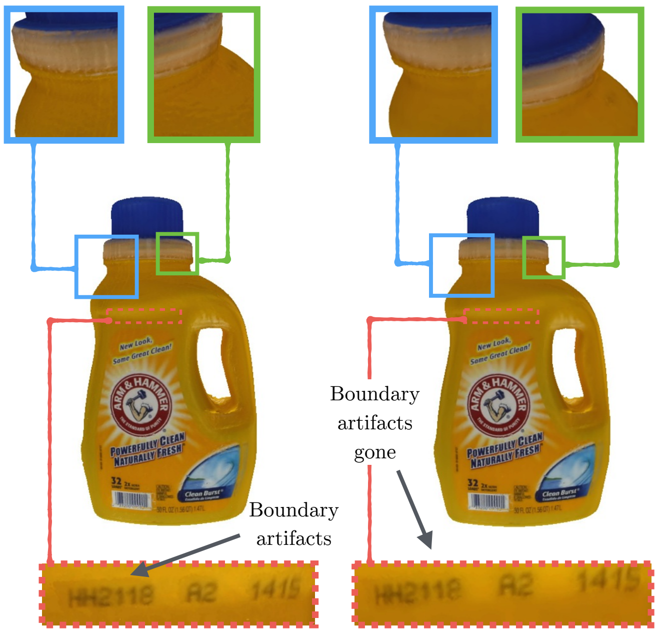


Figure 4.3: The effect of $N = |V'(\mathbf{p}; t_{\mathbf{p}})|$ on the sharpness of textured regions and the smoothness of non-textured regions: smaller (larger) N yield sharper (faded) textures, but blotchy (smooth) non-textured regions. When $N = 1$, we observe white regions on the detergent handle because these colors incorrectly bleed onto the white turntable background in the best viewing image; larger N rectify this problem.

We ameliorate both problems by encouraging color agreement between pairs of vertices lying along edges of $\mathbf{M}$. We need to be careful to not blur away sharp textures, however: so, we only smooth edges where (1) both vertices are non-textured or (2) exactly one vertex is textured. The first case allows us to eliminate blotchiness while the second smoothly connects non-textured and textured vertices. Concretely, we consider optimizing the new objective

$$\begin{aligned}\mathcal{J}(\mathbf{C}, \mathbf{T}) = \frac{1}{2} \sum_{\mathbf{p} \in \mathbf{P}} \sum_{I_i \in V'(\mathbf{p}; t_{\mathbf{p}})} \|\vec{C}(\mathbf{p}) - \vec{\Gamma_i}(\mathbf{p}, \mathbf{T}_i)\|^2 + \\ \frac{\lambda}{2} \sum_{\mathbf{p} \in \mathbf{P}} \sum_{\mathbf{p}' \in N(\mathbf{p})} (1 - t_{\mathbf{p}} t_{\mathbf{p}'}) \cdot \|\vec{C}(\mathbf{p}) - \vec{C}(\mathbf{p}')\|^2\end{aligned} \tag{4.5}$$

where λ is a hyperparameter that allows us to trade off the contribution between the smoothing and original color-agreement terms. Figure 4.4(b) shows the detergent bottle after incorporating smoothing – shown in the blue and green boxes, non-textured regions are much smoother and the boundary artifacts have disappeared. Because we smooth only untextured regions and "texture to non-texture" boundaries, the detergent bottle's serial number is not smoothed away during optimization.



(a) No smoothing, $\lambda = 0$ (b) Smoothing, $\lambda = 10$

Figure 4.4: Detergent bottle's color model, optimized using Equation (4.4). Without smoothing, solid regions remain blotchy. Also, the difference in the number of cameras employed in textured and untextured regions leads to texture artifacts, as shown by the patchy "HH2118," "A2," and "1415" in (a). Optimization using Equation (4.5), which includes smoothing, eliminates blotchy regions while preserving textured regions; it further removes the patchy boundaries between textured and non-textured regions. Please use digital zoom to view details.

4.3 Optimization

We now discuss optimization algorithms to minimize the objective discussed in Section 4.2.2. First, we discuss a naive optimization method based on the Gauss-Newton algorithm. Demonstrating that this is computationally intractable, we introduce an alternating optimization method which tractably minimizes the same objective. Section 4.3.3 concludes our discussion on optimization by discussing several practical improvements including (1) fixes to poor Hessian conditioning and (2) accelerating learning via multiscale optimization.

4.3.1 Gauss-Newton Optimization

Since $\mathcal{J}(\mathbf{C}, \mathbf{T})$ is a non-linear least squares objective, we consider using the Gauss-Newton method for minimization. Equation (4.5) features two types of residuals:

$$\vec{r}^{(1)}_{i,\mathbf{p}} = \vec{C}(\mathbf{p}) - \vec{\Gamma}_i(\mathbf{p}, \mathbf{T}_i) \tag{4.6}$$

$$\vec{r}^{(2)}_{\mathbf{p},\mathbf{p}'} = \vec{C}(\mathbf{p}) - \vec{C}(\mathbf{p}') \tag{4.7}$$

Let $\mathbf{C}^k$ and $\mathbf{T}^k$ denote the values of $\mathbf{C}$ and $\mathbf{T}$ at iteration k, and $\mathbf{x}^k = [\mathbf{C}^k, \mathbf{T}^k]$. We initialize the optimization with $\mathbf{x}^0 = [\mathbf{C}^0, \mathbf{T}^0]$ where (1) $\mathbf{T}^0$ is set to the provided initial calibrated extrinsics and (2) $\mathbf{C}^0(\mathbf{p})$ is set to the average of $\{\vec{\Gamma}_i(\mathbf{p}, \mathbf{T}^0_i)\}_{I_i \in V'(\mathbf{p};t_{\mathbf{p}})}$. The Gauss-Newton procedure prescribes taking steps $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x}^k$ where we solve for $\Delta\mathbf{x}^k$ in the following linear system:

$$\mathbf{J}^T\mathbf{J}\Delta\mathbf{x}^k = -\mathbf{J}^T\mathbf{r} \tag{4.8}$$

where $\mathbf{r} = [\mathbf{r}^{(1)}, \mathbf{r}^{(2)}]$ is the residual vector and $\mathbf{J} = [J_{r^{(1)}}, J_{r^{(2)}}]$ is the Jacobian of $\mathbf{r}$, both evaluated at $\mathbf{x}^k$:

$$\mathbf{r}^{(1)} = [\vec{r}^{(1)}_{i,\mathbf{p}}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^k}]_{(i,\mathbf{p})} \tag{4.9}$$

$$\mathbf{r}^{(2)} = [\vec{r}^{(2)}_{i,\mathbf{p}}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^k}]_{(i,\mathbf{p})} \tag{4.10}$$

$$\mathbf{J}_{r^{(1)}} = [\nabla\vec{r}^{(1)}_{i,\mathbf{p}}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^k}]_{(i,\mathbf{p})} \tag{4.11}$$

$$\mathbf{J}_{r^{(2)}} = [\nabla\vec{r}^{(2)}_{i,\mathbf{p}}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^k}]_{(i,\mathbf{p})} \tag{4.12}$$

We notice that $\mathbf{J}$ has a number of rows and columns that are both linear in $|\mathbf{P}|$; this renders solving Equation (4.8) intractable, since we typically operate on meshes with 100K+ vertices.

4.3.2 Alternating Optimization

We consider optimizing $\mathcal{J}(\mathbf{C}, \mathbf{T})$ by alternating between minimizing $\mathbf{C}$ and $\mathbf{T}$. First, we discuss how to minimize the objective with respect to $\mathbf{C}$. We initialize the optimization method with $\mathbf{x}^0 = [\mathbf{C}^0, \mathbf{T}^0]$, computed as in Section 4.3.1.

Optimizing $\mathbf{C}$. In minimizing $\mathcal{J}(\mathbf{C}, \mathbf{T})$ with respect to $\mathbf{C}$ after fixing $\mathbf{T}$, we are left with minimizing a quadratic objective. There exist many approaches to do this – we found that employing gradient descent with momentum and the adaptive learning rate method described in[48] offers a good tradeoff between speed and accuracy. Computing each gradient takes time linear in the number of edges in the mesh:

$$\nabla_{\vec{C}(\mathbf{p})}\mathcal{J}(\mathbf{C};\mathbf{T}) = \sum_{I_i \in V'(\mathbf{p};\, t_{\mathbf{p}})} [\vec{C}(\mathbf{p}) - \vec{\Gamma}_i(\mathbf{p},\mathbf{T}_i)] + \tag{4.13}$$

$$\lambda \sum_{\mathbf{p}' \in N(\mathbf{p})} (1 - t_{\mathbf{p}} t_{\mathbf{p}'}) \cdot [\vec{C}(\mathbf{p}) - \vec{C}(\mathbf{p}')] \tag{4.14}$$

Optimizing $\mathbf{T}$. We minimize $\mathcal{J}(\mathbf{C},\mathbf{T})$ with respect to $\mathbf{T}$ after fixing $\mathbf{C}$ via Gauss-Newton. Ignoring terms that do not depend on $\mathbf{T}$, we rewrite Equation (4.5) as:

$$\mathcal{J}(\mathbf{C},\mathbf{T}) = \frac{1}{2}\sum_{I_i} \sum_{\{\mathbf{p}: I_i \in V'(\mathbf{p};\, t_{\mathbf{p}})\}} \|\vec{r}^{(1)}_{i,\mathbf{p}}\|^2 \tag{4.15}$$

Decomposing this sum of squares across all I_i, we can now compute separate Gauss-Newton updates for each image I_i, since $r_{j,\mathbf{p}}$ does not depend on $\mathbf{T}_j$ for $i \neq j$. We now discuss how to perform updates for a single image I_i. Defining $\mathbf{x}^k = [\mathbf{C}^k, \mathbf{T}^k]$ where $\mathbf{C}^k$ is fixed, we compute $\mathbf{J}$ and $\mathbf{r}$ as follows:

$$\mathbf{r} = [\vec{r}^{(1)}_{i,\mathbf{p}}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^k}]_{(i,\mathbf{p})} \tag{4.16}$$

$$\mathbf{J} = [\nabla \vec{r}^{(1)}_{i,\mathbf{p}}(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^k}]_{(i,\mathbf{p})} \tag{4.17}$$

We compute $\mathbf{r}$ using Equation (4.6). Computing $\mathbf{J}$ entails computing the partial derivatives of each entry in $\vec{r}_{i,\mathbf{p}}$ with respect to $\mathbf{T}$. For notational simplicity, let $r^{(1)}_{i,\mathbf{p}}$ denote the first entry of $\vec{r}^{(1)}_{i,\mathbf{p}}$ and $\Gamma_i(\mathbf{p},\mathbf{T}_i)$ denote the first entry of $\vec{\Gamma}_i(\mathbf{p},\mathbf{T}_i)$. We parameterize $\mathbf{T}_i$ by locally linearizing around $\mathbf{T}^k_i$; specifically, letting $\xi_i = (\alpha_i, \beta_i, \gamma_i, a_i, b_i, c_i)^T$ represent an incremental transform[‡], we set:

$$\mathbf{T}_i \approx \begin{pmatrix} 1 & -\gamma_i & \beta_i & a_i \\ \gamma_i & 1 & -\alpha_i & b_i \\ -\beta_i & \alpha_i & 1 & c_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{T}^k_i \tag{4.18}$$

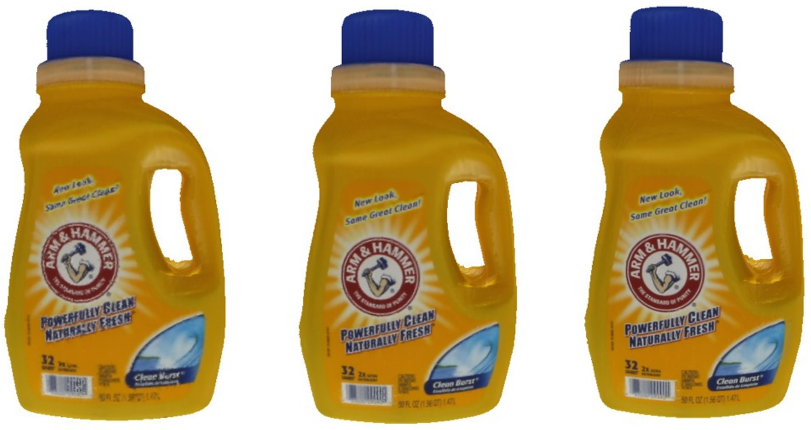
We have that:

$$\nabla_{\mathbf{T}_i} r^{(1)}_{i,\mathbf{p}} = -\frac{\partial}{\partial \xi_i}(\Gamma_i(\mathbf{p},\mathbf{T}_i)) = -\frac{\partial}{\partial \xi_i}(\Gamma_i(\mathbf{u}_i(\mathbf{g}(\mathbf{p},\mathbf{T}_i)))) \tag{4.19}$$

$$= -\nabla\Gamma_i(\mathbf{u})\mathbf{J}_{\mathbf{u}}(\mathbf{g})\mathbf{J}_{\mathbf{g}}(\xi_i)|_{\mathbf{x}=\mathbf{x}^k} \tag{4.20}$$

We use Equation (4.18) to compute $\mathbf{J}_{\mathbf{g}}(\xi_i)$ and Equation (4.2) to compute $\mathbf{J}_{\mathbf{u}}(\mathbf{g})$. We evaluate $\nabla\Gamma_i(\mathbf{u})$ numerically: recall that we compute $\Gamma_i(\mathbf{u})$ via bilinear interpolation, so gradients are valid when $\mathbf{u}$ lies within I_i. After solving for $\Delta\mathbf{x}^k$, we map the resulting

[‡]We use the bundle adjustment technique discussed in our previous work for initialization[90], so initializations of $\mathbf{T}_i$ are close to optimal – as such, incremental transforms are valid.



(a) Level 1, optimized (b) Level 2, optimized (b) Level 3, optimized

Figure 4.5: Multiscale optimization: level 0 optimizes a Lindstrom-Turk-decimated mesh, level 1 optimizes the original mesh, and level 2 optimizes a $\sqrt{3}$-subdivided mesh.

