A 2D + 3D Rich Data Approach to Scene Understanding

by

Jianxiong Xiao

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2013

Author .... [illegible]
Department of Electrical Engineering and Computer Science
August 20, 2013

Certified by .... [illegible]
Antonio Torralba
Associate Professor
Thesis Supervisor

Accepted by .... [illegible]
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Theses

A 2D + 3D Rich Data Approach to Scene Understanding

by

Jianxiong Xiao

Submitted to the Department of Electrical Engineering and Computer Science
on August 20, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

On your one-minute walk from the coffee machine to your desk each morning, you pass by dozens of scenes – a kitchen, an elevator, your office – and you effortlessly recognize them and perceive their 3D structure. But this one-minute scene-understanding problem has been an open challenge in computer vision since the field was first established 50 years ago. In this dissertation, we aim to rethink the path researchers took over these years, challenge the standard practices and implicit assumptions in the current research, and redefine several basic principles in computational scene understanding.

The key idea of this dissertation is that learning from *rich data* under natural setting is crucial for finding the right representation for scene understanding. First of all, to overcome the limitations of *object-centric* datasets, we built the Scene Understanding (SUN) Database, a large collection of real-world images that exhaustively spans all scene categories. This *scene-centric* dataset provides a more natural sample of human visual world, and establishes a realistic benchmark for standard 2D recognition tasks. However, while an image is a 2D array, the world is 3D and our eyes see it from a viewpoint, but this is not traditionally modeled. To obtain a 3D understanding at high-level, we reintroduce geometric figures using modern machinery. To model scene viewpoint, we propose a panoramic place representation to go beyond aperture computer vision and use data that is close to natural input for human visual system. This paradigm shift toward rich representation also opens up new challenges that require a new kind of big data – data with extra descriptions, namely *rich data*. Specifically, we focus on a highly valuable kind of *rich data* – multiple viewpoints in 3D – and we build the SUN3D database to obtain an integrated *place-centric* representation of scenes. We argue for the great importance of modeling the computer's role as an agent in a 3D scene, and demonstrate the power of place-centric scene representation.

Thesis Supervisor: Antonio Torralba
Title: Associate Professor

Acknowledgments

Although a thesis advisor always plays a critical role in the creation of a thesis, I believe that my advisor, Antonio Torralba, deserves special mention. Antonio is truly unique. He always surprised me by providing creative advice in many ways that I wouldn't have expected at all. He led me to think in a creative and unconventional way that completely revolutionized my original view for research. His attitude towards ultimate perfection also drove me to work harder and harder. He has provided a perfect blend of big thinking and practical guidance, while giving me enough freedom to work independently. His excitement about research made me fall in love with computer vision as deeply as he does. I am thankful for having the chance to work closely with him and proud of our accomplishments over the past four years.

My deep gratitude also goes to my other thesis committee members, Bill Freeman and Alyosha Efros. I am grateful for the involvement of Bill Freeman who provided me several key insights into my work and career. Bill is very experienced and has been always very nice to me. He always put the benefits of the students as the number one priority. He has offered many important suggestions that led to significant changes in my research and my life. Bill is an amazing researcher that really knows how to do good research. I am very lucky to witness the development of his recent great works on motion magnification. Although I didn't participate directly, I still learned a significant amount from his examples. I also want to thank Alyosha Efros for many reasons. Alyosha is my role model in many aspects. He is a great thinker, and his research has significant influence on the content of my thesis. He provided me with great advice for my career and he has an amazingly funny personality as well. He is the person that I spent a lot of time to study, from writing style, research idea, to even details like how to illustrate a concept vividly. He also gave me many insightful comments on my thesis.

I am also very grateful for the involvement of Aude Oliva who collaborated with me very closely for several projects. Aude's human-driven perspective to vision is complementary to my computational perspective, which provided tremendous new insights into my works. She is always very hardworking, very generous and very motivated for science.

With experienced background in experimental psychology and cognitive science, Aude also taught me many important details for designing experiments involving human subjects.

I also want to thank my M.Phil. advisor, Long Quan. He introduced me to the amazing world of computer vision research. He trained me to establish a solid foundation of computer vision and multiple-view geometry, which has enabled me to go further in pursuing a Ph.D. and a scientific career. Without his attractive demos on image-based 3D modeling, I wouldn't have started research in computer vision.

I am very lucky to study vision in MIT, probably the only place in the world with so many amazing vision-related faculty members all in one place. Especially, I benefit a lot from my interaction with Fredo Durand, Wojciech Matusik, Ted Adelson, Berthold Horn, Pawan Sinha, Josh Tenenbaum, Shimon Ullman, Tomaso Poggio, Ramesh Raskar and Ruth Rosenholtz. I also learned a lot through lectures from Marvin Minsky, Noam Chomsky, Seth Teller, H. Sebastian Seung, James DiCarlo, Rodney Brooks, Patrick Winston, Matthew Wilson and Nancy Kanwisher from MIT, as well as Jeremy Wolfe, Steven Pinker and George Alvarez from Harvard, and through anecdotes about David Marr.

I also want to thank my internship host in Google, Yasutaka Furukawa, and my internship host in Microsoft Research, Ce Liu. Both of them worked very closely with me, and I benefited a lot through daily interaction during internship. I want to thank Ron Weiss, my academic advisor for providing me great career advice. And thank my Google Ph.D. Fellowship mentor, Mei Han, for her great support and generous help in many situations.

I have enjoyed collaborating with Antonio Torralba, Aude Oliva, James Hays, Bryan Russell, Yasutaka Furukawa, Aditya Khosla, Hao Jiang, Krista A. Ehinger, Andrew Owens, Ping Tan and Phillip Isola. They helped me in so many ways that made all my achievement possible. I am lucky to have the opportunity to work closely with three students, Shuran Song, Erika Lee and Yinda Zhang.

I feel very privileged to have several close friends in the lab. Especially, Andrew Owens, Aditya Khosla, Joseph Lim and Tomasz Malisiewicz have been helping my research and my life in many important ways. They cared about me personally, so they put in a lot of time and effort to deliver honest critical evaluations of my work and my decisions. Sylvain Paris, Michael Rubinstein, Kevin Chen and Michael Bernstein kindly talked with me in several occasions. Tianfan Xue, Yichang Shih and Donglai Wei have been helping me to co-review and discuss about many papers. And Zoya Gavrilov, Jenny Yuen, Dahua Lin, Hamed Pirsiavash, Agata Lapedriza, Ce Liu and Carl Vondrick are always fun to hang out with. I also owe much to innumerable friends and students at MIT and elsewhere, for whom I have great respect and admiration.

I thank many senior researchers in the computer vision community that gave me great advice and feedback during my Ph.D., including Serge Belongie, Steve Seitz, Marc Pollefeys, Yair Weiss, Noah Snavely, James Hays, Richard Szeliski, Sven Dickinson, David Kriegman, J. K. Aggarwal, Rob Fergus, Yann LeCun, Derek Hoiem, Song-chun Zhu, Ashutosh Saxena, Jean Ponce, Larry Zitnick, Larry Davis, Yiannis Aloimonos, David Jacobs, Maneesh Agrawala, Sameer Agarwal, David Forsyth, Svetlana Lazebnik, Sing Bing Kang, Deva Ramanan, Honglak Lee, Jianbo Shi, Yanxi Liu, Zhengyou Zhang, Fei-Fei Li, Silvio Savarese, Adam Tauman Kalai, Carlos Hernández, David Gallup, and many others.

I want to thank all funding agencies to support my research directly or indirectly. Especially, Google Research has kindly granted me their prestigious fellowship in computer vision to support my graduate study.

Finally, I thank my parents, my sister, my wife and my sons, for their many years of love and encouragement.

To my sons, Jesse and Jack.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ability to understand a 3D scene depicted in a static 2D image goes to the very heart of the computer vision problem. By "scene" we mean a place within which a person can act, or a place to which a person could navigate. Scene understanding is the gateway to many of our most valued behaviors, such as navigation, recognition, manipulation, and reasoning about the world around us. Therefore, it is of great value to build computer systems that understand visual scenes, both inferring the semantics and extracting 3D structure for a large variety of environments. Scene understanding is the holy grail of computer vision and has been a central topic of research since the field was first established 50 years ago [138].

In this thesis, I summarize my contributions to scene understanding. In Chapter 2, we will talk about the Scene Understanding (SUN) database, a large collection of images that exhaustively spans most scene categories. In Chapter 4, we discuss 3D scene understanding, using predefined geometric primitives. Furthermore, while the world is 3D, our eyes see it from a particular viewpoint. In Chapter 3, we discuss about viewpoint variation in 3D scenes. In Chapter 5, we will talk about the novel "place-centric" representation that integrates scene understanding over space using multiple views.

There are ten statements of the philosophy that have motivated my research in this thesis. In the rest of this chapter, I will present these statements, each in one section, in an order that is easy to understand. But these statements are logically independent dimensions for the thesis, and they provide different perspectives for big thinking behind.

1.1 Turing test: basic level scene understanding

What does it mean to *understand* a scene? There is no universal answer since it heavily depends on the task involved, and this seemingly simple question hides a lot of complexity. The most popular view in the current computer vision literature is to name the scene and objects present in an image. However, this level of understanding is rather superficial. If we can reason about a larger variety of semantic properties and structures of scenes it will enable many important applications. Furthermore, working on an over-simplified task may distract us from exploiting the natural structures of the problem (*e.g.* relationships between objects and 3D surfaces or the relationship between scene attributes and object presence), which may be critical to solving the scene understanding problem.

What is the ultimate goal of computational scene understanding? A natural goal is to pass the **Turing test for scene understanding**: Given an image depicting a static scene, a human judge will ask a human or a machine questions about the picture. If the judge cannot reliably tell the machine from the human, the machine is said to have passed the test. This task is beyond the current state of the art as humans could ask a huge variety of meaningful visual questions about an image, e.g. Is it safe to cross this road? Who ate the last cupcake? Is this a fun place to vacation? Are these people frustrated? Where can I set these groceries? etc.