ξ_i back into $SE(3)$ and compute $\mathbf{T}_i^{k+1}$ via this update. By employing an alternating optimization strategy, optimizing all $\mathbf{T}$ reduces to solving a total of $|\{I_i\}|$ linear systems with 6 variables each, which we perform in parallel; Zhou et. al. employ a similar method in camera pose optimization[118].

4.3.3 Coarse-to-Fine Levenberg-Marquardt Optimization

In making updates to each $\mathbf{T}_i$, the Hessian $\mathbf{J}^T\mathbf{J}$ may be poorly conditioned, leading to updates that cause some mesh vertices $\mathbf{p}$ to project outside the bounds of an image I_i. As a remedy, we employ damped Hessians $\mathbf{J}^T\mathbf{J} + \eta\mathbf{I}$ during optimization (a.k.a. Levenberg-Marquardt optimization). In updating a single $\mathbf{T}_i$, we first initialize η to 0. Upon making an update, we project all $\{\mathbf{p}|I_i \in V'(\mathbf{p}; t_{\mathbf{p}})\}$ onto I_i; if any vertices fall outside I_i, we increase η to 0.001. We repeat this projection check and continue increasing η by a factor of 1.1 until all vertices fall within I_i; if we have not found a satisfactory update after 5 such trials, we do not update $\mathbf{T}_i$ during the current iteration.

In practice, we find that resolving small texture features such as text requires us to increase the density of vertices in $\mathbf{M}$ before optimization. As such, after reconstructing $\mathbf{M}$ using Narayan et. al.'s method[76], we employ $\sqrt{3}$-subdivision[56] without smoothing to increase the mesh's surface resolution while not altering $\mathbf{M}$'s geometry. We then apply the Levenberg-Marquardt procedure described above.

We empirically accelerate convergence without sacrificing solution quality by employing a coarse-to-fine optimization scheme. Rather than immediately use $\mathbf{M}$ in optimization, consider constructing the series of meshes: $\mathbf{M}_0, \mathbf{M}_1, \mathbf{M}_2$. Here, $\mathbf{M}_1$ is the original unaltered mesh obtained from[76] and $\mathbf{M}_2$ is obtained by a single application of $\sqrt{3}$-subdivision. In the other direction, we apply Lindstrom-Turk polygon simplifica-

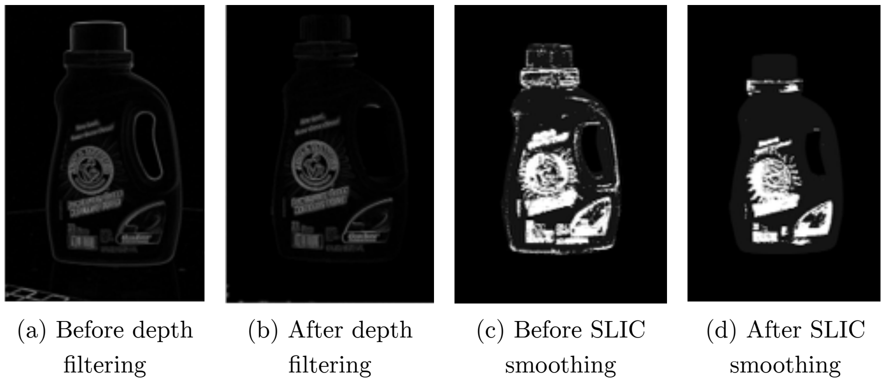

(a) Before depth filtering (b) After depth filtering (c) Before SLIC smoothing (d) After SLIC smoothing

Figure 4.6: (a) shows a sample response map for a single image after applying the technique in Section 4.3.4, before depth discontinuity filtering; white regions denote higher responses. (b) shows the updated map after depth discontinuity filtering. (c) visualizes all $t_{\mathbf{p}}$, without SLIC smoothing. (d) visualizes the updated $t_{\mathbf{p}}$ after SLIC smoothing.

tion[68] to $\mathbf{M}_0$: specifically, $\mathbf{M}_{-1}$ has at most 50% the number of edges in $\mathbf{M}_0$. We use implementations for both $\sqrt{3}$-subdivision and Lindstrom-Turk polygon simplification provided in the open source Computational Geometry Algorithms Library (CGAL)[cga].

We proceed by running Levenberg-Marquardt optimization on $\mathbf{M}_0$. Upon convergence, we initialize a new optimization problem on $\mathbf{M}_1$; because the vertices have changed, we re-initialize $\mathbf{C}$. However, we warm-start this new optimization problem using the converged $\mathbf{T}$ from $\mathbf{M}_0$. We repeat this up to $\mathbf{M}_2$. Figure 4.5 visualizes optimization progress. Multiscale optimization typically yields speedups of $2 - 3\times$ over directly optimizing over $\mathbf{M}_2$ (larger speedups for larger meshes); we did not find any noticeable differences between color models produced with and without multiscale optimization.

4.3.4 Texture Label Assignment

Before presenting empirical results, we discuss how to compute texture labels $t_{\mathbf{p}}$ for $\mathbf{p} \in \mathbf{P}$. Algorithm 2provides the details. We first turn the camera images $\{I_i\}$ into grayscale images $\{I_i^{(gray)}\}$ via the luminosity method. We convolve each $I_i^{(gray)}$ with 10 kernels of size 10×10, whose entries are uniformly sampled from $[-1, 1]$ and sum to 0. Taking the element-wise-maximum over these filtered images then gives us a response map, where higher responses correspond to textured regions in the image (see Figure 4.6(a)). We let this resulting response map be I_i'. Convolving with random filters is known to reveal high-frequency spatial patterns in images, e.g., edges[85]; as such, examining the maximum responses from an ensemble of such filters typically yields regions of high spatial frequency, i.e., textured regions.

As marked in Figure 4.6(a), depth-discontinuities can trigger high responses in the response map near object boundaries, which are not necessarily textured. To combat this, we (1) compute a z-buffered depth map using the intrinsics $\mathbf{K}_i$ and extrinsics $\mathbf{T}_i$

User Study Summary ($n = 133$): Which Method Matches the Reference Most Closely?

	O1	O2	O3	O4	O5	O6	O7	O8
Our Method	0.872	0.841	0.882	0.985	0.587	0.735	0.855	0.655
Zhou et. al.[118]	0.005	0.019	0.000	0.008	0.049	0.027	0.021	0.031
Hernandez et. al.[43]	0.106	0.111	0.109	0.008	0.320	0.239	0.119	0.304
Volumetric blending[80]	0.018	0.029	0.009	0.000	0.044	0.000	0.004	0.010

	O9	O10	O11	O12	O13	O14	O15	O16
Our Method	0.802	0.774	0.859	0.925	0.491	0.894	0.581	0.917
Zhou et. al.[118]	0.054	0.117	0.080	0.016	0.019	0.033	0.018	0.004
Hernandez et. al.[43]	0.126	0.100	0.044	0.055	0.472	0.065	0.353	0.063
Volumetric blending[80]	0.018	0.009	0.016	0.004	0.019	0.008	0.048	0.016

Table 4.1: O1 through O16 represent objects in Figure 4.7: 3m_high_tack_spray_adhesive is O1, windex is O2, etc.

matrices associated with each image I_i and (2) compute a depth discontinuity map; a depth value is considered to be a discontinuity if, within a centered square window of 9 pixels, there is a difference in depth of more than 1 cm. We zero out entries in I'_i that are at a depth discontinuity. To average away sensor noise per image I_i, we (1) compute SLIC superpixels[4] of I_i and (2) set the value of each pixel in I_i to the average of all values in the superpixel that the pixel lies in. We apply a linear transform to ensure that entries of I'_i lie between 0 and 1. Figures 4.6c, d show the values of $t_{\mathbf{p}}$ with and without this noise-reduction step. The smoother consistency of Figure 4.6 yields color models with fewer blotchy regions.

Computing $t_{\mathbf{p}}$ entails projecting $\mathbf{p}$ onto all images where $\mathbf{p}$ is visible; let $\mathbf{P}_i \subset \mathbf{P}$ denote the subset of vertices which is visible in image I_i. We efficiently compute each $\mathbf{P}_i$ using a z-buffer technique; additionally, vertices within 9 pixels of a depth discontinuity are discarded from $\mathbf{P}_i$. We proceed by projecting $\mathbf{p}$ onto each response map I'_i for which $\mathbf{p} \in \mathbf{P}_i$, accumulating the lookup values into a list. We set $t_{\mathbf{p}}$ to a weighted mean of this list of values. The weight for image I_i is the absolute value of the cosine of the angle subtending image I_i's optical axis and vertex $\mathbf{p}$'s normal. We finally assign $t_{\mathbf{p}}$ to 0 or 1 by simply rounding its value.

4.4 Experiments

We conduct experiments using objects drawn from the BigBIRD dataset, discussed in Chapter 1. All experiments are run on an Intel i7-4930K with 64 GB of memory. We provide timing information on the source code page (see abstract); in general, the optimization process typically takes 1-5 minutes per object, depending on the number of object vertices.

We display 64 reconstructed color models in Figure 4.7: 16 objects $\times 4$ methods we consider: (1) ours, (2) Zhou et. al's[118], (3) Hernandez et. al.'s[43], and (4) PCL's volumetric blending[80]. We do not employ the deformation grid optimization that Zhou et. al. discuss, as the resulting LM updates are not small, leading to divergence.

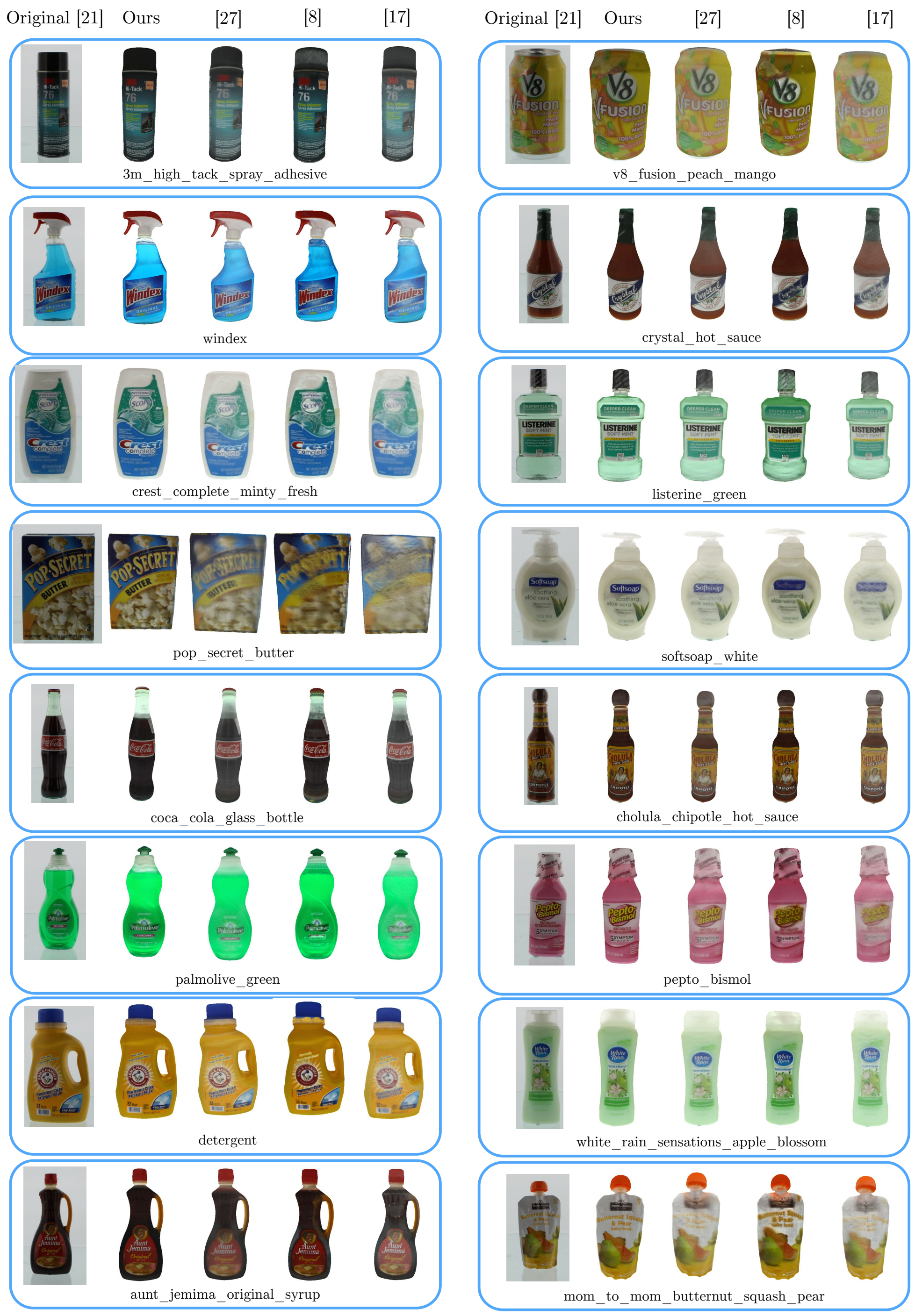


Figure 4.7: Juxtapositions of original BigBIRD images[90] with color model reconstructions from (1) our method, (2) Zhou et. al.[118] without deformation grid optimization, (3) Hernandez et. al.[43], and (4) PCL's volumetric blending[80]. Note that we do not employ Zhou et. al.'s deformation grid optimization, since this leads to divergence. Please use a PDF reader's digital zoom to view details.

4.4.1 Evaluation Methodology

To quantitatively compare our method with competing approaches, we conducted an online user survey (https://goo.gl/forms/Feo2OqOdw4), where each participant was given 16 multiple choice questions, one per object in Figure 4.7. Each question asked the participant: “Which of the following images matches ‘Reference’ most closely?” The reference image displayed the original BigBIRD image; we juxtaposed the reference image with color models estimated from our method and competing methods. We randomized the order in which the color models appeared within a question. Participants were allowed to answer with a tie, electing two methods as the best for a question. Participants were given as much time as needed to complete the 16 questions; we advertised the survey primarily via department emails. There were a total of 133 participants in the survey.

In creating Table 4.1, for each question asked per participant, we assigned 1 point for a single elected method and 0.5 point to two elected methods when the participant chose a tie. To ensure that we did not receive overwhelming amounts of spurious data, we included two objects whose color models were difficult to distinguish in quality – O13 (cholula_chipotle_hot_sauce) and O15 (white_rain_sensations_apple_blossom); in the former case, our method was voted the best by 49.1% of participants while Hernandez et. al.’s method received a close 47.2%. In the latter case, our method received 58.1% while Hernandez et. al.’s received 47.2%. Most of the other color models had fairly distinguishing features.

4.4.2 Analysis

Our method received the highest votes in all objects we considered, in most cases, by more than 30%, and often by more than 75%. Because Zhou et. al’s method averages in all images that view a vertex[118], it provides ghosted, faded textures. Examples include front faces for pop_secret_butter and mom_to_mom_butternut_squash_pear as well as caps for pepto_bismol and listerine_green. Other objects with Zhou et. al. are often faded, as seen with the 3m_high_tack_spray_adhesive and v8_fusion_peach_mango. PCL’s volumetric blending[80] suffers from similar problems, although in most cases, ghosting is more severe.

Per vertex, Hernandez et. al.’s approach averages the top 3 camera views that have the highest saturation on an HSV scale; they do not jointly optimize camera poses[43]. This model can produce vibrant, but highly distorted color models. Examples include front faces for pop_secret_butter, detergent, crystal_hot_sauce, and palmolive_green and caps for listerine_green, pepto_bismol, and mom_to_mom_butternut_squash_pear.

Despite the highly specular nature of our objects, our approach can reconstruct very fine textural details. In pepto_bismol, we clearly see the “digestive relief” text on the cap. In coca_cola_glass_bottle, we observe that Coca Cola created the bottle on 15 Sep 14 at 7:40 AM.[§] In listerine_green, the white plastic holding down the cap as well

[§]To clearly see such tiny details, we recommend visiting the online survey: https://goo.gl/forms/Feo2OqOdw4 is crisp, as is the text “listerine” on the black cap.

Our method is not perfect. v8_fusion_peach_mango reveals blotchiness above the V8 symbol. palmolive_green shows ghosting around the green logo. detergent’s barcode is not perfectly crisp and has few splotches of yellow.

4.5 Conclusion

We recover high-quality color models from the BigBIRD dataset by jointly optimizing a non-linear least squares objective over camera poses and a mesh color model. We incorporate 2D texture cues, vertex color smoothing, and texture-adaptive camera viewpoint selection into the objective, which allows us to outperform competing methods. We also discuss strategies to accelerate optimization speeds.

Algorithm 2 Compute Texture Intensities

```
$\mathbf{M} \leftarrow$ input mesh, with vertex set $\mathbf{P}$
$\{I_i^{(gray)}, \mathbf{K}_i, \mathbf{T}_i\} \leftarrow$ calibrated grayscale images $I$, associated intrinsics $\mathbf{K}$, and extrinsics matrices $\mathbf{T}$
$F \leftarrow$ a list of 10 kernels of size $10 \times 10$, whose entries are uniformly sampled from $[-1, 1]$ and sum to 0
$I' \leftarrow \{\}$
for $I_i^{(gray)} \in \{I_i^{(gray)}\}$, in parallel, do
    $I'_i \leftarrow$ matrix of zeros, with size of $I_i$
    for $f \in F$ do
        $C \leftarrow$ convolve $f$ with $I_i^{(gray)}$
        $I'_i \leftarrow$ element-wise-max of $I'_i$ and $C$
    end for
    $Z \leftarrow$ depth map for $I_i^{(gray)}$, computed with $\mathbf{M}, \mathbf{K}_i, \mathbf{T}_i$.
    $D \leftarrow$ depth discontinuity map computed from $Z$ (see Section 4.3.4)
    $I'_i \leftarrow$ zero $I'_i$ where depth discontinuities exist in $D$.
    $I'_i \leftarrow (I'_i - \min(I'_i))/(\max(I'_i) - \min(I'_i))$
    $S \leftarrow$ compute SLIC superpixels for $I_i$
    for $s \in S$ do
        $p \leftarrow$ average value of $I'_i$ in superpixel $s$
        Replace all values in $I'_i$ corresponding to pixels in $s$ with value $p$
    end for
    $I'[i] \leftarrow I'_i$
end for
$t \leftarrow \{\}$
for $\mathbf{p} \in \mathbf{P}$, in parallel, do
    $v \leftarrow$ list of values obtained by projecting $\mathbf{p}$ onto each $I'_i$ where $\mathbf{p}$ is visible in $I'_i$
    $t_\mathbf{p} \leftarrow$ weighted mean of values in $v$ (see Section 4.3.4 for weights)
    $t_\mathbf{p} \leftarrow 0$ if $t_\mathbf{p} \leq 0.5$, otherwise 1
end for
return $t$
```

5 Alpha-Beta Divergences Discover Micro and Macro Structures in Data

Our exposition so far has examined how to extract a 3D color mesh from a "real-world" object, by (1) first detailing the construction and calibration of a 3D scanner, (2) next discussing techniques to extract a 3D mesh from the object, and (3) finally providing algorithms to color the resulting 3D mesh. The mechanics of the scanner arguably allow us to amass a vast collection of 3D objects; indeed, in addition to the objects present in the moderately-sized public datasets which have made use of the setup detailed in this thesis[90,18,89], we have collected over 1500 other objects (not publicly released).

As with any growing dataset, the ability to holistically view dataset characteristics becomes an evergrowing need. In particular, useful visualizations of a dataset's instances can provide yield invaluable intuition. Such intuitions are particularly difficult to discover in very large datasets, especially those with large dimensionality. Concretely, we consider visualizing datasets with anywhere between hundreds and tens of millions of data instances, where each instance has anywhere between ten and millions of features. Such scales are not (yet) common in 3D computer vision (such scales will arguably arise very soon in this field), but are very common in 2D computer vision as well as eclectic disciplines, ranging from physics to biology to music. This work specifically considers non-linear dimensionality reduction in holistic visualization. This work was published in ICML 2015[75].

5.1 Introduction and Related Work

Data visualization techniques aim to generate a low-dimensional representation of a dataset which data scientists and researchers can inspect to gain insight into the structure and complexity of the data. Vital to data-driven decision making, visualizations graphically represent latent structure and meaning in the dataset. Many recent approaches produce low-dimensional embeddings of data instances, which we can view via scatter plots[19,45,99,64,15,103,113]. Particularly, recent variants on stochastic neighborhood embedding (SNE) have become popular[45]; the recently published t-SNE (t-distributed stochastic neighbor embedding) is particularly popular[99]. At a high level, SNEs (i) capture neighborhood information from pairs of points in the original dataset with m data instances into an $m \times m$ data similarity matrix, and (ii) learn a low-dimensional embedding of those points whose similarity matrix closely matches the original. Computing an embedding which matches the structure of the original data involves minimizing a divergence measure, e.g., KL-divergence, between the data and embedding similarity matrices. Indeed, most methods employ the KL-divergence[45,99,100,111].

Unfortunately, most proposed methods either (1) are hyperparameter-free, giving researchers little room to directly communicate what types of patterns they are searching for in the data, or (2) have non-intuitive hyperparameters that require expensive, tedious grid searches.

Contributions. We propose a method featuring 2 hyperparameters (α, β). Our theoretical analysis predicts that (1) setting $\alpha + \beta < 1$ reveals macro-structures (categories, e.g., dogs), (2) $\alpha < 1$ reveals micro-structures (fine-grained classes, e.g. dalmations), and (3) $\alpha + \beta > 1$ reveals instances close to class boundaries (e.g., digits that are easily confused as 1 vs 7 or 4 vs 9) (Section 5.4). The meaning of the parameters makes data exploration intuitive, and can obviate the need for extensive grid searches over hyperparameter settings. We emphasize that these settings are dataset-agnostic, empirically substantiated on 10 datasets covering a wide swath of sizes (100 – 11M instances) drawn from a broad set of domains (computer vision, biology, particle physics). Our method allows for fast parallel CPU/GPU implementations; our GPU implementation runs 20× faster than the state of the art SNE-based implementations (Section 5.5).

Our theoretical analysis additionally answers a question recently posed in[15]: under what circumstances should we choose a particular divergence to minimize in the SNE framework, and what consequences does this choice have? While minimizing divergences other than the KL-divergence in the SNE objective has recently been explored computationally, these works do not answer this question. Yang et. al. demonstrate that several popular SNE variants arise from varying the divergence that is being minimized[112]. Yang et. al. show that a novel optimization equivalence theorem between α-divergences, β-divergences, and γ-divergences yields a class of methods that build on the best aspects of graph layout and vectorial embedding. Bunte et. al. apply t-SNE variants using several Bregman divergences, f-divergences, and γ-divergences to two small datasets (COIL-20[77] and the Olivetti face dataset[84])[15]. Although Bunte et. al. acknowledge that varying divergences produce different visualizations, they admit that they are unable to deliver an overall recipe for choosing a particular divergence in a given task. To the best of our knowledge, this work is the first to provide such a recipe.

Past work primarily explores minimizing purely a single divergence in the SNE framework. We discover that minimizing the generalized alpha-beta divergence (a.k.a. AB-divergence)[23], which blends the α- and β- divergences, is crucial to discovering important micro and macro structures in the data (Section 5.4).

5.2 Background: t-distributed Stochastic Neighborhood Embedding

Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m\}$, with data instances $\mathbf{x}_i \in \mathbb{R}^n$, t-distributed Stochastic Neighbor Embedding (t-SNE)[99] aims to learn an embedding $\mathcal{E} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_m\}$, where $\mathbf{y}_i \in \mathbb{R}^d$ (usually, $d = 2$ or 3). To achieve this goal, t-SNE defines

$$\mathbf{P}_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{ik}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}, \tag{5.1}$$

$$\mathbf{P}_{ij} = (\mathbf{P}_{i|j} + \mathbf{P}_{j|i})/2m, \tag{5.2}$$

$$\mathbf{Q}_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} \tag{5.3}$$

where additionally, $\mathbf{P}_{i|i} = \mathbf{Q}_{ii} = 0$. Determining the individual variances σ_i^2 involves running a binary search such that the perplexity (2 raised to the entropy) of the conditional $\mathbf{P}_{\cdot|j}$ equals k, a free parameter[100]. The embedding employs a Student-t kernel rather than a Gaussian to prevent embedding points from crowding near the center of the visualization map without clear clustering, a.k.a. the "crowding problem." t-SNE prescribes learning the embedding vectors $\mathbf{y}_i$ by running gradient descent to minimize the resulting non-convex KL-divergence, $\mathcal{J}(\mathcal{E}) = \sum_{i\neq j}\mathbf{P}_{ij}\log\mathbf{P}_{ij}/\mathbf{Q}_{ij}$:

$$\frac{\partial\mathcal{J}}{\partial\mathbf{y}_i} = 4\sum_{i\neq j} Z(\mathbf{y}_i - \mathbf{y}_j)(\mathbf{P}_{ij}\mathbf{Q}_{ij} - \mathbf{Q}_{ij}^2) \tag{5.4}$$

$$= \sum_{i\neq j}\underbrace{4\mathbf{P}_{ij}\mathbf{Q}_{ij}Z(\mathbf{y}_i - \mathbf{y}_j)}_{\text{Force 1}} - \sum_{i\neq j}\underbrace{4\mathbf{Q}_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)}_{\text{Force 2}} \tag{5.5}$$

where $Z = \sum_{k\neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}$. Naively computing this gradient takes $\mathcal{O}(m^2)$ time, making visualizations of greater than 50K data vectors prohibitively expensive.

Barnes-Hut-SNE (BHSNE)[100] approximates the t-SNE's gradient in sub-quadratic time, yielding nearly identical visualizations to t-SNE while allowing for visualizations of millions of data vectors in a few hours. To this end, BHSNE only retains $\mathbf{P}_{ij}$ where data vector $\mathbf{x}_j$ is one of $\mathbf{x}_i$'s closest $3k$ neighbors, and sets the rest to 0. In practice, BHSNE constructs a vantage point tree to execute all nearest neighbor searches in $\mathcal{O}(kmn\log m)$ time[100,114]. BHSNE computes Force 1 in $\mathcal{O}(km)$ time by adding only terms involving positive $\mathbf{P}_{ij}$ and computing each $\mathbf{Q}_{ij}Z = (1+\|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}$ in constant time; we ignore the dimensionality of the $\mathbf{y}_i$, as BHSNE typically seeks a 2D or 3D embedding. BHSNE employs a Barnes-Hut approximation algorithm to compute Force 2 in $\mathcal{O}(m\log m)$ time: given $\mathbf{y}_i, \mathbf{y}_j$, and $\mathbf{y}_k$ where $\|\mathbf{y}_i - \mathbf{y}_j\| \approx \|\mathbf{y}_i - \mathbf{y}_k\| \gg \|\mathbf{y}_j - \mathbf{y}_k\|$, the contributions of $\mathbf{y}_j$ and $\mathbf{y}_k$ to Force 2 will be roughly equal. The Barnes-Hut algorithm exploits this in computing the sum of all contributions to an embedding vector $\mathbf{y}_i$ by (i) constructing a quadtree over $\{\mathbf{y}_i\}$, (ii) traversing the quadtree via a depth-first-search, and (3) at every quadtree node, deciding whether the corresponding cell can summarize the gradient contributions for all points in that cell. In computing Force 2 for a point

$\mathbf{y}_i$, if a cell is sufficiently small and far away from $\mathbf{y}_i$, then $\mathbf{Q}_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)$ will be similar for all points $\mathbf{y}_j$ in that cell. As such, BHSNE approximates the total contribution as $N_{cell} \cdot \mathbf{Q}_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)$ where N_{cell} denotes the total number of points in the cell. To compute Z efficiently, BHSNE (i) runs a separate Barnes-Hut procedure to compute a $z_i = \sum_j K_q(\|\mathbf{y}_i - \mathbf{y}_j\|^2)$ for each embedding point i and (ii) sums over the z_i's to yield Z. BHSNE then uses this value of Z in the Barnes-Hut procedure to compute Force 2. BHSNE decides whether a cell can summarize the points that it contains by checking whether $\|\mathbf{y}_i - \mathbf{y}_{cell}\|^2 / r_{cell} < \theta$, where r_{cell} is the length of the cell diagonal, y_{cell} is the cell's center of mass, and θ is a threshold that trades off speed and accuracy (larger values lead to poorer approximations).