Therefore, we propose a set of goals that are suitable for the current state of research in computer vision. They should be neither too simplistic nor too challenging and should lead to a natural representation of scenes. Based on these considerations, we define the task of scene understanding as predicting the scene category, scene attributes, the 3D enclosure of the space, and all the objects in the images. For each object, we want to know its category and 3D bounding box, as well as its 3D orientation relative to the scene. Since an image is a viewer-centric observation of the space, we also want to recover the camera parameters, such as observer viewpoint and field of view. Beyond an individual snapshot, we also need to be able to link the current snapshot to the entire environment in a nearby area. We call this set of tasks **basic level scene understanding**, with analogy to *basic level* in cognitive categorization [142]. It has practical applications for providing sufficient information for simple interaction with the scene, such as navigation, object manipulation, and intuitive physics reasoning.

In this thesis we discuss several aspects of basic level scene understanding, including semantics (Chapter 2), 3D structure (Chapter 4), viewpoint (Chapter 3), and space understanding for the entire place aggregated from multiple views (Chapter 5).

1.2 Big data vs. rich data

Ubiquitous access to image datasets has been responsible for much of the recent progress in object recognition [134] after decades of proverbial wandering in the desert.

– Torralba and Efros [170]

Traditionally, researchers focused on developing better, more powerful algorithms, while the data used by these algorithms was considered to be relatively unimportant. Recently, this view has reversed: researchers have come to realize that the *data*, not the algorithm, is the key [134, 170]. Data has been responsible for recent progress in visual recognition [170]. For example, face detection has been a great success for computer vision, but it was the availability of face training data – more than the perceived advances in algorithm design – that produced the first breakthrough [170]. As a community, what we have learned is that instead of starting with a model, we should start with the data and find a model that can realistically represent the data.

Meanwhile, we are in the midst of a big data revolution, especially big visual data. Every minute, 48 hours of video are uploaded to YouTube, and streaming video accounts for 45% of total Internet traffic. Furthermore, bigger visual data is coming. In the near future, every person will wear a camera (e.g. Google Glass) recording every minute of their lives. Every autonomous car will have a camera recording the street, and inside every home there will be a Kinect-style camera supporting touch-less interaction with digital devices. We are going to have a lot more visual data, far more than what a person sees in a lifetime through their eyes.

Big visual data had brought a revolution to computer vision research, such as scene completion using Flickr images [65] or deep learning algorithms trained on YouTube videos [29]. But what is next? Beyond the number of data points, this thesis argues for another dimension of big data: the *richness* of data. By "rich data", we mean data that has a rich description, such as annotation and extra sensor input associated with it. For example, while websites like Google, Flickr and Facebook have billions of images, these images are not rich because they have very little extra information associated with them. The poorness of the disconnected data makes computer vision still mostly unsolvable.

A picture is worth a thousand words, not just one word. In the current stage, researchers typically regard visual recognition as an image classification process (either on the whole image or a region of an image). But there is something very wrong with this image classification framework. The goal of computer vision is not just to guess for an integer number, the category label, or just a single word. Over-simplifying the problem just means that we are not researching towards the right direction. Instead, the major goal of computer vision is to study the right representation [77], and the associated algorithm to convert images into this representation of the world. Note that it is the representation of the world. It is not to study the representation for an image. Some people call scene recognition as image understanding. But "image understanding" is very a misleading name. An image is just a 2D array. Computer vision is not about how to understand that boring 2D array. But is about how to understand the beautiful world out there through the image. The representation of the world is very rich. While an image is a 2D array, the world is 3D and our eyes see it from a viewpoint, but this is not traditionally modeled in the current recognition pipeline.

The 3D issue and the viewpoint issues require a much richer representation for the 3D world. But *rich representation* needs *rich data* for training. The kind of data directly constrains the kind of representation that we can use. For example, for children to develop a visual understanding of the world, they don't just sit still and keep watching, but instead, they keep actively exploring everywhere. James Gibson proposed the popular "actively exploring organism" theory [58]: The world of the babies is not an imagined world, constructed out of his ideas and physical stimuli, but a livable world, made up of interesting objects; It is a habitat to be seen and explored, in which even a child can learn about what things are like, not a world that stimulates him to react and then forces him to infer what things must be like. Essentially, active exploration provides very rich and connected data to learn a rich representation of the world.

In this thesis, we study how to construct rich visual dataset and how to use these rich visual data for scene understanding. The underlying principle is that exploring and learning from these rich descriptions of data will be crucial for finding the right representation to close the performance gap between computer and human vision systems.

1.3 Object-centric vs. scene-centric dataset

Our visual world is extraordinarily complex, and this makes it difficult for computers to understand scenes. For example, one of the most basic tasks of scene understanding is to classify a natural image into one of many semantic categories. What are these scene categories? From a human-centric perspective, scene categories should capture the richness and diversity of environments that make up our daily experiences. Although the visual world is continuous, most environmental scenes are visual entities that can be organized in functional and semantic groups. Particular scenes or places may allow for specific actions, such as eating in a restaurant or sleeping in a bedroom. To capture this diversity, we constructed an exhaustive taxonomy of scene categories and a dataset representing the diversity of scenes that are encountered in the real world. We used an English dictionary and manually selected all of the terms that describe scenes, places, and environments. We collected images belonging to each scene category in the above list using various image search engines, manually checked them for accuracy, and named this large image collection the SUN (Scene UNderstanding) database [184]. To provide data for research and natural statistics of objects in scenes, we have also labeled objects in a large portion of the image collection with polygon outlines and object category names. To date, there are 326,582 manually segmented objects for the 5,650 object categories labeled. This database allows us to systematically study the space of visual scenes commonly encountered by humans as well as to establish a benchmark for both scene and object recognition.

Note that the way we construct the SUN database determines that it is a scene-centric

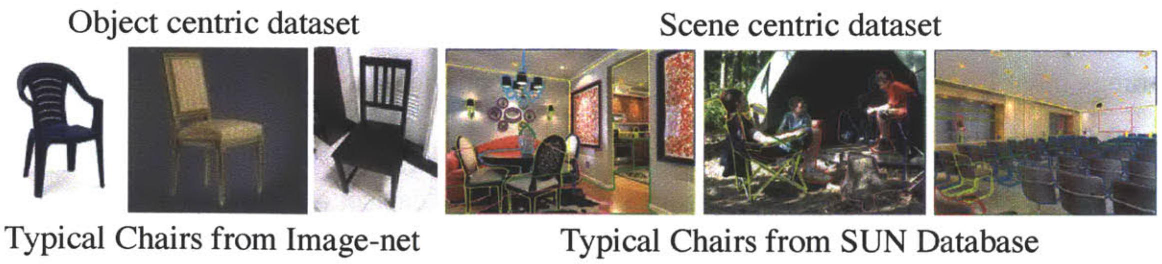

Object centric dataset
Scene centric dataset
Typical Chairs from Image-net
Typical Chairs from SUN Database

Figure 1-1: An example to highlight the difference between an object-centric dataset and a scene-centric dataset.

dataset. Alternatively, an object-centric dataset, *e.g.* Image-net [32], constructed by querying object category names from image search engine usually produces images with very different statistics. Querying object category names tend to return product-like images from the Internet, as shown in Figure 1-1. The images from an object-centric database are typically biased in the following ways that a scene-centric database wouldn't suffer from:

- Location Bias: The object is usually at the center of the image.
- Size Bias: The object mostly covers the whole image.
- Number Bias: Usually there is one instance only (or one pair of shoes).
- Context Bias: Only one object in the image, and there is not context relationship with other objects, such as a person sitting on a chair.
- Occlusion Bias: There is no occlusion, except self-occlusion.
- Background Bias: A lot of pictures have single color background or very synthetic background from photo studio.
- Lighting Bias: To have an attractive photo for a product, they tend to have certain lighting restrictions.
- Viewpoint Bias: The canonical view of the object is typically used.

The object-centric database could be very useful for training systems to automatically recognize products for online shopping business. But it is naturally ill suited for real-world

	Spatial Understanding	Semantic Understanding
Traditional Name	Reconstruction	Recognition
High Level	Shape Understanding	Semantic Perception
	↕	↕
Low Level	Depth Measurement	Pixel Color

Figure 1-2: The gap between low-level depth measurement and high-level shape understanding, is just as huge as the gap between low-level pixel-color and high level semantic percetpion.

object recognition and scene understanding. A scene-centric database is definitely not bias-free, but it is likely less biased in all of these ways, thanks to avoiding direct query of object categories.

Obsessive focus on research using only individual objects outside scene context may also hinder the progress of computer vision[1], because it is not a natural way that human and animal use to develop their biological system during evolution. There is no known working biological vision system that can evolve (analogous to parameter learning in computer vision) from only seeing individual objects. It is important to use data that is closer to the natural input that biological vision systems have, and ensure that there actually exists a working solution for the vision learning process.

1.4 3D reconstruction is not just a low-level task

Although an image is a 2D array, we live in a 3D world where scenes have volume, affordances, and are spatially arranged with objects occluding each other. The ability to reason about these 3D properties would be useful for tasks such as navigation and object manipulation. The dominant research focus for 3D reconstruction is in obtaining more accurate depth maps or 3D point clouds. However, even with a depth map, we are still unable to manipulate an object because there is no high-level representation of the 3D world. Es-

[1]in a way that is not easy to tell

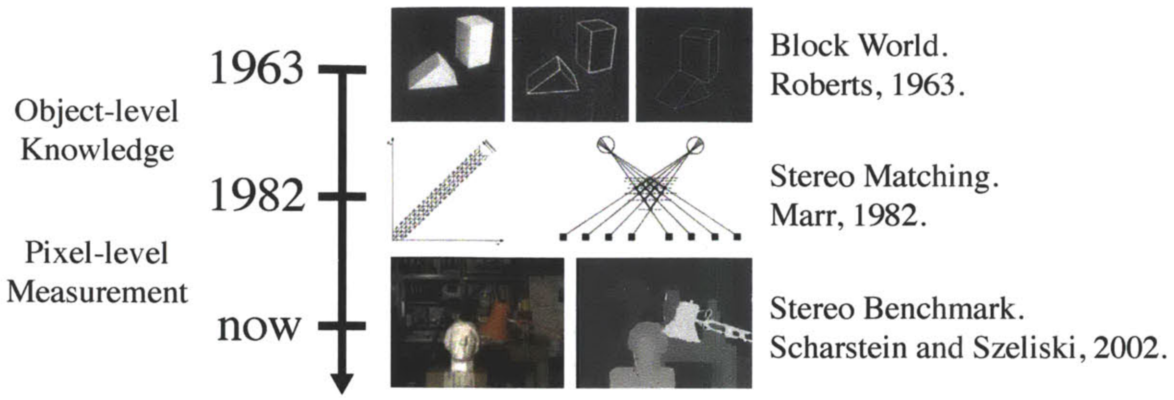

1963
Object-level Knowledge
1982
Pixel-level Measurement
now
Block World. Roberts, 1963.
Stereo Matching. Marr, 1982.
Stereo Benchmark. Scharstein and Szeliski, 2002.

Figure 1-3: 50 years of 3D Reconstructions (from 1963-2013).

sentially, 3D reconstruction is not just a low-level task. Obtaining a depth map to capture a distance at each pixel is analogous to inventing a digital camera that captures the color value at each pixel. The gap between low-level depth measurements and high-level shape understanding is just as large as the gap between pixel colors and high-level semantic perception. Moving forward, we need a higher-level intelligence for 3D reconstruction.

Historical perspective Physics (radiometry, optics, and sensor design) and computer graphics study the forward models about how light reflects off objects' surfaces, is scattered by the atmosphere, refracted through camera lenses (or human eyes), and finally projected onto a 2D image plane. In computer vision, we are trying to do the inverse [165], *i.e.* to describe the world that we see in one or more images and to reconstruct its properties, such as shape. In fact, the desire to recover the three-dimensional structure of the world from images and to use this as a stepping stone towards full scene understanding is what distinguished computer vision from the already existing field of digital image processing 50 years ago [138, 165].

Early attempts at 3D reconstruction involved extracting edges and then inferring the 3D structure of an object or a "blocks world" from the topological structure of the 2D lines [138]. Staring from late 70s, more quantitative approaches to 3D were starting to emerge, including the first of many feature-based stereo correspondence algorithms [33, 115, 63, 116], and simultaneously recovering 3D structure and camera motion [174, 175, 112], *i.e.* structure from motion. After three decades of active research, nowadays, we can achieve very good performance with high accuracy and robustness, for both stereo matching [148, 150, 186] and structure from motion [159, 160, 189, 166].

However, there is a significant difference between these two groups of approaches. The first group represented by "block world", focuses on high-level reconstruction of objects and scenes. The second group, *i.e.* stereo correspondence and structure from motion, targets on very low-level 3D reconstruction. For example, the introduction of structure from motion was inspired by "the remarkable fact that this interpretation requires neither familiarity with, nor recognition of, the viewed objects" from [175]. It was totally aware that this kind of 3D reconstruction at low level is just a milestone towards higher-level 3D understanding, and is not the end goal.

However, this message somehow got mostly lost in the course of developing better-performing systems. In the past three decades, there are a lot more success we achieve for the low-level 3D reconstruction for stereo correspondence and structure from motion, than for the high level 3D understanding. For low-level 3D reconstruction, thanks to the better understanding of geometry, more realistic image features, more sophisticated optimization routines and faster computers, we can obtain a reliable depth map or 3D point cloud together with camera poses. In contrast, for higher-level 3D interpretation, because the line-based approaches hardly work for real images, this field diminished after a short burst. Nowadays, the research for 3D reconstruction almost exclusively focuses on only low-level reconstruction, in obtaining better accuracy and improving robustness for stereo matching and structure from motion. Most researchers seems to have forgotten the end goal of such low-level reconstruction, *i.e.* to reach a full interpretation of the scenes and objects . Given that we can obtain very good result on low-level reconstruction now, it is important to put greater emphasis on mid-level and high-level 3D understanding.

We should consider an algorithm and a task that an algorithm is solving separately. The failure of line-based approach for high-level 3D understanding should only indicate that we need a better algorithm. It shouldn't mean that higher-level 3D understanding is not important and we can stop working on it. In other words, we should focus on designing better approaches for high-level 3D understanding, which is independent of the fact that line based approach is less successful than key-point and feature based approach.

In fact, the term "reconstruction" is very misleading. "Reconstruction" makes 3D understanding sound like reconstructing a signal from sources in engineering, which is in general closer to low-level computer vision[2]. In long term, this name misled researchers' thinking subconsciously, and I suggest to abandon this name completely, replaced by more reasonable names, *e.g.* "spatial scene understanding".

1.5 Semantic vs. spatial scene understanding

There are roughly two kind of understanding in scene recognition process: semantic understanding and spatial understanding. Traditionally, the semantic scene understanding has been mostly formulated as scene categorization, object detection and segmentation. The spatial scene understanding is mostly limited to mostly about a point cloud or surface reconstruction. Recently, the volumetric 3D reasoning of indoor layout marked the beginning of 3D reconstruction beyond low level. Yu *et al.* [198] inferred the 3D spatial layout from a single 2D image by grouping: edges are grouped into lines, quadrilaterals, and finally depth-ordered planes. Because it aimed to infer the layout of a room, it is forced to reason about the 3D structure beyond low level. Since then, several groups independently started working on 3D geometric reasoning.

In parallel to the computer vision researchers' effort to develop engineering solutions for recovering the three-dimensional shape of objects in imagery, perceptual psychologists have spent centuries trying to understand how the human visual system works. The *two-streams* hypothesis is a widely accepted and influential model of the neural processing of vision [46]. The hypothesis, given its most popular characterization in [59], argues that humans possess two distinct visual systems. As visual information exits the occipital lobe, it follows two main pathways, or "streams". The ventral stream (also known as the "what pathway") travels to the temporal lobe and is involved with object identification and recognition. The dorsal stream (or, "how pathway") terminates in the parietal lobe and is involved with processing the objects spatial location relevant to the viewer.

[2] *i.e.* image in and image out (*e.g.* image debluring and denoising), or images in and point cloud out (stereo depth map).

Human vision	Computer vision	Low Level	Mid Level	High Level
Ventral stream	Recognition	Color value	Grouping & Alignment	Semantic $\rightarrow$ Context
Dorsal stream	Reconstruction	Distance value	Grouping & Alignment	Shape $\rightarrow$ Structure
Question to answer at each level		How to process signal?	Which are together?	What is where?

Table 1.1: Different levels and different streams for both human and computer vision systems.

The two-streams hypothesis remarkably matched well with the two major branches of computer vision – recognition and reconstruction. The ventral stream is associated with object recognition and form representation, which is the major research topic for recognition in computer vision. On the other hand, the dorsal stream is proposed to be involved in the guidance of actions and recognizing where objects are in space. Also known as the parietal stream, the "where" stream, this pathway seems to be a great counterpart of reconstruction in computer vision.

The two-steams hypothesis in human vision is the result of study of human brain. But the distinction of recognition and reconstruction in computer vision rise automatically from the researchers in the field without much awareness. The computer vision researchers naturally separate the vision task into such two major branches, based on the nature of the tasks, at the computational theory level.

This interesting coincidence enables us to make further analysis of the research focuses in computer vision. For recognition, *i.e.* counterpart of ventral stream, it is widely accepted that the task can be divided into three levels, as shown in Table 1.1. However, there is not separation of the three levels for reconstruction, simply because the current research of reconstruction exclusively focus on the low level part only. The mid level and high level for 3D reconstruction are mostly ignored. A large portion of researchers is not aware of the existing of the problem.

Now, with this analogy between human vision and computer vision, we can now try to answer what are the core tasks of three different levels of reconstruction. Since both ventral and dorsal stream start from the primary visual cortex (V1), we can expect that the low level task for reconstruction should be signal processing and basic feature extraction, such as V1-like features and convolution of Gabor-like filter bank, or time-sensitive filter bank for motion detection to infer the structure. The mid level focuses on grouping and alignment. By grouping, we mean the grouping of pixels within the current frame for either color of depth value, *i.e.* the segmentation of the image plane into meaningful areas. This can happen in both 2D and 3D [196, 185]. By alignment, we mean the matching of the current input with previous exposed visual experience, *e.g.* as matching of a local patch with patches in a training set [155]. The grouping happens within the current frame, and the alignment happens between the current frame and previous visual experience. In both cases, the fundamental computational task for this level is to answer "which are together?" For the high level of recognition, the task is to infer the semantic meaning, *i.e.* the categories of objects, and furthermore, the context of multiple objects in the scene. For the high level of reconstruction, the task is to recognize the shape of individual objects, and to understand the 3D structure of the scene, *i.e.* the spatial relationship of objects in the scene (a shape is on top of another shape). At the end of computation, together with both recognition and reconstruction, or ventral stream and dorsal stream, the vision system will produce answers for "what is where?"

1.6 Reintroducing geometric figures using modern machinery

Since the very beginning of computer vision, geometric figures started to play an important role. Around 1956, Marvin Minsky and John McCarthy started computer vision with a simplification of the world[3], the so-called "block world", so that the mathematical models can apply rigorously and to solve the resulting recognition problem completely before proceeding to more difficult situations. Roberts's PhD thesis [138] represented the most complete and powerful recognition system of the blocks world during the stage, by careful consideration of how polyhedra project into perspective images and established a generic library of polyhedral components that could be assembled into a composite structure.

The next major advance in representations for recognition was the generalized cylinder

[3] According to one well-known story [165], in 1966, Marvin Minsky at MIT asked his undergraduate student Gerald Jay Sussman to "spend the summer linking a camera to a computer and getting the computer to describe what it saw".

(GC) originated by Thomas Binford [6]. The key insight is that many curved shapes can be expressed as a sweep of a variable cross section along a curved axis. The generalized cylinder theory is then followed up by Biederman's Recognition-By-Components theory [17]. In his theory, geons are the simple 2D or 3D forms such as cylinders, bricks, wedges, cones, circles and rectangles corresponding to the simple parts of an object. The theory proposes that the visual input is matched against structural representations of objects. These structural representations consist of geons and their relations, and only a modest number of geons are assumed. When combined in different relations to each other (*e.g.* , on-top-of, larger-than, end-to-end, end-to-middle) and coarse metric variation such as aspect ratio and 2D orientation, billions of possible 2- and 3-geon objects can be generated. While these approaches have achieved notable early successes, they could not be scaled-up due to their heavy dependence on reliable contour extraction from natural images. And the field is mostly abandoned after three decades.

This thesis aims to reintroduce geometric figures into computer vision using modern machinery. To make the problem simpler, we focus on the most common and simplest geometric figure: a 3D rectangular cuboid.