5.3 Alpha-Beta Stochastic Neighborhood Embedding

Alpha-Beta Stochastic Neighborhood Embedding (ABSNE), our proposed method, differs from t-SNE in that it minimizes the alpha-beta divergence (AB-divergence). We minimize the cost $\mathcal{J}_{\text{ABSNE}}(\mathcal{E}; \alpha, \beta) = D_{AB}^{\alpha\beta}(\mathbf{P}\|\mathbf{Q})$, computed as

$$\frac{1}{\alpha\beta} \sum_{i \neq j} \left(-\mathbf{P}_{ij}^{\alpha} \mathbf{Q}_{ij}^{\beta} + \frac{\alpha}{\alpha+\beta} \mathbf{P}_{ij}^{\alpha+\beta} + \frac{\beta}{\alpha+\beta} \mathbf{Q}_{ij}^{\alpha+\beta} \right), \tag{5.6}$$

where $\alpha \in \mathbb{R}\backslash\{0\}, \beta \in \mathbb{R}$ are hyperparameters. It is possible to set $\beta = 0$ or $\alpha + \beta = 0$ by extending the AB-divergence via continuity, e.g., by applying l'Hôpital's rule (see[23]); this does not affect the form of the gradient we present below (see Supplementary Materials). We use the definitions of $\mathbf{P}_{ij}$ and $\mathbf{Q}_{ij}$ employed in BHSNE (see Section 5.2). We minimize the ABSNE objective via gradient descent. The gradient, $\partial \mathcal{J}_{\text{ABSNE}} / \partial \mathbf{y}_i$, is computed as

$$\sum_j 4Z\mathbf{Q}_{ij}^2 (\mathbf{y}_i - \mathbf{y}_j)(\underbrace{\mathbf{P}_{ij}^{\alpha} \mathbf{Q}_{ij}^{\beta-1}}_{\text{Force 1}} - \underbrace{\mathbf{Q}_{ij}^{\alpha+\beta-1} - J_1 + J_2)}_{\text{Force 2}} \tag{5.7}$$

where $J_1 = \sum_{k \neq l} \mathbf{P}_{kl}^{\alpha} \mathbf{Q}_{kl}^{\beta}$ and $J_2 = \sum_{k \neq l} \mathbf{Q}_{kl}^{\alpha+\beta}$. We compute ABSNE gradients in $\mathcal{O}(m \log m + mk)$ time using BHSNE's computational tricks: we can compute Force 1 and J_1 in $\mathcal{O}(km)$ time using $\mathbf{P}$'s sparsity and Force 2 via a Barnes-Hut algorithm similar to the one described in Section 5.2 after pre-computing Z and J_2 using a separate Barnes-Hut procedure.

5.4 How α and β Discover Hidden Structures

The AB-divergence offers two hyperparameters, α and β, which have strong intuitive meaning. This conveniently removes the need for tedious grid searches. Defining $\lambda =$

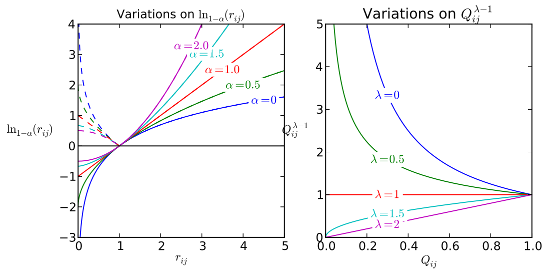


Figure 5.1: Functions in the (left) Tsallis deformed q-logarithm and (right) power families. Dotted lines in the left figure denote absolute values of functions. $|\ln_{1-\alpha}(r_{ij})|$.

$\alpha+\beta$, we inspect the updates taken during learning:

$$\Delta \mathbf{y}_i = -\frac{\partial D_{AB}^{\alpha\beta}(\mathbf{P}\|\mathbf{Q})}{\partial \mathbf{y}_i} = \sum_j \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\mathbf{Q}_{ij}^{\lambda-1}\ln_{1-\alpha}\left(\frac{\mathbf{P}_{ij}}{\mathbf{Q}_{ij}}\right) \tag{5.8}$$

$$= \sum_j \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\mathbf{Q}_{ij}^{\lambda-1}\ln_{1-\alpha}(r_{ij}) \tag{5.9}$$

where $\ln_q(x)$ is the Tsallis deformed q-logarithm* and $r_{ij} = \mathbf{P}_{ij}/\mathbf{Q}_{ij}$. Our theoretical analysis considers how each force $\mathbf{f}_{ij} = \partial\mathbf{Q}_{ij}/\partial\mathbf{y}_i \cdot \mathbf{Q}_{ij}^{\lambda-1}\ln_{1-\alpha}(r_{ij})$ affects $\Delta\mathbf{y}_i$. The term $\partial\mathbf{Q}_{ij}/\partial\mathbf{y}_i$ is a vector parallel to the ray $k(\mathbf{y}_i - \mathbf{y}_j)$. Since the t-distribution monotonically decreases for positive arguments, we must have $k < 0$, i.e., $\mathbf{f}_{ij}$ points towards $\mathbf{y}_j$, implying that $\mathbf{f}_{ij}$ attracts $\mathbf{y}_i$ towards $\mathbf{y}_j$ if $\mathbf{Q}_{ij}^{\lambda-1}\ln_{1-\alpha}(r_{ij}) > 0 \Rightarrow r_{ij} > 1$ (see Figure 5.1) and repulses $\mathbf{y}_i$ from $\mathbf{y}_j$ if $r_{ij} < 1$. Specifically, non-neighboring point pairs in the original dataset $\mathcal{D}$ with $\mathbf{P}_{ij} = 0 \Rightarrow r_{ij} = 0 < 1$ repel each other.

We interleave theoretical intuition with empirical verification on two datasets (more results presented in Section 5.6): MNIST[67], a dataset of 70K 28×28 grayscale images depicting handwritten digits 0-9 and CIFAR-10[59], a dataset of 32×32 color images depicting 10 distinct object classes. We use the raw pixel data as MNIST's feature representation. For CIFAR-10, we train a 3-layer convolutional neural network with the cudaconvnet[58] architecture using Caffe[51] and employ only third layer convolutional features; we visualize the test set to avoid having training labels directly influence the embedding. Applying PCA to center and reduce each dataset to 100 dimensions, we ran ABSNE under various (α, λ) for both datasets with perplexity 30 (Figure 5.2). We initialize all $\mathbf{y}_i$ in experiments with the same random seed and run exactly 1000 iterations of gradient descent per configuration (see Section 5.5 for optimization details), so structural details across a row should be comparable; the first column ($\alpha = 1.0, \lambda = 1.0$) denotes vanilla t-SNE.

Intuition Behind α. To study α, let us fix $\lambda = 1$. Consider a cluster of embedding points consisting of a few sub-clusters. After convergence, the sub-clusters will be

*$\ln_q(x) \equiv (x^{1-q}-1)/(1-q)$ if $q \neq 1$, else $\ln_q(x) = \ln x$.

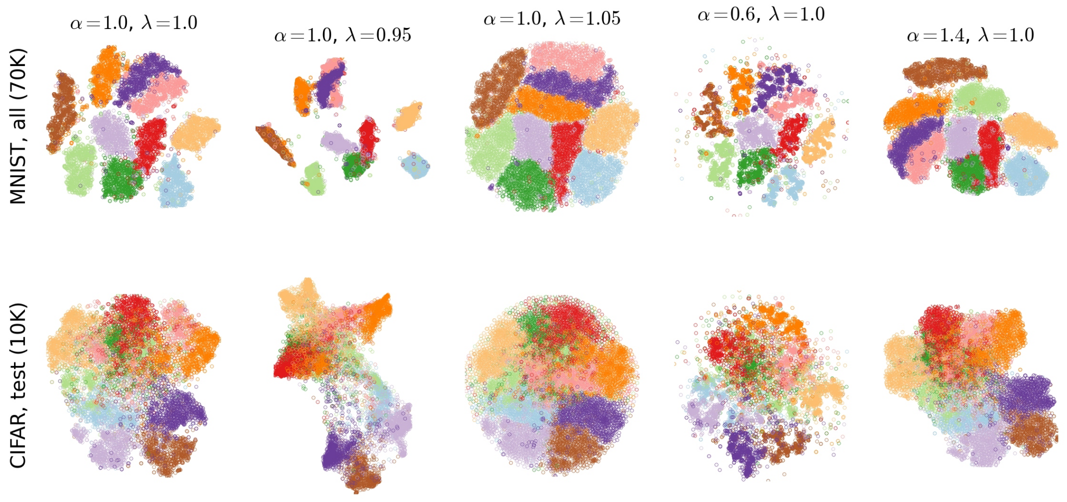


Figure 5.2: Empirical evidence of the theory in Section 5.4: (col. 2) $\lambda < 1$ reveals macro-structures, (col. 4) $\alpha < 1$ reveals micro-structures, and (col. 3) $\lambda > 1$ reveals instances close to class boundaries (Section 5.4). Further evidence on larger datasets (1M+ instances) is provided in Section 5.6. During data analysis, if supervision is not provided, column 2 may help in identifying classes. If supervision is provided, the right three plots can help understand "easily confused" (boundary) instances and instances within fine-grained categories. For reference, the left-most column displays t-SNE's visualization, i.e., in the limit $(\alpha, \lambda) \to (1, 1)$ (this derivation is non-trivial due to the limits and presented in the Supplementary Materials).

placed together in such a way that the attractive and repulsive forces are balanced. According to Figure 5.1a., decreasing α below 1 emphasizes the magnitude of (repulsive) forces with $r_{ij} < 1$ relative to (attractive) forces with $r_{ij} > 1$. The emphasized repulsive forces and diminished attractive forces should cause sub-clusters to be placed further apart, implying that ABSNE should tend to produce lots of small, fine-grained clusters for $\alpha < 1$. Because $r_{ij} < 1 \Rightarrow \mathbf{Q}_{ij} > \mathbf{P}_{ij}$ implies that points $\mathbf{y}_i, \mathbf{y}_j$ are closer than they are supposed to be, the $\mathbf{f}_{ij}$ operating on close-proximity points $\mathbf{y}_i, \mathbf{y}_j$ should be emphasized more than those of far away points, implying that setting $\alpha < 1$ should lead to fewer global changes in visualization structure in comparison with t-SNE ($\alpha = \lambda = 1$), but lots of change in local structure. Similarly, setting $\alpha > 1$ should lead to fewer, larger clusters with more global visualization changes. Inspecting columns 4 and 5 in Figure 5.2, we notice that varying α yields the anticipated effect of local clustering: in both CIFAR-10 and MNIST, individual clusters are more tight and fine-grained for $\alpha < 1$ and loose for $\alpha > 1$. As predicted, little global restructuring takes place for $\alpha < 1$ in comparison with $\alpha > 1$. Variations on α could be useful to a user interested in inspecting the relationships between sub-clusters arising within larger clusters of the data at various scales.

Intuition Behind λ. To study λ, let us fix $\alpha = 1$. According to Figure 5.1b., setting

$\lambda < 1$ emphasizes $\mathbf{f}_{ij}$ with low $\mathbf{Q}_{ij}$ (distant $\mathbf{y}_i, \mathbf{y}_j$), over $\mathbf{f}_{ij}$ with high $\mathbf{Q}_{ij}$ (near $\mathbf{y}_i, \mathbf{y}_j$). So, changing λ should primarily affect global over local structure. Thus, $\lambda < 1$ increases the magnitudes of forces on distant, repulsive point pairs, exaggerating repulsion of non-neighboring point pairs in the original dataset, which we conjecture leads to greater cluster separation while setting $\lambda > 1$ leads to low separation. Inspecting column 2, setting $\lambda < 1$ yields the anticipated effect of greater cluster separation: examining MNIST for $\lambda = 0.95 < 1$, ABSNE places each cluster of points further away from the others. Similarly, the purple and brown clusters are more separated from the other clusters with CIFAR-10. In column 3, setting $\lambda = 1.05 > 1$ yields a single large glob of points containing smaller globs corresponding to the same class, as expected. This setting could be useful if the user wishes to inspect "boundary" cases between embedding points with known classes.

The Importance of Blending α and β. While past work has individually applied the α- and β- divergences to the SNE problem[113,15,112], the heavy dependence of the theory on $\lambda = \alpha + \beta$ shows that incorporating both divergences in the objective is crucial to discovering important micro and macro structures in data.

5.5 Parallel CPU and GPU Implementations

While many optimization methods exist for embeddings, e.g. spectral descent[72], partial Hessian strategies[102], fast multipole methods with L-BGFGS[103], we found that warm-started gradient descent[100] obtained strong results: we (1) initialize all $\mathbf{y}_i$ from a 2D isotropic Gaussian with variance 10^{-4} and (2) update each $\mathbf{y}_i$ via gradient descent (GD) with momentum (step size 200). For the first 250 descent iterations, we use momentum 0.5 and multiply all $\mathbf{P}_{ij}$ values by a user-defined constant $\alpha = 12$. For the last 750 iterations, we use momentum 0.8. We use a per-parameter adaptive learning rate scheme to speed up GD convergence[48], otherwise known to be very slow and sensitive to local optima in practice[102]. Concrete reasonings behind these choices can be found in[99,100]. We now detail how to compute gradients and update the $\mathbf{y}_i$ on the GPU, particularly using the NVIDIA compute unified device architecture (CUDA).

5.5.1 Parallel GPU Gradients

We store the $\mathbf{y}_i$ in two arrays on the GPU, one array per dimension, to take advantage of cache locality, as done in[16]. We store the (sparse) affinity matrix $\mathbf{P}$ as a list of triplets. We take advantage of $\mathbf{P}$'s symmetry to minimize the number of memory reads by storing only the upper half of the matrix. Our implementation consists of 13 kernels, which we now discuss.

Kernels 1 – 5: Partially Computing Force 2. Recall from Section 5.3 that the AB-SNE gradient consists of two types of forces: Force 1 and Force 2. We first compute Force 2, since it will yield structures useful in computing Force 1.