First of all, we use modern appearance learning and part-based models to build a 3D cuboid detector (Section 4.1) to recognize rectangular cuboids and localize their corners from a single, real-world photograph. This algorithm can also be naturally extended to cylinders, pyramids, and other shapes. Beyond a single cuboid, we propose a context model to reason about the interaction between multiple cuboids in 3D (Section 4.2). To obtain a global optimal solution of cuboid selection configuration, a branch and bound algorithm is used to solve the formulated mixed integer linear programming. Beyond one view and one cuboid, we also make use of a combination of several 3D cuboids to represent the complete 3D shape of the environment from the observations aggregated from multiple views. Therefore, we propose an algorithm called "InverseCSG" (Section 5.7) to greedily add or subtract a 3D cuboid from the current solution to approximate the shape of the entire space.

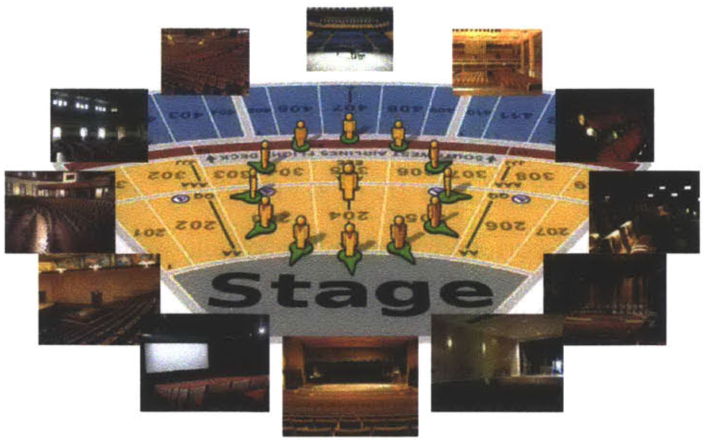


Figure 1-4: Scene Viewpoint [187].

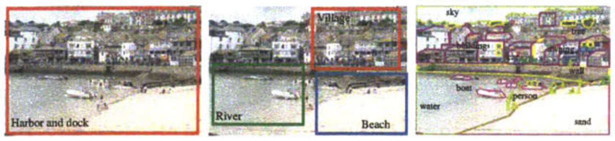
1.7 View-based vs. place-centric scene understanding

Besides 3D, another key issue for scene understanding is the viewpoint of the observers. Within a particular scene, the meaning and functions vary considerably with viewpoint. For instance, as shown in Figure 1-4, a theater has a distinct distribution of objects – a stage on one side and seats on the other – which define unique views at different orientations. Just as observers will choose a view of a television that allows them to see the screen, observers in a theater will sit facing the stage when watching a show. In [187], we propose an algorithm for the problem of *scene viewpoint recognition*, the goal of which is to classify the type of place shown in a photo, *and also* to recognize the observer's viewpoint within that place, e.g. facing the theater's stage.

Besides different viewpoints, there may also be several sub-scenes in a given view. For example, as shown in Figure 1-5 (a) and (b), there can be very different levels of scene understanding even for a single image. As shown in Figure 1-5 (b), in contrast to a holistic scene categorization on the left or a detailed object segmentation on the right, we propose an intermediate sub-scene detector [195] to localize bounded regions of the environment that have a distinct functionality with respect to the rest, such as for identifying the river in order to go swimming.

Furthermore, when a human being navigates an environment, the visual input is basically a video. But humans do not understand the scene as many individual disconnected snapshots from a video. We understand the environment as an integrated space. Essentially,

(a) These scenes of a beach, a village, and a river are all from a single image. They have totally different semantic meanings and functions



(b) There are many complementary levels of image understanding.

Figure 1-5: Scene Detection [195].

to jointly reason about 3D structure and viewpoint in space, we should have a "place-centric" representation of the 3D space around the observer. However, the SUN database is view-based, in the sense that it only contains snapshots that capture particular views but not the full 3D extent of a place. We desire a "place-centric" representation, *i.e.* one that has a comprehensive model of the entire 3D space as its representation rather than a limited set of views. To study this kind of representation, we need a database that allows us to model the 3D context of objects in space, to reason about mechanics and intuitive physics, and to answer questions such as: "what does this object look like from behind?".

A place-centric description of a scene is more complete and largely resembles the real world, providing the strong advantage of being more invariant to viewpoint changes. Therefore, we introduced the SUN3D database of full environments that were scanned with an RGBD sensor, by a person who walked and mimicked human exploration, thoroughly covering natural viewpoints. We designed a robust structure-from-motion pipeline for long RGBD sequences to produce full 3D models of entire houses, offices, *etc.* However, existing methods of structure from motion that use RGBD cameras often fail when trying to reconstruct large places. Therefore, we introduce a bundle adjustment algorithm that incorporates semantic labels introduced by user to obtain an accurate 3D reconstruction of large places scanned with an RGBD camera, and to provide object segmentations and labels for all the objects in the environment. The user only needs to label some frames from the video using a LabelMe [144] style annotation tool. The method will propagate the labels to all the other frames and use the annotations to help constraint the 3D reconstruction.

Going beyond a view to a whole environment, not only can reconstruction improve recognition, but *recognition can also improve reconstruction.* We designed a 3D reconstruction algorithm [191] that represents a complete environment by a combination of volumetric primitives, which are recognized jointly by a bottom-up and top-down inference. This new place-centric representation goes beyond low-level 3D reconstruction and imposes powerful global regularization constraints that exploit structural regularities. This representation can be used to produce photorealistic maps for large indoor environments. We used our system to create human navigation tools for various museums, including the Metropolitan Museum of Art in New York City – one of the biggest art galleries in the world.

1.8 Don't forget the "computer" in "computer vision"

The vast quantity of experimental research in the textbooks and handbooks is concerned with snapshot vision, fixed eye vision, or aperture vision and it is not relevant.

– James J. Gibson [58]

Currently, most computer vision researchers tend to forget about the role of "computer" in computer vision, in the sense that they only use computer as a computational device, but forgetting about the role of computer as an agent in the scene. Just like James Gibson criticized the human vision research in the 80s, when psychologists study human vision, they just let the human sit there and keep watching a lot of snapshot pictures. But this is not a natural setting that the way human vision is developed, and we are suffering from exactly the same problem now in computer vision. Almost all research in computer vision is devoted into understanding snapshot pictures from the Internet and forgets about the role that computer should play. We just show the computer a set of random pictures and force the computer to make sense of it. For example, we don't consider the viewpoint or allow them to move, and there is no place centric representation of the space. We are simply too harsh to the poor computer. Gibson proposed the ecological approach [58], which is valued by many famous psychologists as the Newton theory in human vision [136]. If we want a rich representation of the scene for computer vision, we need to change our data gathering method to make computer become an active explorer and provide continuous multiple view information as well.

Note that I am not arguing that a real-time online robotics vision system is the only way to support this claim and save computer vision. I am saying that we should be aware of this problem, and consider more about the implications of this problem. We should model the viewpoint of the observer and change the underlaying scene representation from view-based 2D snapshots to place-entric integrated representation. Real-time online robotics vision is interesting. But it adds a lot overhead and extra constraints for research and prototype engineering, which may require unrealistic amount of computation at the current stage, and hence force the research to go towards a different path – acceleration or simplification of exisiting algorithms[4], while I am talking about designing new algorithms. But in the future, I am optimistic that we will find the right trade-off to do this effectively.

1.9 Going beyond aperture computer vision

The approximate field of view of an individual human eye is 95° away from the nose, 75° downward, 60° toward the nose, and 60° upward, allowing humans to have an almost 180-degree forward-facing horizontal field of view. With eyeball rotation of about 90° (head rotation excluded, peripheral vision included), horizontal field of view is as high as 270°.

However, the available field of view for the digital cameras commonly used in computer vision is typically 54.4° horizontally and 37.8° vertically (full-frame 35mm digital camera with focal length 35mm), which is significantly smaller than human eyes.

On the other hand, Ehinger [40] shows that peripheral vision is a very important part of scene reorientation. More specifically, human behavioral study [40] shows that it is actually easier for human to reorient in a scene with only peripheral visual information than it is with only central visual information. This demonstrates the great importance of having a large field of view for human vision system.

[4]In the current robotic research community, time and energy are hard constraints for robots, and they are very popular research topics.

Therefore, the very small field of view of cameras is very likely to be one of the main reasons for poor performance computer vision system. It is unfair to compare human vision and computer vision performance while they are having a significantly different field of view. Again, we are too harsh to the poor computer. Moving forward, we need to go beyond aperture computer vision, to make larger field of view images to be the main images used in computer vision systems. To this end, in this thesis, we constructed a 360° full-view panorama database for training computer vision systems (Section 3) to recognize scene viewpoint and dramatically extrapolate the available field of view.

1.10 Post-Internet computer vision

In the past decade, the Internet has become an increasingly massive, interesting, and useful source of imagery for computer vision research. Image available on the Internet has become a major driven-force for computer vision algorithms. For example, Snavely *et al.* [159] is one of the first to use distributed images available online to produce a massive 3D reconstruction of the world's popular sites. Furthermore, many computer vision datasets are constructed from Internet images, such as PASCAL VOC [44], Tiny Image [168] and Image-Net, as well as SUN database. Many unsupervised object recognition systems are also developed to automatically segment, co-segment and recognize common objects from massive online image collection. Very recently, deep-learning algorithms (*e.g.* [29]) has been demonstrated to be particularly suitable for using images from the Internet.

However, as mentioned in the previous sections, there are many limitations to use Internet photos and videos, which are not captured for the purpose of training computer vision algorithms. On the other hand, great popularity of affordable depth sensors, such as Microsoft Kinect, Asus Xtion and PrimeSense, makes depth acquisition very easy. More and more datasets, such as the SUN3D database (Chapter 2) and the NYU dataset [154], are captured by computer vision researchers. As the field progresses, the limitation of Internet images are going to become the major bottleneck. The post-Internet computer vision is emerging. The sensor capturing and vision algorithms are going to get closer and closer.