Discussed in Section 5.2, computing Force 2 involves building a quadtree over the $\mathbf{y}_i$. To do this, we (Kernel 1) construct a bounding box over the $\mathbf{y}_i$, (Kernel 2) hierarchically subdivide the bounding box until each cell contains at most a single $\mathbf{y}_i$, and (Kernel 3) compute the center of mass and cumulative mass per cell. Next, we (Kernel 4) perform an in-order traversal of the quadtree, which approximately places nearby cells next to each other in the traversal; this is crucial in accelerating Kernel 5, which actually computes the forces on the individual $\mathbf{y}_i$.

To understand why the in-order traversal is necessary, recall that in CUDA, all threads within a single warp will execute in lockstep on entering a conditional statement only if the conditional evaluation is identical for all threads; otherwise, threads within the warp belonging to different branches will execute serially, often severely affecting performance. The in-order traversal substantially reduces such within-warp thread divergence in Kernel 5, leading to an order of magnitude savings in run-time. For more details, see[16], which we follow closely in implementing these five kernels.

Jointly, this set of kernels computes and stores $g_1 = \sum_j 4Z\mathbf{Q}_{ij}^2(\mathbf{y}_i - \mathbf{y}_j)$ and $g_2 = \sum_j 4Z\mathbf{Q}_{ij}^2(\mathbf{y}_i - \mathbf{y}_j)\mathbf{Q}_{ij}^{\alpha+\beta-1}$ in GPU memory. We incorporate the contributions of J_1 and J_2 in later kernels.

Kernel 6, 7: Computing J_2 and Z. We evaluate $J_2 = \sum_{k\neq l} \mathbf{Q}_{kl}^{\alpha+\beta}$ by (1) computing a $J_2^{(i)} = \sum_{k\neq i} \mathbf{Q}_{ki}^{\alpha+\beta}$ per $\mathbf{y}_i$ and (2) executing a reduction to compute $J_2 = \sum_i J_2^{(i)}$. We efficiently perform (1) by computing auxiliary $J_2^{(i)}$ variables per $\mathbf{y}_i$ in executing Kernel 5. We perform the reduction through the open-source Thrust library[12]. We compute Z similarly.

Kernel 8, 9: Computing J_1, Partially Computing Force 1. Rather than computing Force 1 by iterating over each $\mathbf{y}_i$, we iterate through the positive entries of $\mathbf{P}$, whose contributions we add to the prescribed $\mathbf{y}_i$ (Kernel 8). To parallelize computation, we split the list of triplets $(i, j, \mathbf{P}_{ij})$ across enough blocks to allow for 1024 threads per block; each thread processes a single triplet and updates points i and j. One caveat of this approach is that these updates must be made atomically; e.g., parallel updates for tuples $(2, 3)$ and $(3, 5)$ without atomic updates would yield a race condition for $\mathbf{y}_3$. Kernel 8 computes and stores $g_3 = \sum_j 4Z\mathbf{Q}_{ij}^2(\mathbf{y}_i - \mathbf{y}_j)\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta-1}$ in GPU memory.

Computing J_1 entails iterating through the positive entries of $\mathbf{P}$, computing the associated summands, and executing a reduce operation (Kernel 9). With large datasets, the GPU cannot fully store $\mathbf{P}$ in memory; so, we swap batches of $\mathbf{P}$ between CPU and GPU memory and apply Kernels 8 and 9 per batch, accumulating only J_1 and g_3 in GPU memory.

Kernels 10 – 12: Updating $\mathbf{y}_i$. Kernel 10 executes the gradient updates per $\mathbf{y}_i$. We compute the gradient by modifying each $\mathbf{y}_i$ according to the entry in $g = g_1 * (J_2 - J_1) - g_2 + g_3$, taking into account momentum and the adaptive learning rates described in Section 5.5.

5.5.2 Parallel CPU Gradients

We use a very similar computation flow in computing parallel CPU gradients. In computing Force 2 on the CPU, we build the quadtree serially, as this step typically takes less than 10% of the full training time. After constructing the quadtree, we similarly partially compute Force 2 using OpenMP to parallelize computations over

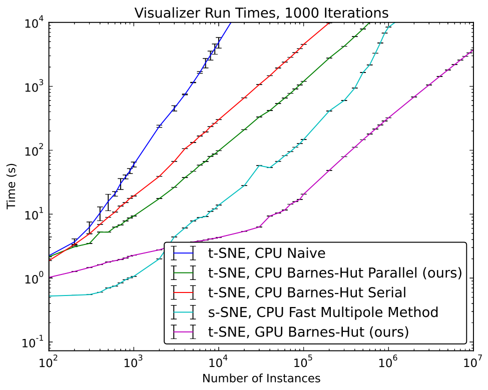


Figure 5.3: Best viewed in color. Timing experiments comparing cpu-naive[99], fmm[103], cpu-bh[100] with our cpu-par-bh and gpu-bh implementations. Error bars denote 2 standard deviations of time across 30 experiments.

each $\mathbf{y}_i$. We then compute Z and J_2 via OpenMP's parallel reduce operation, using similar auxiliary variables as in the GPU implementation. We compute Force 1 and J_1 serially; in practice, we found that using atomic additions in parallel ran slower. We then similarly update the $\mathbf{y}_i$ in parallel.

5.5.3 Performance Experiments

All experiments in this chapter employ a machine with 2x Intel Xeon X5570 CPUs (8 cores total, 2.93 GHz), 64GB memory, and an NVIDIA Tesla K40c graphics card. Corroborated by[100], fixing $\theta = 0.25$ offers a good tradeoff between speed and accuracy. We explore the efficiency and scalability of (1) cpu-naive: a naive CPU implementation[99], (2) cpu-bh: a serial Barnes-Hut CPU implementation[100], (3) fmm: a fast multipole method, a scheme with linear gradient time complexity, the fastest published code available online)[103], (4) cpu-par-bh: our parallel Barnes-Hut CPU implementation, and (5) gpu-bh: our Barnes-Hut GPU implementation. All implementations run t-SNE by setting $\alpha = 1, \beta = 0$. fmm runs s-SNE, since code for t-SNE was not available; timing comparisons are still valid, since the number of operations involved in computing s-SNE and t-SNE gradients are similar. In our experiments, we sample subsets of the HIGGS dataset[8], a large dataset consisting of 11M instances and 28 features (see Section 5.6 for dataset details and visualizations); we use a perplexity of 20. Figure 5.3 summarizes our findings in a log-log plot. As expected, cpu-naive timings have a slope of 2 while cpu-bh, gpu-bh, and fmm timings have slopes close to 1, indicating the expected theoretical complexities.

Barnes-Hut CPU Parallel On datasets with more than 50K instances, cpu-par-bh yields speedups of more than 3.5x over cpu-bh, approaching about half of linear speedup; we do not attain full linear speedup, since Force 1's computation is not parallelized. Corroborated by[103], fmm runs substantially faster than cpu-bh $(20 - 30\text{x})$;

on datasets with more than 1M instances, cpu-par-bh closes this gap to 2.5x.

Barnes-Hut GPU Parallel On datasets with more than 50K instances, gpu-bh yields speedups of $150 - 200$x over cpu-bh and $55 - 60$x over cpu-par-bh. While FMM is 2x faster than gpu-bh on datasets with size < 2K, gpu-bh is $5 - 10$x faster on datasets with size $20 - 500$K and more than 20x faster on datasets with size $1 - 10$M. Between 100 and 50K instances, gpu-bh has a slope smaller than 1; this happens because (i) Force 2 (quadtree) computations dominate computation time and (ii) we are able to process ALL data instances simultaneously on the GPU, effectively yielding $\mathcal{O}(\log m)$ computation time. At the 100K mark, Force 1 dominates computation times because (i) we need to update each data instance's gradient atomically while (ii) swapping portions of the sparse matrix P in and out of GPU memory, causing the slope to return to 1.

5.6 Case Studies

Evaluating visualization quality is a difficult problem; in previous work, researchers have used supervised labels in tandem with a k-nearest neighbors metric to quantitatively assess performance[99,100,111]. However, such scores may not directly translate to "better visualization": For example, a data-miner aspiring to explore micro, within-class clusters in a dataset without supervised labels may assign a low score to an embedding that perfectly separates high-level macro clusters. We now explore AB-SNE for use in such goal-driven visualization. We strongly encourage readers to use a computer in tandem with digital zoom set to around 400% in looking at the large visualization in this section.

5.6.1 ILSVRC 2012: Kingdoms to Species

A subset of ImageNet, ILSVRC 2012[29] features a training set of 1.28M images of varying size, validation set of 50K images, and 1000 object categories. [†] We employ fc7 features yielded by Alexnet[60] implemented in Caffe[51], trained on the 1.28M images.[‡] We show a t-SNE plot of the validation set in the top-middle of Figure 5.4; we also present plots and 2 zoom views for $(\alpha = 0.8, \lambda = 1)$ and $(\alpha = 0.95, \lambda = 0.98)$. Zoom views present a random subset of images from the plot; displayed images were not hand-picked. As Section 5.4 predicts, the top scatter plots show tight clusters and greater class separation in $(\alpha = 0.95, \lambda = 0.98)$ compared with the other settings, particularly clusters corresponding to birds, mammals, and dogs. The maroon oval in the purple panel shows a transition between perched birds to monkeys in trees, while the blue and orange boxes in the red panel show well-separated classes. As expected, we found stronger intra-class clustering for $(\alpha = 0.8, \lambda = 1)$; the purple panel shows

[†] These classes correspond to leaves in a hierarchical tree of classes, with "entity" being the root. For convenient visualization, we greedily group together classes with the same parent until 11 classes remain of roughly equal cardinality ([31] employ a similar method).

[‡] We visualize the validation set rather than the training set, because the training labels were used to train the Caffe model, and using the training labels themselves would yield perfect separation in visualization.

separate clusters for hummingbirds (green), macaw parrots (yellow), toucans (orange), flamingos (blue), chickens (red), ducks (purple), and birds in the sky (gray). Similarly, the blue panel separately clusters dalmations (green), Bernese mountain dogs (blue), black and brown dogs, e.g. Doberman Pinschers and German Rottweilers (orange), and black dogs (red). In comparison, the red panel associated with $(\alpha = 0.95, \lambda = 0.98)$ shows fewer, vague clusters with all birds (orange): chickens (red) and swans/ducks (purple). The green panel shows fewer classes: Bernese mountain dogs (blue) and black and brown dogs (orange).

5.6.2 HIGGS: Discovering Higgs Bosons

We return to the HIGGS dataset[8], whose goal is to distinguish between signal processes which produce Higgs bosons and background processes which do not. Each data instance consists of 28 features: the first 21 describe kinematic properties measured by detectors in the accelerator, while the last 7 are high-level functions of the first 21. Again, we first run t-SNE on the HIGGS dataset (bottom left of Figure 5.4). For HIGGS and SUSY, we first initialize and optimize with 0.5M random instances. Upon convergence, we place another 0.5M new random instances near their nearest neighbor in the original dataset with isotropic Gaussian noise with variance 0.01. Repeating until all points have been embedded, this produces final objectives with lower value.

Interestingly, while clustering occurs, the clusters don't correspond to the desired classes. Hoping that tighter global clustering will lead to more intuitive results, we set $\alpha = 0.98$ (bottom middle plot). As predicted in Section 5.4, straggling points align with existing clusters to yield a cleaner plot. While some clusters appear to have slightly denser concentrations of positive signals, there still is not any concrete class separation. While we are not aware of what the resulting clusters mean, we believe that the clusters could yield further insights.

5.6.3 SUSY: Discovering Supersymmetric Particles

Discovering evidence of supersymmetry (SUSY) constitutes a major goal in the Large Hadron Collider's central mission; one ramification of the theory includes the discovery of dark matter candidate particles[8]. We explore the SUSY dataset[8], which similarly tries to distinguish between processes which do and do not produce supersymmetric particles. There is currently a vigorous effort to improve performance in this classification task[21,9,83,14]. As with HIGGS, we found that setting $\alpha = 0.98, \lambda = 1$ yields a cleaner plot with greater separation than that of t-SNE. While we do not observe perfect cluster separation, there is a distinct purple region corresponding to observed SUSY particles. Perhaps, replicating experimental conditions leading to particles in this region would have a higher chance of yielding SUSY particles.

5.6.4 Connections to Spectral Clustering

The SNE variations discussed and Laplacian Eigenmaps (LE), a widely used spectral embedding method[11], are closely related[19]. Specifically, LE optimizes the function

$\mathcal{J}_{\text{LE}}(\mathcal{E}) = \sum_{i,j} \mathbf{P}_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2$ where the formulation of $\mathbf{P}$ constitutes a design choice. If we expand the original SNE formulation, we are asked to minimize $\mathcal{J}_{\text{SNE}}(\mathcal{E}) = \sum_{i,j} \mathbf{P}_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 + \log \sum_{k \neq l} \exp(-\|\mathbf{x}_k - \mathbf{x}_l\|^2)$. The log-exp term constitutes the primary difference between LE and SNE methods; while the weighted least squares term encourages nearby data vectors to lie closely together in the final embedding, the log-exp term, coming from the denominator of $\mathbf{Q}$ as defined in Equation (5.1), is a data independent term whose gradient causes the embedding points to all repel each other[19]. Several comparisons demonstrate that this term substantially improves the spacing between individual points and clusters[99,19,100].

To arrive at the LE formulation, we can set ABSNE's parameters to reflect BHSNE and eliminate the normalization parameter in $\mathbf{Q}$.[3] Surprisingly, we can compute the globally optimal solution in this case, namely the nontrivial trailing eigenvectors of the normalized Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{P}$, where $\mathbf{D}_{ii} = \text{diag}(\sum_i \mathbf{P}_{ij})$ and 0 elsewhere in $\mathcal{O}(km \log m + km)$ time[11]; the need to only compute the bottom few eigenvectors as well as the Laplacian's sparsity allows for a procedure much faster than the canonical $\mathcal{O}(m^3)$ time.[§]

5.7 Discussion

Although several works have explored varying divergences in SNE methods, this is the first to theoretically attribute and empirically verify how divergence parameters qualitatively affect visualization. Our analysis reveals that parameter variation in (α, β) for the AB-divergence discovers micro and macro structures within data in a dataset-agnostic fashion. Our well-optimized GPU implementation yields speedups of more than 20x on datasets with $1 - 10$M instances over the state of the art implementation[103]. This yields the largest published SNE visualization of a dataset (11M instances).