Chapter 2

Scene Understanding Database

Scene understanding is the gateway to many of our most valued behaviors, such as navigation, recognition, and reasoning with the world around us. By "scene" we mean a place within which a person can act, or a place to which a person could navigate. How many kinds of scenes are there? In this chapter we hope to address many questions about the "space of scenes" such as: How can scene categories be organized? Are some exemplars better than others? Do scenes co-occur in images? How do the current state-of-the-art scene models perform on hundreds of scene categories encountered by humans, and how do computational models compare to human judgments about scenes?

Given that most of the places we interact in are built by and for people, the number of scene classes or partitions one can make of the world is in constant evolution: there may be finer-grained categories emerging from economical or functional constraints (e.g., compact apartment) or categories with only one exemplar (e.g., a specific space station) [22]. Despite this variability, there are a core number of places that most people encounter in the world, that form the basis of categorical knowledge for the field of scene understanding. The list proposed here represents a lower bound on the number of places that can be named.

·Whereas most computational work on scene and place recognition has used a limited number of semantic categories, representing typical indoor and outdoor settings [104, 48,

The materials presented in this chaper are based on [184, 188] in collaboration with Antonio Torralba, Aude Oliva, Krista A. Ehinger, and James Hays.

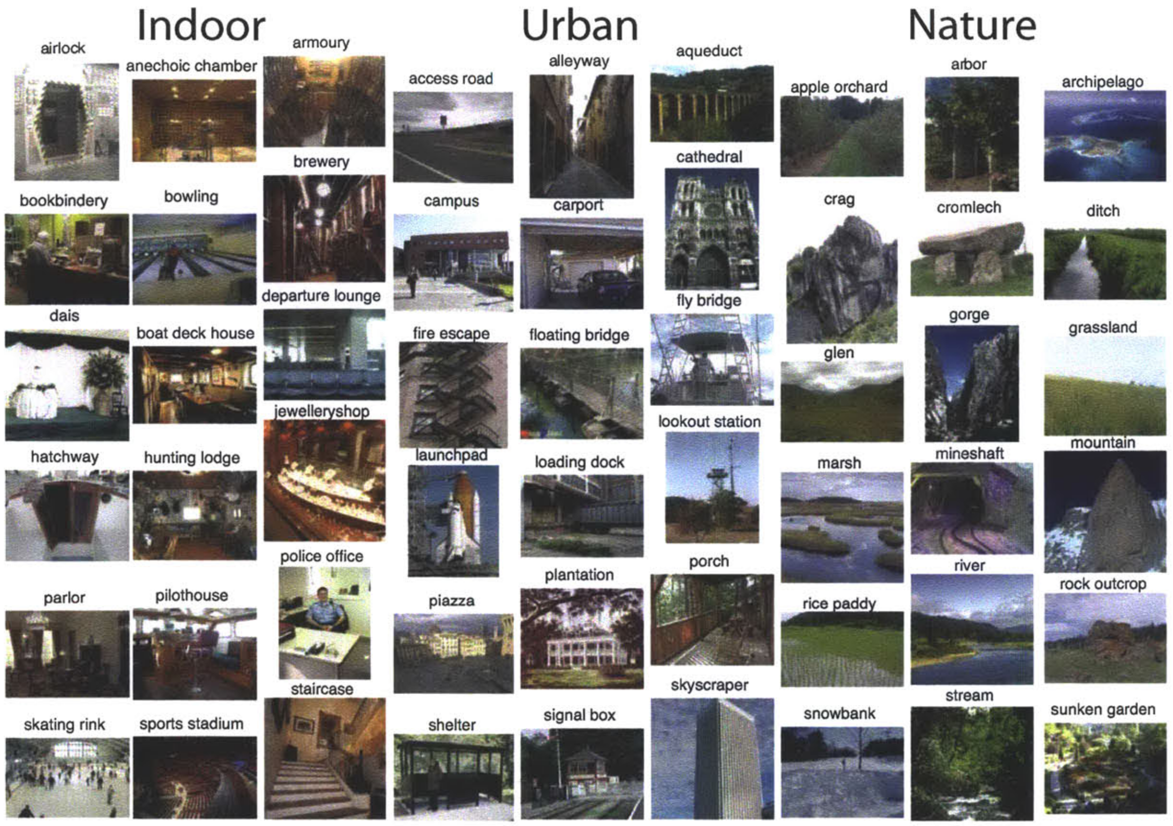


Figure 2-1: Examples of scene categories in our dataset.

137, 179, 126, 14, 169], access to large quantities of images on the Internet now makes it possible to build comprehensive datasets of images organized in categories [60, 168, 32] in order to capture the richness and diversity of environments that make up our daily experience.

Although the visual world is continuous, most environmental scenes, like objects, are visual entities that can be organized in functional and semantic groups. Like objects, particular environments will trigger specific actions, such as eating in a restaurant, drinking in a pub, reading in a library, and sleeping in a bedroom. However, when faced with environments from a given basic-level semantic category (e.g. kitchen), people may behave differently and have different expectations depending on the specifics of the place (e.g. a house kitchen, a restaurant kitchen, an industrial kitchen). Therefore, it is critical for artificial vision systems to discriminate the type of environments at the same level of specificity as humans. Here, we provide a quasi-exhaustive taxonomy and dataset representing the diversity of visual scene categories that can be encountered in the world and we provide computational benchmarks for large-scale scene categorization tasks.

This chapter has the following four objectives. First, we propose a method to quasi-exhaustively determine the number of different scene categories. We identify all the scenes and places that are important enough to have unique identities in discourse, and build a large-scale dataset of scene image categories. Second, we perform experiments to measure how accurately humans can classify exemplars of scenes into hundreds of categories, how "typical" particular scenes are of their assigned scene category, and how scene categories relate in terms of high-level semantic properties. Third, we evaluate the scene recognition and indoor vs outdoor classification on this large-scale scene database using a combination of many image features. Finally, we introduce the scene detection task with the goal of determining which scene categories are present in local image regions. Where appropriate, we explore the relationship between human experiments and machine performance.

2.1 Building the SUN database

In this section we describe our procedure to build a large-scale database of scenes. We provide a rough estimate of the number of common scene types that exist in the visual world and build an extensive image database to cover as many of these as possible. We refer to this dataset as the *SUN* (Scene UNderstanding) database[1] [184].

In order to define a list of scene categories, we follow a procedure similar to Biederman's [17] process for determining the number of objects by counting object names in dictionary. Here, we used WordNet [49], an electronic dictionary of the English language containing more than 100,000 words. We first selected the 70,000 words that correspond to non-abstract terms and that are available in the tiny images dataset [168] . We then manually selected all of the terms that described scenes, places and environments (any concrete noun which could reasonably complete the phrase "I am in a *place*", or "Let's go to the *place*"). Most of the terms referred to basic and entry level places [173, 140, 141, 91]. In reference to visual scenes, these entry-level terms would refer to a set of environments that share visual similarities and objects, which may lead to similar motor interactions and intended actions. We did not include specific place names (like Grand Canyon or New York) or terms that did not seem to evoke a specific visual identity (territory, workplace, outdoors). Non-navigable scenes (such as desktop) were not included, nor were vehicles (except for views of the inside of vehicles) or scenes with mature content. We included specific types of buildings (skyscraper, house, hangar), because, although these can be seen as objects, they are known to activate scene-processing-related areas in the human brain. [43]. We also maintained a high tolerance for vocabulary terms that may convey significance to experts in particular domains (e.g. a baseball field contains specialized subregions such the pitcher's mound, dugout, and bullpen; a wooded area could be identified as a broadleaf forest, rainforest, or orchard, depending upon its layout and the particular types of plants it contains). To the WordNet collection we added a few categories that seemed like plausible scenes but were missing from WordNet, such as jewelry store and mission.

This gave about 2500 initial scene words, and after manually combining synonyms

[1]All the images and scene definitions are available at `http://sundatabase.mit.edu`.

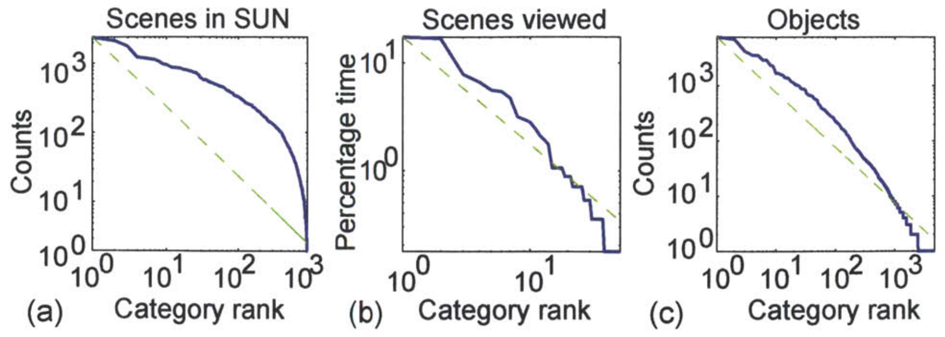


Figure 2-2: (a) Sorted distribution of scene classes in the SUN database. (b) Sorted distribution of scene classes encountered while recording daily visual experience. (c) Sorted distribution of object counts. The dashed line corresponds to the function $A/rank$, where the constant A is the max of each curve.

(provided by WordNet) and separating scenes with different visual identities (such as indoor and outdoor views of churches), the final dataset reaches 908 categories.

It is possible to use a similar procedure to get an estimate of the number of object words in the WordNet database. As with the scenes, we started with the 70,000 non-abstract terms from WordNet. We then selected a random 2% of the words and determined what proportion of these were objects. Including synonyms, there are about 2,500 scene words and about 27,000 object words; that is to say, there are about 10 times as many object words as there are scene words. This difference reflects the fact that there are more subordinate-level terms in the object domain (e.g., names for each individual species of plant and animal) and more synonyms for the same object (an individual species has both a scientific name and one or more common names). What this analysis makes clear is that there are far more words for objects than scenes, and likely more distinct categories of objects than there are distinct categories of scene. Although we can think of scenes as distinct combinations of objects, there are far fewer scene categories than there are possible configurations of objects. This is because not all distinct object configurations give rise to different scene categories, and most scene categories are flexible in terms of their constituent objects (e.g., living rooms can contain many different types of objects in various configurations).

Once we have a list of scenes, the next task is to collect images belonging to each scene category. Since one of our goals is to create a very large collection of images with variability in visual appearance, we collected images available on the Internet using online

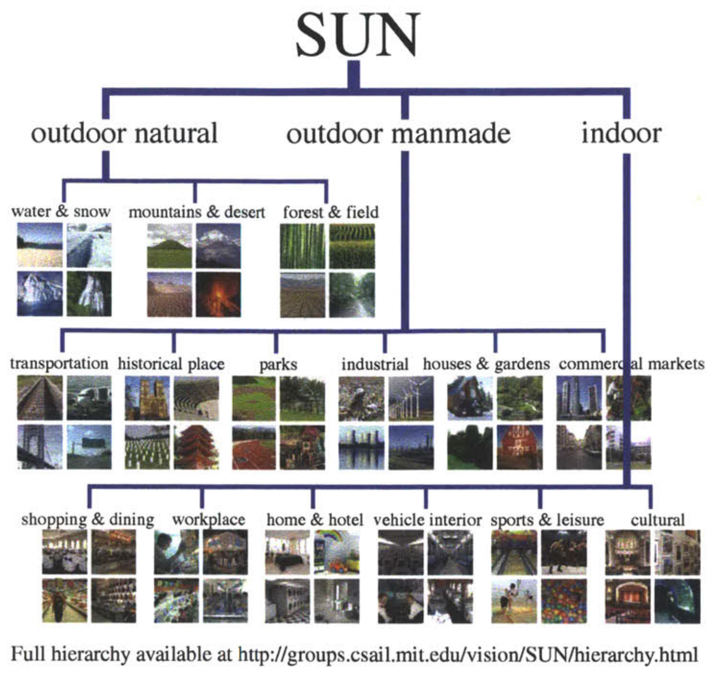


Figure 2-3: The first two levels of a hierarchy of scene categories.

search engines for each scene category term. Similar procedures have been used to create object databases such as Caltech 101 [47], Caltech 256 [60], 80 million images [168] and ImageNet [32].

For each scene category, images were retrieved using a WordNet term from various search engines on the web. When a category had synonyms, images for each term were retrieved and then the images were combined. Only color images of 200×200 pixels or larger were kept. For similar scene categories (e.g. "abbey", "church", and "cathedral") explicit rules were formed to avoid overlapping definitions. Images that were low quality (very blurry or noisy, black-and-white), clearly manipulated (distorted colors, added text or borders, or computer-generated elements) or otherwise unusual (aerial views, incorrectly rotated) were removed. Duplicate images, within and between categories, were removed. Then, a group of participants (N=9, including some of the authors) manually removed all the images that did not correspond to the definition of the scene category.

For many of the 908 SUN categories an image search returns relatively few unique photographs. The success of each search depends upon how common the category is (it is easier to find photographs of living rooms than it is to find pictures of the inside of

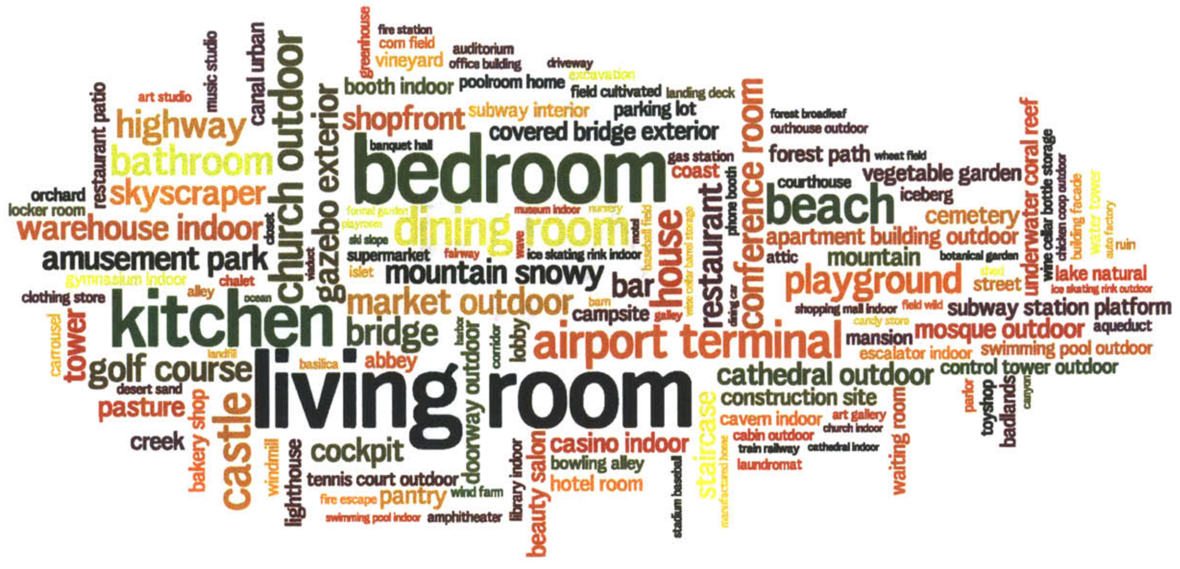


Figure 2-4: Visualization of the scene categories based on the number of images in each category. Bigger font size corresponds to larger number of images in the corresponding categories.

airplanes), how common the category name is and how many synonyms exist (even when people take pictures of the inside of an airplane, they may not label them as "airplane cabin" when posting them on the Internet). As a result, it is more difficult to find images for some scene categories than for others and the distribution of images across scene categories is not uniform. Examples of scene categories with more images are living room, bedroom, and bookstore. Examples of under-sampled categories include airlock, editing room, grotto, launchpad, naval base, oasis, ossuary, salt plain, signal box, sinkhole, sunken garden, and winners circle.

The final dataset contains 908 categories and 131,072 images. Estimating the number of categories that compose a set of items from a finite sample is a challenging task (see [22] for a review). In the case of scene categories, this number might be infinite as there might always be a new, very rare, category with a specific function not considered before. Our dataset is not an exhaustive list of all scene categories, but we expect that the coverage is large enough as to contain most of the categories encountered in everyday life.

Fig. 2-2(a) shows the distribution of the number of images collected for each scene category in the SUN database. The categories are sorted in decreasing order of available images. For the scene classification and detection experiments in this chapter we use *only*

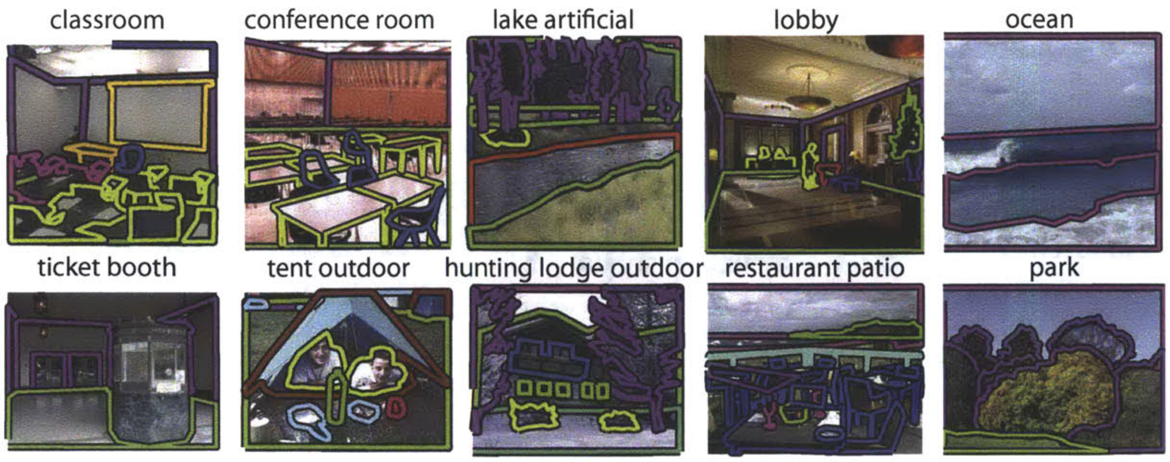


Figure 2-5: Examples from 19,503 fully annotated images in SUN.

the 397 categories for which there are at least 100 unique photographs.

It is important to acknowledge that the procedure used here is not the only way to create a list of scene categories and collect images. There are different ways to define and categorize "scenes", which would generate different organizations of the images, and different categories, than the one used here. For instance, an alternate strategy would be to record the visual experience of an observer and to count the number of different scene categories viewed. We had 7 participants (including 2 of the authors) write down, every 30 minutes, the name of the scene category in which they were located, for a total of 284 hours across participants. During that time, the participants reported a total of 54 distinct places. All the scenes provided were already part of the previous list produced from WordNet which we take as an indication of the completeness of the list provided by WordNet. This procedure is unlikely to produce a complete list of all scene categories, as many scene categories are only viewed on rare occasions (e.g., cloister, corn field, etc.) and would be dependent on the individual daily activities. However, this method would have the advantage of producing a list that would also provide information about the real frequency of environments encountered under normal viewing conditions.

Fig. 2-2(b) shows the sorted distribution of scenes obtained in this way. In this plot, the vertical axis corresponds to the percentage of time spent in each scene type (which is an indication of the number of images that would be collected if we recorded video). Note that the distribution looks quite different from the one in Fig. 2-2(a). For comparison, we also

Figure 2-6: Examples of 12,839 chairs that were manually annotated in SUN.

show the distribution of objects from the LabelMe dataset ([144, 162]). The distributions in Fig. 2-2(b) and Fig. 2-2(c) look similar and can be approximated by a Zipf law with the form $A/rank$, where A is a constant and the number of instances of any category is inversely proportional to its rank in the sorted list.

There are also different ways to sample the visual world in order to create a collection of images for each category. For example, one might decide that different views of the same place qualify as different scenes, or one might choose to subdivide scenes based on spatial layout or surface features (e.g., forests with or without snow). Our goal here is to propose an initial list that is extensive enough as to cover most of plausible scene categories. The list is also limited to scene categories for which a name exists in English. Like estimating the number of visual object categories, counting the number of scene categories is an open problem and here we are providing a first estimate.