[§]LE additionally introduces the constraint $Y^T L Y = I$ to avoid the $\mathbf{y}$'s collapsing to 0; this isn't necessary in the t-SNE case because of the log-exp term, which forces repulsion.

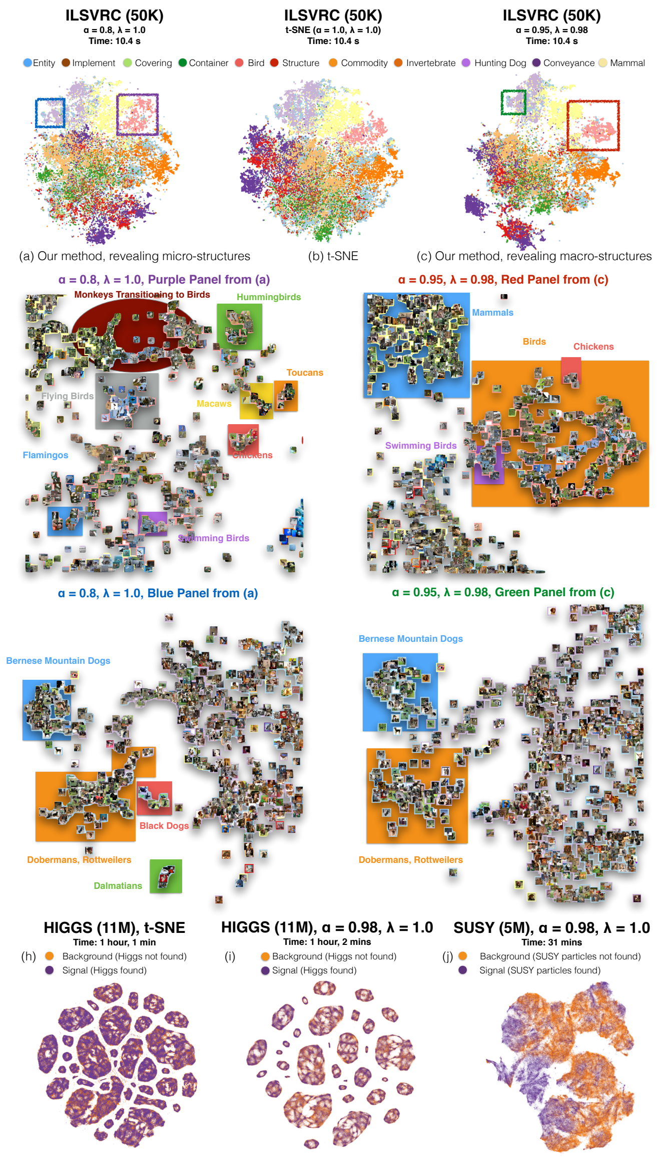


Figure 5.4: Please use digital zoom at high levels to view details. (a - c) depict ILSVRC 2012's micro (specific animal species) in the purple, blue panels and macro (classes under the animal kingdom) structures in the red, green panels. As expected, macro clusters are more widely separated in (c) than (a) and in the red, green panels than purple, blue panels. (h - j) depict analogous structures in the HIGGS and SUSY datasets; we are searching for what physical meaning these structures have in the Standard Model of particle physics.

6 Conclusion

We describe an end-to-end solution that produces high-quality 3D color scans of real world objects. In particular, our approach offers significant improvements in model acquisition speed and quality at a budget. As the world shifts towards producing and consuming 3D media, we believe that our approach will allow for progress in both research and broader consumer industries. Researchers can quickly scan and produce high-quality 3D datasets at scale; as seen in the field of computer vision with the collection of large-scale datasets such as ImageNet[29], 3D computer vision would likely quickly advance as a field. In the broder consumer industry, content developers would be able to rapidly scan existing objects, iterate, and use them in products: from 3D printing to 3D gaming, the possibilities are endless.

Concretely, in this thesis, we discuss how to construct a scanning rig that with a single button click, automatically scans an object that fits within a 1 cubic meter space. This scanning rig addresses reconstruction issues that other techniques face, including transparent and translucent objects, objects with concavities, and objects that lack texture. Capitalizing on the strengths of the data modalities that this rig provides for each object, we detail an algorithm that provides high-fidelity 3D meshes of object scans. In comparison with existing techniques, our method produces reconstructions that are hole-free, 3D printable, and substantially more detailed. We next detail an optimization algorithm that produces high-fidelity textures. In comparison with existing techniques, our algorithm produces textures that are more detailed and less blurry. Ultimately, as an example, given a Coca Cola bottle, we are able to produce a high-quality 3D color scan; difficult features that our algorithms recover include (1) nooks and crannies in the surface of the bottle, despite transparencies, (2) the bottle's barcode and date of creation, both consisting of features under 1 mm in size, (3) a detailed color map of the overall object, despite the highly non-Lambertian nature of the object. Finally, we detail a non-linear dimensionality reduction algorithm that allows us to intuitively visualize large-scale high-dimensional datasets. Although this algorithm currently does not directly have applications in 3D scanning, we anticipate that it will over the next decade, as 3D scanning explodes. Currently, however, our algorithm has applications in fields ranging from computer vision to botany to particle physics.

At the time this thesis was written, the most successful reconstruction algorithms had many moving parts and involved a fair amount of “parameter tweaking,” as with most subfields within artificial intelligence. Indeed, many algorithms in complementing fields often produce hard-earned incremental improvements over the state of the art primarily via minor tunings. Every once in a while, however, a fundamentally new idea changes the game. Once the field of 3D scanning is solved, it is difficult to not wonder whether the final solution will be one where future generations turn through the pages of history and wonder how we could have been so gullible.

References

[rgb] Rgbdtoolkit.

[sip] Solutions in perception challenge.

[cga] Cgal. http://www.cgal.org.

[4] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Süsstrunk, S. (2010). Slic superpixels. Technical report.

[Agarwal et al.] Agarwal, S., Mierle, K., & Others. Ceres solver.

[6] Aldoma, A., Tombari, F., Prankl, J., Richtsfeld, A., Di Stefano, L., & Vincze, M. (2013). Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In ICRA.

[7] Arthur, D. & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In SODA.

[8] Baldi, P., Sadowski, P., & Whiteson, D. (2014). Deep learning in high-energy physics: improving the search for exotic particles. arXiv preprint arXiv:1402.4735.

[9] Barr, A., Lester, C., & Stephens, P. (2003). mt2: the truth behind the glamour. Journal of Physics G: Nuclear and Particle Physics, 29(10), 2343.

[10] Baumgart, B. G. (1974). Geometric Modeling for Computer Vision. Technical report, DTIC Document.

[11] Belkin, M. & Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. In NIPS.

[12] Bell, N. & Hoberock, J. (2011). Thrust: A productivity-oriented library for cuda. GPU Computing Gems, 7.

[13] Bloomenthal, J. (1988). Polygonization of implicit surfaces. Computer Aided Geometric Design, 5(4), 341–355.

[14] Buckley, M. R., Lykken, J. D., Rogan, C., & Spiropulu, M. (2013). Super-razor and searches for sleptons and charginos at the lhc. arXiv preprint arXiv:1310.4827.

[15] Bunte, K., Haase, S., Biehl, M., & Villmann, T. (2012). Stochastic neighbor embedding (sne) for dimension reduction and visualization using arbitrary divergences. Neurocomputing, 90, 23–45.

[16] Burtscher, M. & Pingali, K. (2011). An efficient cuda implementation of the tree-based barnes hut n-body algorithm. GPU Computing Gems Emerald edition, (pp.75).

[17] Butler, D. A., Izadi, S., Hilliges, O., Molyneaux, D., Hodges, S., & Kim, D. (2012). Shake'n'sense: reducing interference for overlapping structured light depth cameras. In Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems (pp. 1933–1936).: ACM.

[18] Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., & Dollar, A. M. (2015). Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols.

[19] Carreira-Perpinan, M. A. (2010). The elastic embedding algorithm for dimensionality reduction. In ICML.

[20] C.E., H. & Schmitt, F. (2004). Silhouette and stereo fusion for 3d object modeling. CVII.

[21] Cheng, H. & Han, Z. (2008). Minimal kinematic constraints and mt2. Journal of High Energy Physics, 2008(12), 063.

[22] Chiu, W. C., Blanke, U., & Fritz, M. (2011). Improving the kinect by cross-modal stereo. In BMVC, volume 1 (pp.3).

[23] Cichocki, A., Cruces, S., & Amari, S. (2011). Generalized alpha-beta divergences and their application to robust nonnegative matrix factorization. Entropy, 13, 134–170.

[24] Cignoni, P., Corsini, M., & Ranzuglia, G. (2008). Meshlab: an open-source 3d mesh processing system. ERCIM News, 2008(73).

[25] Collet, A., Martinez, M., & Srinivasa, S. S. (2011). The MOPED framework: Object Recognition and Pose Estimation for Manipulation.

[26] Collins, R. T. (1996). A space-sweep approach to true multi-image matching. In CVPR (pp. 358–363).

[27] Cremers, D. & Kolev, K. (2011). Multiview stereo and silhouette consistency via convex functionals over convex domains. 33.

[28] Dalal, N. & Triggs, B. (2005). Histograms of oriented gradients for human detection. In ICCV.

[29] Deng, J., Dong, W., Socher, R., Li, L., Li, K., & Fei-Fei, L. (2009a). Imagenet: A large-scale hierarchical image database. In CVPR.

[30] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009b). ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09.

[31] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., & Darrell, T. (2013). Decaf: A deep convolutional activation feature for generic visual recognition. arXiv preprint arXiv:1310.1531.

[32] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. International Journal of Computer Vision, 88(2), 303–338.

[33] Fei-Fei, L., Fergus, R., & Perona, P. (2007). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. Comput. Vis. Image Underst., 106(1), 59–70.

[34] Forbes, K., Voigt, A., Bodika, N., et al. (2004). Visual hulls from single uncalibrated snapshots using two planar mirrors. In Pattern Recognition Association of South Africa.

[35] Franco, J., Boyer, E., et al. (2003). Exact polyhedral visual hulls. In BMVC, volume 1 (pp. 329–338).

[36] Furukawa, Y., Curless, B., Seitz, S. M., & Szeliski, R. (2010). Towards internet-scale multi-view stereo. In CVPR.

[37] Furukawa, Y. & Ponce, J. (2006). Carved visual hulls for image-based modeling. In ECCV (pp. 564–577).

[38] Furukawa, Y. & Ponce, J. (2010a). Accurate, dense, and robust multi-view stereopsis. PAMI, 32(8).

[39] Furukawa, Y. & Ponce, J. (2010b). Accurate, dense, and robust multiview stereopsis. PAMI.

[40] Gal, R., Wexler, Y., Ofek, E., Hoppe, H., & Cohen-Or, D. (2010). Seamless montage for texturing models. In CGF.

[41] Geiger, A., Moosmann, F., Car, O., & Schuster, B. (2012). A toolbox for automatic calibration of range and camera sensors using a single shot. In ICRA.

[42] Guillemaut, J. & Hilton, A. (2011). Joint multi-layer segmentation and reconstruction for free-viewpoint video applications. Int. J. Comput. Vision, 93(1), 73–100.

[43] Hernández, C. (2004). Stereo and silhouette fusion for 3d object modeling from uncalibrated images under circular motion. Doctoral thesis.

[44] Herrera, C. D., Kannala, J., & Heikkilä, J. (2011). Accurate and practical calibration of a depth and color camera pair. In CAIP.

[45] Hinton, G. & Roweis, S. (2003). Stochastic neighbor embedding. In NIPS.

[46] Huang, G. B., Ramesh, M., Berg, T., & Learned-Miller, E. (2007). Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. Technical report.

[47] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., & Fitzgibbon, A. (2011). Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In UIST.

[48] Jacobs, R. (1988). Increased rates of convergence through learning rate adaptation. Neural Networks, 1, 295–307.

[49] Jancosek, M. & Pajdla, T. (2011). Multi-view reconstruction preserving weakly-supported surfaces. In CVPR (pp. 3121–3128).

[50] Janoch, A., Karayev, S., Jia, Y., Barron, J. T., Fritz, M., Saenko, K., & Darrell, T. (2011). A category-level 3-d object dataset: Putting the kinect to work.

[51] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding.

[52] Kasper, A., Xue, Z., & Dillmann, R. (2012). The kit object models database: An object model database for object recognition, localization and manipulation in service robotics. The International Journal of Robotics Research, 31(8), 927–934.

[53] Kazhdan, M., Bolitho, M., & Hoppe, H. (2006a). Poisson surface reconstruction. In SGP.

[54] Kazhdan, M., Bolitho, M., & Hoppe, H. (2006b). Poisson surface reconstruction. In ESGP.

[55] Kobbelt, L. (2000a). Sqrt 3-subdivision. In CGIT.

[56] Kobbelt, L. (2000b). Sqrt 3-subdivision. In CGIT.

[57] Konolige, K. & Mihelich, P. (2013). Technical description of kinect calibration.

[58] Krizhevsky, A. (2011). cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. https://code.google.com/p/cuda-convnet/.

[59] Krizhevsky, A. & Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report.

[60] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In NIPS.

[61] Labatut, P., Pons, J.-P., & Keriven, R. (2009). Robust and efficient surface reconstruction from range data. In CGF, volume 28 (pp. 2275–2290).

[62] Ladikos, A., Benhimane, S., & Navab, N. (2008). Efficient visual hull computation for real-time 3d reconstruction using cuda. In CVPR Workshop (pp. 1–8).

[63] Lai, K., Bo, L., Ren, X., & Fox, D. (2011). A large-scale hierarchical multi-view rgb-d object dataset. In ICRA.

[64] Lawrence, N. (2011). Spectral dimensionality reduction via maximum entropy. In AISTATS.

[65] Lazebnik, S., Furukawa, Y., & Ponce, J. (2007). Projective visual hulls. IJCV.

[66] Le, Q. V. & Ng, A. Y. (2009). Joint calibration of multiple sensors. In IROS.

[67] LeCun, Y. & Cortes, C. (1998). The mnist database of handwritten digits.

[68] Lindstrom, P. & Turk, G. (1998). Fast and memory efficient polygonal simplification. In Visualization'98.

[69] Lorensen, W. E. & Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In SIGGRAPH.

[70] Martinez, M., Collet, A., & Srinivasa, S. (2010). Moped: A scalable and low latency object recognition and pose estimation system. In ICRA.

[71] Matusik, W., Buehler, C., Raskar, R., Gortler, S. J., & McMillan, L. (2000). Image-based visual hulls. In Computer Graphics and Interactive Techniques (pp. 369–374).

[72] Memisevic, R. & Hinton, G. (2005). Improving dimensionality reduction with spectral gradient descent. Neural networks, 18(5), 702–710.

[73] Mücke, P., Klowsky, R., & Goesele, M. (2011). Surface reconstruction from multi-resolution sample points. In VMV (pp. 105–112).

[74] Narayan, K. S. & Abbeel, P. (2015). Optimized color models for high-quality 3d scanning. In IROS.

[75] Narayan, K. S., Punjani, A., & Abbeel, P. (2015a). Alpha-beta divergences discover micro and macro structures in data. In ICML.

[76] Narayan, K. S., Sha, J., Singh, A., & Abbeel, P. (2015b). Range sensor and silhouette fusion for high-quality 3d scanning. In ICRA.

[77] Nene, S., Nayar, S., & Murase, H. (1996). Columbia object image library (COIL-20). Technical report, Technical Report CUCS-005-96.

[78] Neugebauer, P. J. & Klein, K. (1999). Texturing 3d models of real world objects from multiple unregistered photographic views. In CGF.

[79] Newcombe, R., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A., Kohli, P., Shotton, J., Hodges, S., & Fitzgibbon, A. (2011a). Kinectfusion: Real-time dense surface mapping and tracking. In ISMAR.

[80] Newcombe, R. A., Davison, A. J., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., Molyneaux, D., Hodges, S., Kim, D., & Fitzgibbon, A. (2011b). Kinectfusion: Real-time dense surface mapping and tracking. In ISMAR (pp. 127–136).

[81] Pulli, K., Piiroinen, S., Duchamp, T., & Stuetzle, W. (2005). Projective surface matching of colored 3d scans. In 3-DIM.

[82] Pulli, K. & Shapiro, L. G. (2000). Surface reconstruction and display from range and color data. Graphical Models.

[83] Rogan, C. (2010). Kinematical variables towards new dynamics at the lhc. arXiv preprint arXiv:1006.2727.

[84] Samaria, F. S. & Harter, A. C. (1994). Parameterisation of a stochastic model for human face identification. In Applications of Computer Vision.

[85] Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., & Ng, A. Y. (2011). On random weights and unsupervised feature learning. In ICML.

[86] Schneider, D. (2009). Visual Hull. Technical report, The University of Edinburgh.

[87] Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., & Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In CVPR, volume 1 (pp. 519–528).

[88] Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In ECCV.

[89] Singh, A., Narayan, K., Kehoe, B., Patil, S., Goldberg, K., & Abbeel, P. (2016). Amazon picking challenge object scans.

[90] Singh, A., Sha, J., Narayan, K. S., Achim, T., & Abbeel, P. (2014). Bigbird: A large-scale 3d database of object instances. In ICRA.

[91] Smisek, J., Jancosek, M., & Pajdla, T. (2013a). 3d with kinect. In Consumer Depth Cameras for Computer Vision (pp. 3–25). Springer.

[92] Smisek, J., Jancosek, M., & Pajdla, T. (2013b). 3d with kinect. In Consumer Depth Cameras for Computer Vision (pp. 3–25).

[93] Steinbrucker, F., Sturm, J., & Cremers, D. (2011). Real-time visual odometry from dense rgb-d images. In Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on (pp. 719–722).

[94] Strecha, C., Fransens, R., & Van Gool, L. (2004). Wide-baseline stereo from multiple views: a probabilistic account. In CVPR, volume 1 (pp. I–552).

[95] Sturm, J., Engelhard, J., Endres, F., Burgard, W., & Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In IROS.

[96] Tang, J., Miller, S., Singh, A., & Abbeel, P. (2012a). A textured object recognition pipeline for color and depth image data. In ICRA.

[97] Tang, J., Miller, S., Singh, A., & Abbeel, P. (2012b). A textured object recognition pipeline for color and depth image data. In ICRA.

[98] Treece, G. M., Prager, R. W., & Gee, A. H. (1999). Regularised marching tetrahedra: improved iso-surface extraction. Computers & Graphics, 23(4), 583–598.

[99] van der Maaten, L. (2008). Visualizing high-dimensional data using t-sne. JMLR, 9, 2579–2605.

[100] van der Maaten, L. (2013). Barnes-hut-sne. In ICLR.

[101] Vaskevicius, N., Pathak, K., Ichim, A., & Birk, A. (2012). The jacobs robotics approach to object recognition and localization in the context of the icra'11 solutions in perception challenge. In ICRA.

[102] Vladymyrov, M. & Carreira-Perpinan, M. (2012). Partial-hessian strategies for fast learning of nonlinear embeddings. In ICML.

[103] Vladymyrov, M. & Carreira-Perpinán, M. A. (2014). Linear-time training of nonlinear low-dimensional embeddings. In AISTATS (pp. 968–977).

[104] Warren, M., McKinnon, D., & Upcroft, B. (2013). Online Calibration of Stereo Rigs for Long-Term Autonomy. In International Conference on Robotics and Automation (ICRA) Karlsruhe.

[105] Whelan, T., Johannsson, H., Kaess, M., Leonard, J. J., & McDonald, J. (2012a). Robust tracking for real-time dense rgb-d mapping with kintinuous.

[106] Whelan, T., Johannsson, H., Kaess, M., Leonard, J. J., & McDonald, J. (2013). Robust real-time visual odometry for dense rgb-d mapping. In ICRA.

[107] Whelan, T., Kaess, M., Fallon, M., Johannsson, H., Leonard, J., & McDonald, J. (2012b). Kintinuous: Spatially extended KinectFusion. In RSS-W on RGB-D: Advanced Reasoning with Depth Cameras.

[108] Xie, Z., Singh, A., Uang, J., Narayan, K. S., & Abbeel, P. (2013). Multimodal blending for high-accuracy instance recognition. In IROS.

[109] Xu, D., Cai, J., Cham, T. J., Fu, P., & Zhang, J. (2012). Kinect-based easy 3d object reconstruction. In Advances in Multimedia Information Processing–PCM 2012 (pp. 476–483).

[110] Yamauchi, H., Lensch, H., Haber, J., & Seidel, H. (2005). Textures revisited. The Visual Computer.

[111] Yang, Z., King, I., Xu, Z., & Oja, E. (2009). Heavy-tailed symmetric stochastic neighbor embedding. In NIPS.

[112] Yang, Z., Peltonen, J., & Kaski, S. (2013). Scalable optimization of neighbor embedding for visualization. In ICML.

[113] Yang, Z., Peltonen, J., & Kaski, S. (2014). Optimization equivalence of divergences improves neighbor embedding. In Proceedings of the 31st International Conference on Machine Learning (ICML-14) (pp. 460–468).

[114] Yianilos, P. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. In ACM-SIAM Symposium on Discrete Algorithms (pp. 311–321).

[115] Zhang, C. & Zhang, Z. (2011). Calibration between depth and color sensors for commodity depth cameras. In ICME.

[116] Zhang, Z. (1999). Flexible camera calibration by viewing a plane from unknown orientations. In ICCV.

[Zhou & Koltun] Zhou, Q. & Koltun, V. Simultaneous localization and calibration: Self-calibration of consumer depth cameras.

[118] Zhou, Q. & Koltun, V. (2014). Color map optimization for 3d reconstruction with consumer depth cameras. ACM TOG.

[119] Zhou, Q., Miller, S., & Koltun, V. (2013). Elastic fragments for dense scene reconstruction. In ICCV (pp. 473–480).

A

Continuity and Gradients for the Alpha-Beta Divergence

The form of the original Alpha-Beta divergence used in this thesis, defined $\mathcal{J}(\mathcal{E};\alpha,\beta) = D_{AB}^{\alpha\beta}(\mathbf{P}\|\mathbf{Q})$, is computed as:

$$\frac{1}{\alpha\beta}\sum_{i\neq j}\left(-\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta}+\frac{\alpha}{\alpha+\beta}\mathbf{P}_{ij}^{\alpha+\beta}+\frac{\beta}{\alpha+\beta}\mathbf{Q}_{ij}^{\alpha+\beta}\right), \tag{A.1}$$

where $\alpha \in \mathbb{R}\setminus\{0\}, \beta \in \mathbb{R}$ are hyperparameters. According to[23], to account for cases such as $\beta = 0$ and $\alpha+\beta = 0$ in this objective, we can re-define:

$$D_{AB}^{\alpha\beta}(\mathbf{P}\|\mathbf{Q}) = \sum_{ij} d_{AB}^{(\alpha,\beta)}(\mathbf{P}_{ij},\mathbf{Q}_{ij}), \tag{A.2}$$

where:

$$d_{AB}^{(\alpha,\beta)}(\mathbf{P}_{ij},\mathbf{Q}_{ij}) = \begin{cases} -\frac{1}{\alpha\beta}\left(\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta}-\frac{\alpha}{\alpha+\beta}\mathbf{P}_{ij}^{\alpha+\beta}-\frac{\alpha}{\alpha+\beta}\mathbf{Q}_{ij}^{\alpha+\beta}\right), & \alpha,\beta,\alpha+\beta\neq 0\\ \frac{1}{\alpha^2}\left(\mathbf{P}_{ij}^{\alpha}\ln\frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha}}-\mathbf{P}_{ij}^{\alpha}+\mathbf{Q}_{ij}^{\alpha}\right), & \alpha\neq 0,\beta=0\\ \frac{1}{\alpha^2}\left(\ln\frac{\mathbf{Q}_{ij}^{\alpha}}{\mathbf{P}_{ij}^{\alpha}}+\frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha}}-1\right), & \alpha=-\beta\neq 0\\ \frac{1}{\beta^2}\left(\mathbf{Q}_{ij}^{\beta}\ln\frac{\mathbf{Q}_{ij}^{\beta}}{\mathbf{P}_{ij}^{\beta}}+\mathbf{P}_{ij}^{\beta}\right), & \alpha=0,\beta\neq 0\\ \frac{1}{2}(\ln\mathbf{P}_{ij}-\ln\mathbf{Q}_{ij})^2, & \alpha=\beta=0 \end{cases} \tag{A.3}$$

As described in this thesis, note that we obtain the KL-divergence by setting $\alpha = 1, \beta = 0$ and the Itakura-Saito divergence by setting $\alpha = 1, \beta = -1$, since $\mathbf{P}$ and $\mathbf{Q}$ are probability distributions. Given these various definitions, we would like to ensure that the gradient described in this thesis is the same particularly for the first three cases (namely where $\alpha \neq 0$), as these are the ones that arise in our reductions. Specifically, we need to ensure that the gradients $\partial D_{AB}^{(\alpha,\beta)}(\mathbf{P}\|\mathbf{Q})/\partial \mathbf{y}_i$ all match.

Case 1: $\alpha, \beta \neq 0$. We have that:

$$\begin{aligned}
\frac{\partial D_{AB}^{(\alpha,\beta)}(\mathbf{P}\|\mathbf{Q})}{\partial \mathbf{y}_i} &= \frac{\partial}{\partial \mathbf{y}_i}\left[-\frac{1}{\alpha\beta}\left(\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta} - \frac{\alpha}{\alpha+\beta}\mathbf{P}_{ij}^{\alpha+\beta} - \frac{\beta}{\alpha+\beta}\mathbf{Q}_{ij}^{\alpha+\beta}\right)\right] && \text{(A.4)}\\
&= \frac{\partial}{\partial \mathbf{y}_i}\left[-\frac{1}{\alpha\beta}\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta} + \frac{1}{\alpha(\alpha+\beta)}\mathbf{Q}_{ij}^{\alpha+\beta}\right] && \text{(A.5)}\\
&= \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\cdot\left[-\frac{1}{\alpha\beta}\mathbf{P}_{ij}^{\alpha}\cdot\beta\cdot\mathbf{Q}_{ij}^{\beta-1} + \frac{1}{\alpha(\alpha+\beta)}\cdot(\alpha+\beta)\cdot\mathbf{Q}_{ij}^{\alpha+\beta-1}\right] && \text{(A.6)}\\
&= \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\cdot\left[-\frac{1}{\alpha}\mathbf{P}_{ij}^{\alpha}\cdot\mathbf{Q}_{ij}^{\beta-1} + \frac{1}{\alpha}\cdot\mathbf{Q}_{ij}^{\alpha+\beta-1}\right] && \text{(A.7)}\\
&= -\frac{1}{\alpha}\cdot\frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\cdot\left[\mathbf{P}_{ij}^{\alpha}\mathbf{Q}_{ij}^{\beta-1} - \mathbf{Q}_{ij}^{\alpha+\beta-1}\right] && \text{(A.8)}
\end{aligned}$$

In the remainder of the cases, we set β to the appropriate value to demonstrate that the gradients match.

Case 2: $\alpha \neq 0, \beta = 0$. We have that:

$$\begin{aligned}
\frac{\partial D_{AB}^{(\alpha,\beta)}(\mathbf{P}\|\mathbf{Q})}{\partial \mathbf{y}_i} &= \frac{\partial}{\partial \mathbf{y}_i}\left[\frac{1}{\alpha^2}\left(\mathbf{P}_{ij}^{\alpha}\ln\frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha}} - \mathbf{P}_{ij}^{\alpha} + \mathbf{Q}_{ij}^{\alpha}\right)\right] && \text{(A.9)}\\
&= \frac{\partial}{\partial \mathbf{y}_i}\left[-\frac{1}{\alpha^2}\mathbf{P}_{ij}^{\alpha}\ln\mathbf{Q}_{ij}^{\alpha} + \frac{1}{\alpha^2}\mathbf{Q}_{ij}^{\alpha}\right] && \text{(A.10)}\\
&= \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\left[-\frac{1}{\alpha^2}\mathbf{P}_{ij}^{\alpha}\cdot\alpha\mathbf{Q}_{ij}^{\alpha-1}\cdot\frac{1}{\mathbf{Q}_{ij}^{\alpha}} + \frac{1}{\alpha}\mathbf{Q}_{ij}^{\alpha-1}\right] && \text{(A.11)}\\
&= \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\left[-\frac{1}{\alpha}\frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha}} + \frac{1}{\alpha}\mathbf{Q}_{ij}^{\alpha-1}\right] && \text{(A.12)}\\
&= -\frac{1}{\alpha}\cdot\frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i}\cdot\left[\frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha}} - \mathbf{Q}_{ij}^{\alpha-1}\right] && \text{(A.13)}
\end{aligned}$$

Setting $\beta = 0$ in Equation (A.8), we observe that the gradients match, as desired.