By modern standards, the SUN database is not especially large, containing on the order of one hundred thousand scenes. But the SUN database is, instead, **richly annotated** with scene categories, scene attributes [128], "memorability" measurements [86, 95], and object polygon-based annotation. To date, there are 326,582 manually segmented objects for the 5,650 object categories labeled. Object categories are visualized in Figure 2-7 and

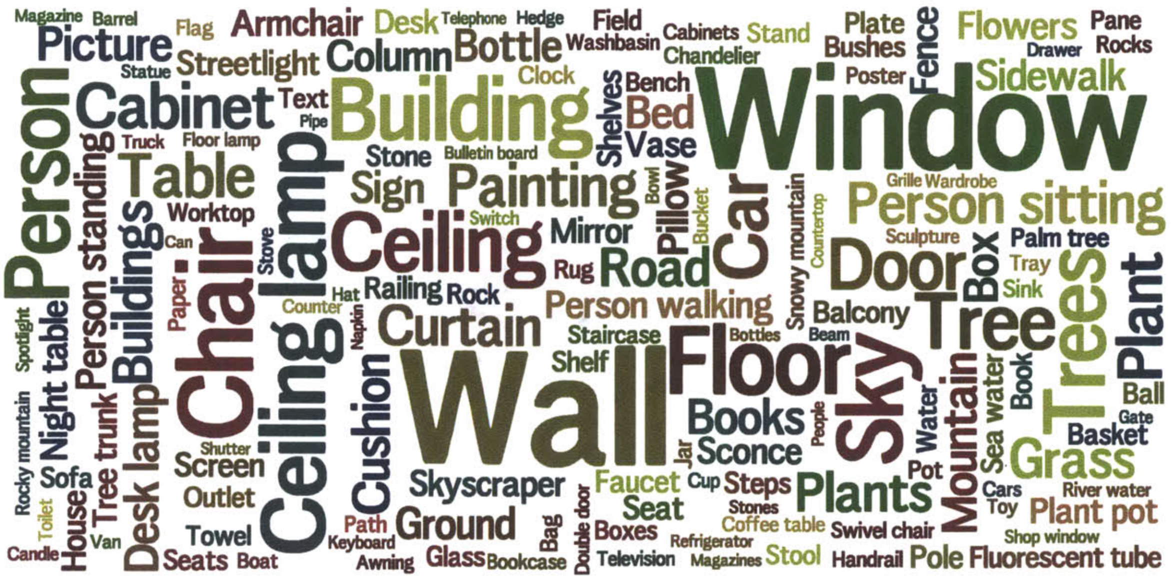


Figure 2-7: Object categories in the SUN database. The area of each word is proportional to the frequency of that object category.

annotated objects are shown in Figures 2-5, 2-8, 2-9 and 2-6. We believe the SUN database is the largest database from which one can learn the relationship among these object and scene properties. This combination of scene diversity and rich annotation is important for scaling scene understanding algorithms to work in the real world.

Figure 2-8: Sample object segments from popular object categories in the SUN database.

Figure 2-9: Samples of person in the SUN database.

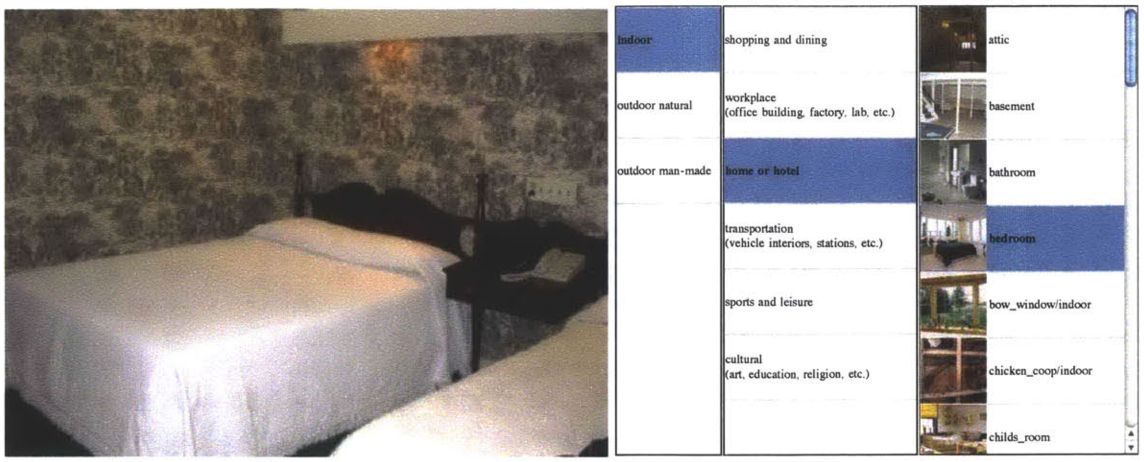


Figure 2-10: Graphical User Interface of Amazon's Mechanical Turk task for 397-category Alternative Forced Choice.

2.2 Behavioral studies using the SUN database

In this section, we study (1) human scene classification performance on the SUN database, (2) human estimates of the "typicality" of every image with respect to its scene category, and (3) human estimates of various high level "spatial envelope" properties of each scene category.

2.2.1 Scene categorization

We ask human participants to classify images from the database into one of 397 scene categories in an alternative forced choice setting. For this experiment, we have two goals: 1) to show that our database is constructed consistently and with minimal overlap between categories 2) to give an intuition about the difficulty of 397-way scene classification and to provide a point of comparison for computational experiments (Section 2.3.2).

Measuring human classification accuracy with 397 categories is challenging. We don't want to penalize humans for being unfamiliar with our specific scene taxonomy, nor do we want to train people on the particular category definitions and boundaries used in our database (however, such training was given to those who built the database). To help participants know which labels are available, we provide the interface shown in Fig. 2-10. Participants navigate through an over complete three-level hierarchy to arrive at a specific

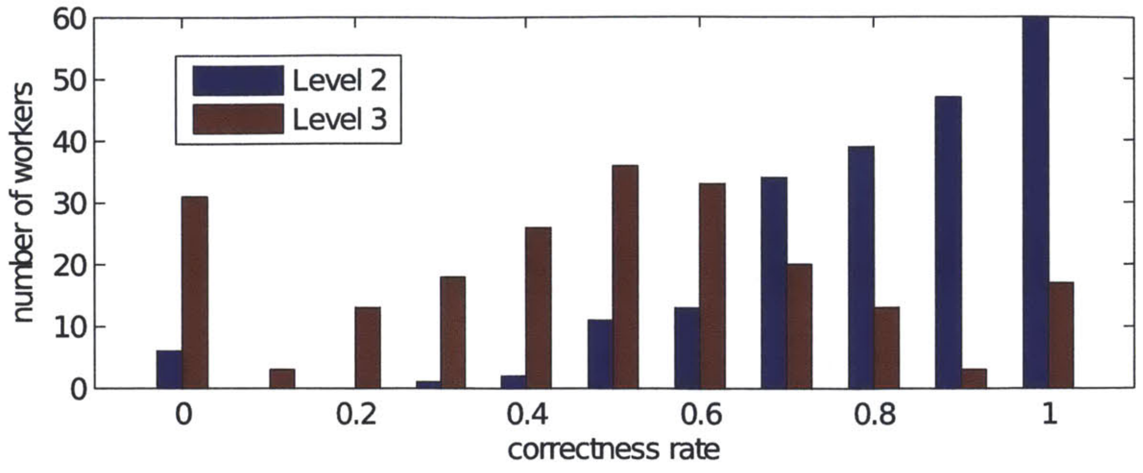


Figure 2-11: Histogram of the scene recognition performances of *all* AMT workers. Performance is measured at the intermediate (level 2) and leaf (level 3) levels of our hierarchy.

scene type (e.g. "bedroom") by making relatively easy choices (e.g., "indoor" versus "outdoor natural" versus "outdoor man-made" at the first level). The 3-level tree contains then 397 leaf nodes (SUN categories) connected to 15 parent nodes at the second level that are in turn connected to 3 nodes at the first level (super-ordinate categories). The mid-level categories were selected to be easily interpreted by workers, have minimal overlap, and provide a fairly even split of the images in each super-ordinate category. When there was any confusion about the best super-ordinate category for a category (e.g., "hayfield" could be considered natural or man-made), the category was included in both super-ordinate categories. This hierarchy is used strictly as a human organizational tool, and plays no roll in the experimental evaluations. For each leaf-level SUN category the interface shows a prototypical image from that category.

We measure human scene classification accuracy using Amazon's Mechanical Turk (AMT). For each SUN category we measure human accuracy on 20 test scenes, for a total of $397 \times 20 = 7940$ trials. We restricted these HITs to participants in the U.S. to help avoid vocabulary confusion.

The accuracy of all AMT workers is shown in Fig. 2-11. On average, workers took 61 seconds per HIT and achieved 58.6% accuracy at the leaf level. This is quite high considering that chance is 0.25% and numerous categories are closely related (e.g., "church", "cathedral", "abbey", and "basilica"). However, a significant number of workers have 0%

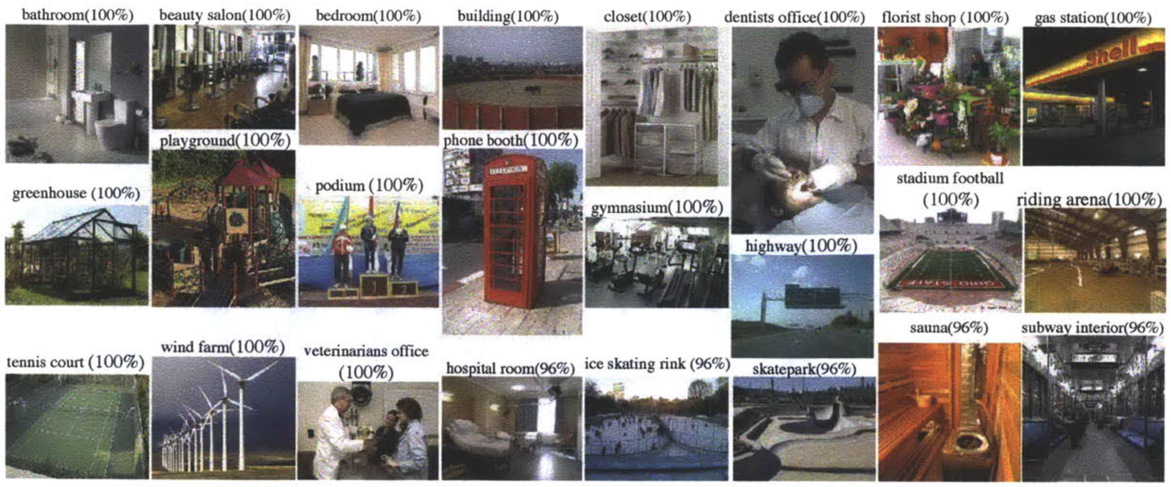


Figure 2-12: SUN categories with the highest human recognition rate.

accuracy – they do not appear to have performed the experiment rigorously. If we instead focus on the "good workers" who performed at least 100 HITs and have accuracy greater than 95% on the relatively easy first level of the hierarchy the leaf-level accuracy rises to 68.5%. These 13 "good workers" accounted for just over 50% of all HITs. For reference, an author involved in the construction of the database achieved 97.5% first-level accuracy and 70.6% leaf-level accuracy. In the remainder of the chapter, all evaluations and comparisons of human performance utilize only the data from the good AMT workers.