Case 3: $\alpha = -\beta \neq 0$. We have that:

$$\frac{\partial D_{AB}^{(\alpha,\beta)}(\mathbf{P}\|\mathbf{Q})}{\partial \mathbf{y}_i} = \frac{\partial}{\partial \mathbf{y}_i}\left[\frac{1}{\alpha^2}\left(\ln \frac{\mathbf{Q}_{ij}^{\alpha}}{\mathbf{P}_{ij}^{\alpha}} + \frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha}} - 1\right)\right] \tag{A.14}$$

$$= \frac{\partial}{\partial \mathbf{y}_i}\left[\frac{1}{\alpha^2}\left(\ln \mathbf{Q}_{ij}^{\alpha} + \frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha}}\right)\right] \tag{A.15}$$

$$= \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i} \cdot \left[\frac{1}{\alpha^2} \cdot \alpha \mathbf{Q}_{ij}^{\alpha-1} \cdot \frac{1}{\mathbf{Q}_{ij}^{\alpha}} + \frac{1}{\alpha^2} \cdot \mathbf{P}_{ij}^{\alpha} \cdot -\alpha \cdot \frac{1}{\mathbf{Q}_{ij}^{\alpha+1}}\right] \tag{A.16}$$

$$= -\frac{1}{\alpha} \cdot \frac{\partial \mathbf{Q}_{ij}}{\partial \mathbf{y}_i} \cdot \left[\frac{\mathbf{P}_{ij}^{\alpha}}{\mathbf{Q}_{ij}^{\alpha+1}} - \frac{1}{\mathbf{Q}_{ij}}\right] \tag{A.17}$$

Setting $\beta = -\alpha$ in Equation (A.8), we again observe that the gradients match, as desired. It follows that for all cases in this thesis that we explore, we can simply use the gradient in Equation (A.8).

B ABSNE Embeddings of Various Datasets

In Figures B.1, B.2 and B.3 we showcase visualizations of datasets on which we presented some results in the main text, but did not include plots for ease in exposition.

These plots aim to show the impact of α and β on qualitative properties of the embedding: changing λ should primarily affect global over local structure, where (i) $\lambda < 1$ should lead to greater cluster separation while (ii) $\lambda > 1$ should lead to low separation. Further, ABSNE should tend to produce lots of small, fine-grained clusters for $\alpha < 1$ with few global changes in visualization while $\alpha > 1$ should lead to fewer, larger clusters with more global visualization changes.

We directly use pixels for the ORL and COIL-20 vision datasets, while we compute fc7 features yielded by Caffe's ImageNet model[51] for Caltech256. We use raw features for the other datasets. For all datasets with over 100 dimensions, we first apply 100 dimensional PCA before computing neighborhood scores.

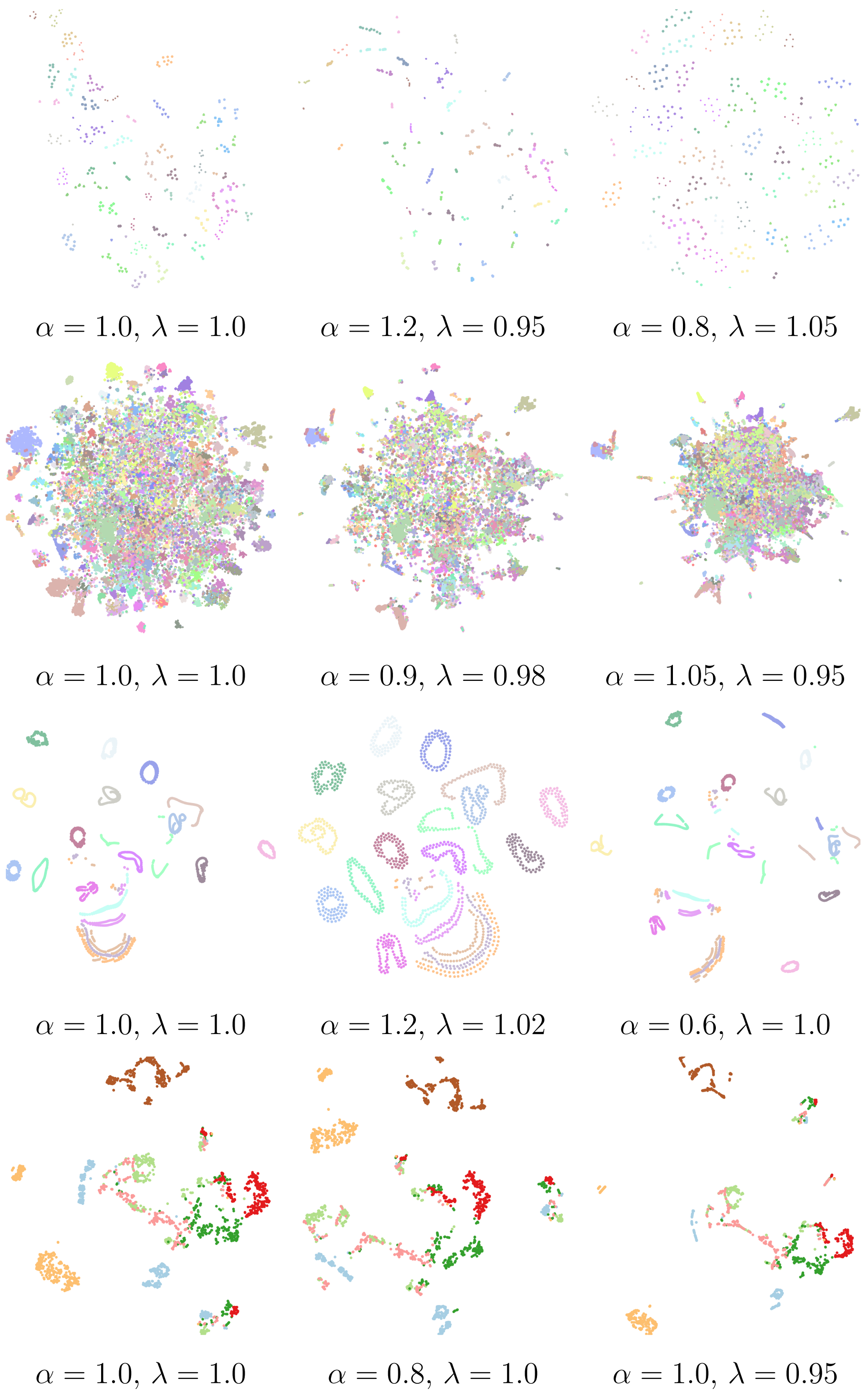


Figure B.1: ABSNE visualizations for (rows from the top) ATT-Faces, Caltech256, COIL20, Segmentation Datasets. The left column corresponds to t-SNE.

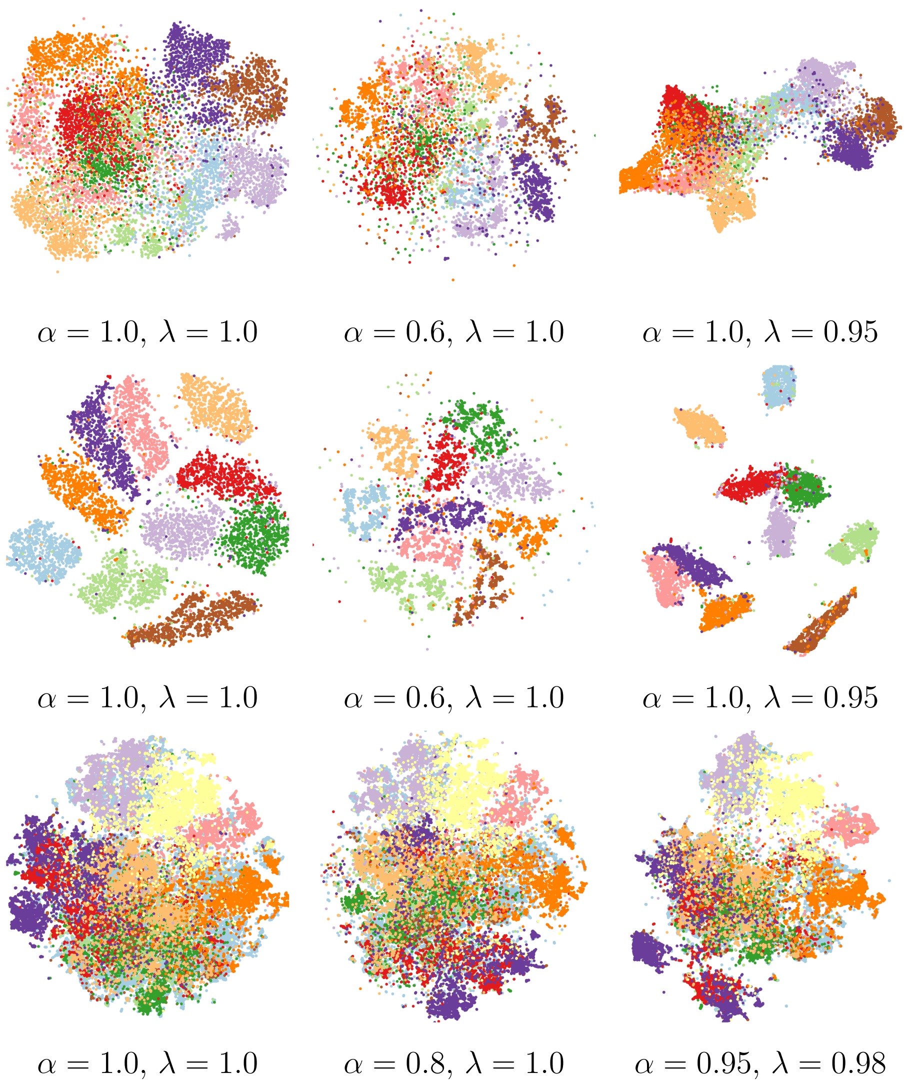


Figure B.2: ABSNE visualizations for (rows from the top) CIFAR10, MNIST, ILSVRC2012 (validation) Datasets. The left column corresponds to t-SNE.

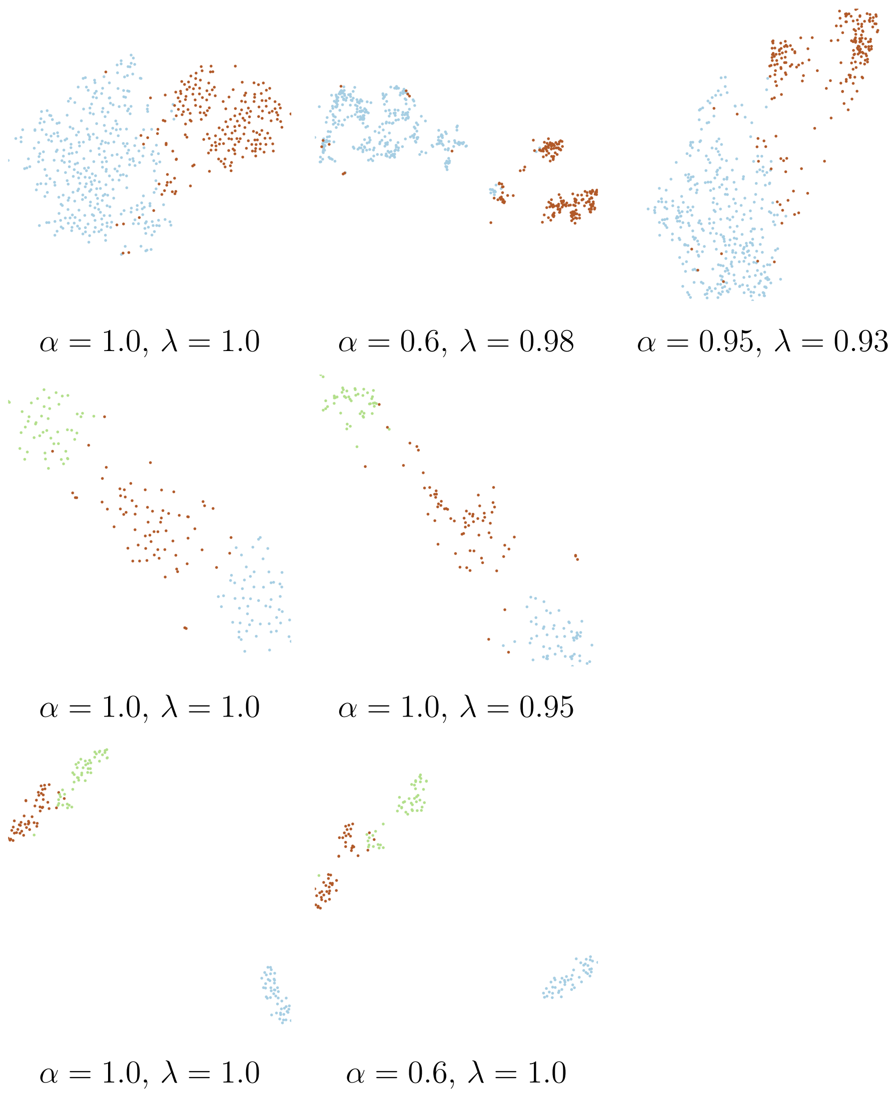


Figure B.3: BSNE visualizations for (rows from the top) WDBC, WINE, IRIS Datasets. The left column corresponds to t-SNE. WINE and IRIS are very small datasets with clear clusters, and ABSNE plots do not differ much from t-SNE.