Fig. 2-12 and 2-13 show the SUN categories for which the good workers were most and least accurate, respectively. For the least accurate categories, Fig. 2-13 also shows the most frequently confused categories. The confused scenes are semantically similar – e.g. abbey and church, bayou and river, and sandbar and beach. Within the hierarchy, indoor sports and leisure scenes are the most accurately classified (78.8%) while outdoor cultural and historical scenes were least accurately classified (49.6%). Even though humans perform poorly on some categories, the confusions are typically restricted to just a few classes (Fig. 2-22).

Human and computer performance are compared extensively in Section 2.3.2. It is important to keep in mind that the human and computer tasks are not completely equivalent. The "training data" for AMT workers was a text label, a single prototypical image, and their past visual experience with each category (which could be extensive for every-

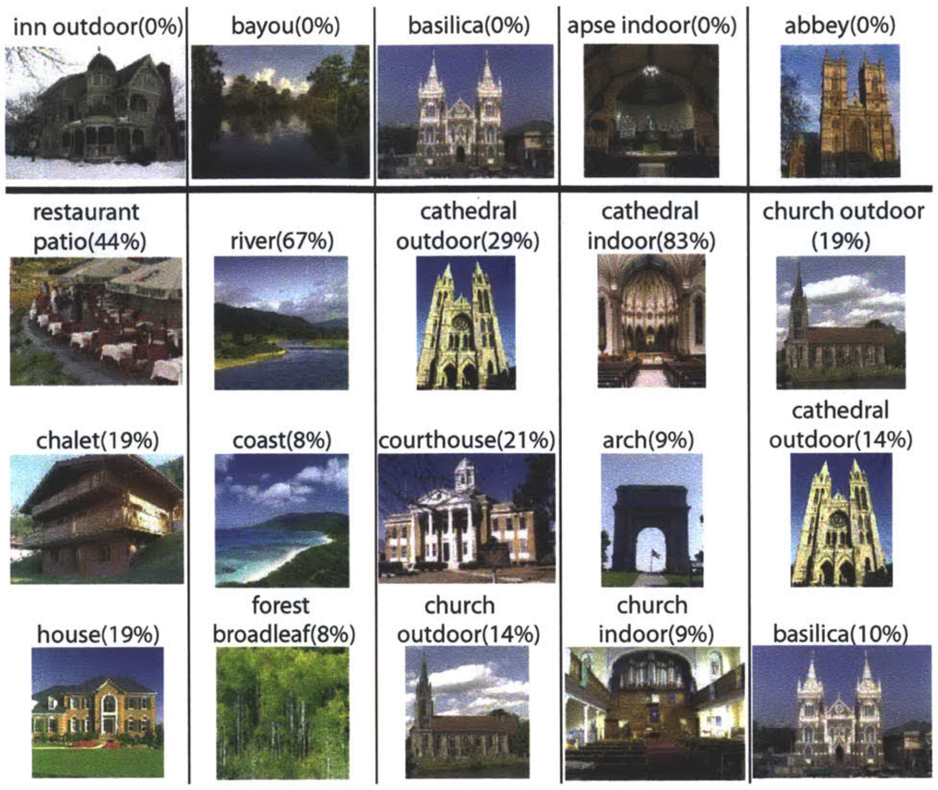
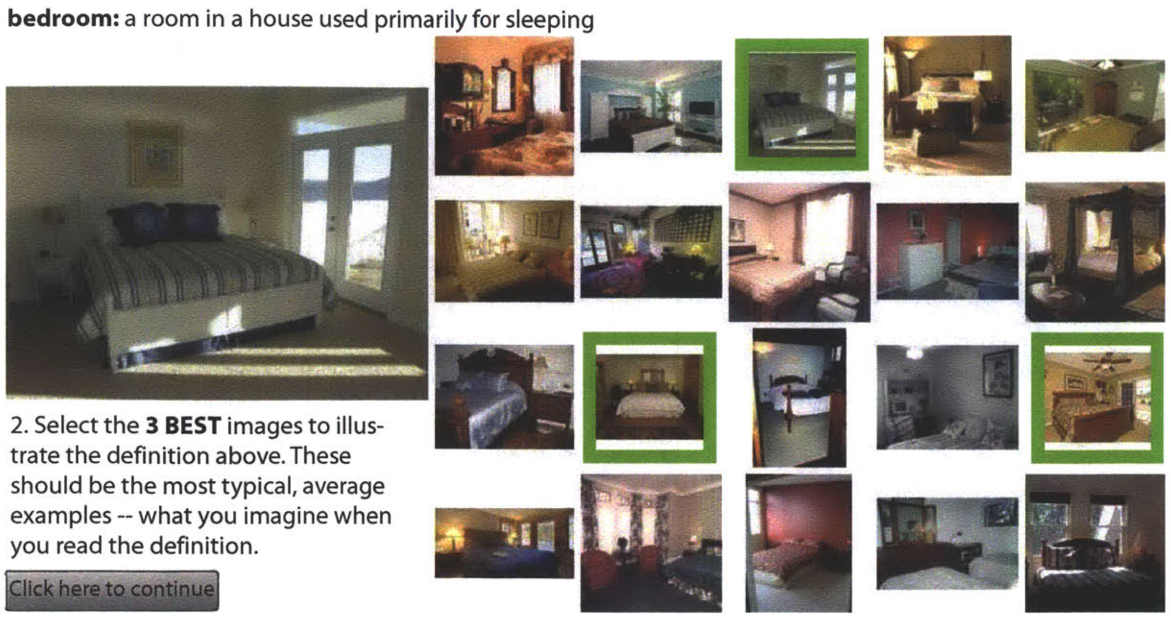


Figure 2-13: Top row: SUN categories with the lowest human recognition rate. Below each of these categories, in the remaining three rows, are the most confusing classes for that category.

day categories like "bedroom" but limited for others). The computational model had 50 training examples per category. It is also likely that human and computer failures are qualitatively different – human misclassifications are between semantically similar categories (e.g. "food court" to "fastfood restaurant"), while computational confusions are more likely to include semantically unrelated scenes due to spurious visual matches (e.g., "skatepark" to "van interior"). In Fig. 2-23 we analyze the degree to which human and computational confusions are similar. The implication is that the human confusions are the most reasonable possible confusions, having the shortest possible semantic distance. But human performance isn't necessarily an upper bound – in fact, for many categories the humans are less accurate than the best computational methods (Fig. 2-21).


bedroom: a room in a house used primarily for sleeping

2. Select the **3 BEST** images to illustrate the definition above. These should be the most typical, average examples -- what you imagine when you read the definition.

Figure 2-14: The display seen by participants in the typicality rating task.

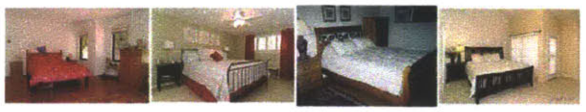
2.2.2 Typicality of scenes

In the computer vision literature, the organization of visual phenomena such as scenes into *categories* is ubiquitous. Each particular instance is assumed to be an equally good representative of the category. This is a useful high level model for many computational experiments, but most theories of categorization and concepts agree that category membership is graded - some items are more typical examples of their category than others [173]. The most typical examples of a category show many advantages in cognitive tasks: for example, typical examples are more readily named when people are asked to list examples of a category, and response times are faster for typical examples when people are asked to verify category membership [140].

To study the typicality of scenes, we ran a task on Amazon's Mechanical Turk to ask human annotators to choose most and least typical examples from a list of images [39]. Participants were told that the goal of the experiment was to select illustrations for a dictionary. Each trial consisted of three parts. First, participants were given the name of a scene category from the database, a short definition of the scene category, and four images. Workers were asked to select which of the four images matched the category name and definition (one of the four images was drawn from the target category and the other three were

(a) Most typical beaches

(b) Least typical beaches


(c) Most typical bedrooms

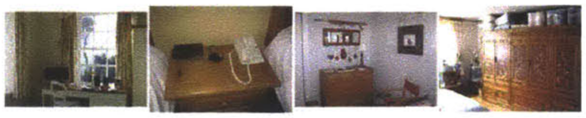

(d) Least typical bedrooms

Figure 2-15: Example images rated as the most and least typical by participants from Amazon's Mechanical Turk.

randomly selected from other categories). The purpose of this task was to ensure that participants read the category name and definition before proceeding to the rating task. Next, participants were shown 20 images in a 4×5 array. These images were drawn randomly from the target category, and did not include the image which had served as the target in the previous task. Images were shown at a size of 100×100 pixels, but holding the mouse over any image caused a larger 300×300 pixel version of that image to appear. An example of this display is shown in Fig. 2-14. Workers were asked to select, by clicking with the mouse, three images that best illustrated the scene category. In the third part of the task, workers were shown the same 20 images (but with their array positions shuffled) and were asked to select the three worst examples of the target scene category.

For this experiment we used the 706 scene categories from our SUN database that contained at least 22 exemplars[2]. On each trial, the set of 20 images was drawn randomly from the set of images in the target category. These random draws were such that each image appeared at least 12 times, and no more than 15 times over the course of the experiment. This resulted in 77,331 experimental trials. Each trial was completed by a single participant. 935 people participated in the experiment[3]. Participants could complete as many trials as they wished; the average number of trials completed per participant was 82.7 trials (median 7 trials).

[2]Category size ranged from 22 images in the smallest categories to 2360 in the largest. A total of 124,901 images were used in the experiment.

[3] All workers were located in the United States and had a good performance record with the service (at least 100 HITs completed with an acceptance rate of 95% or better). Workers were paid $0.03 per trial.

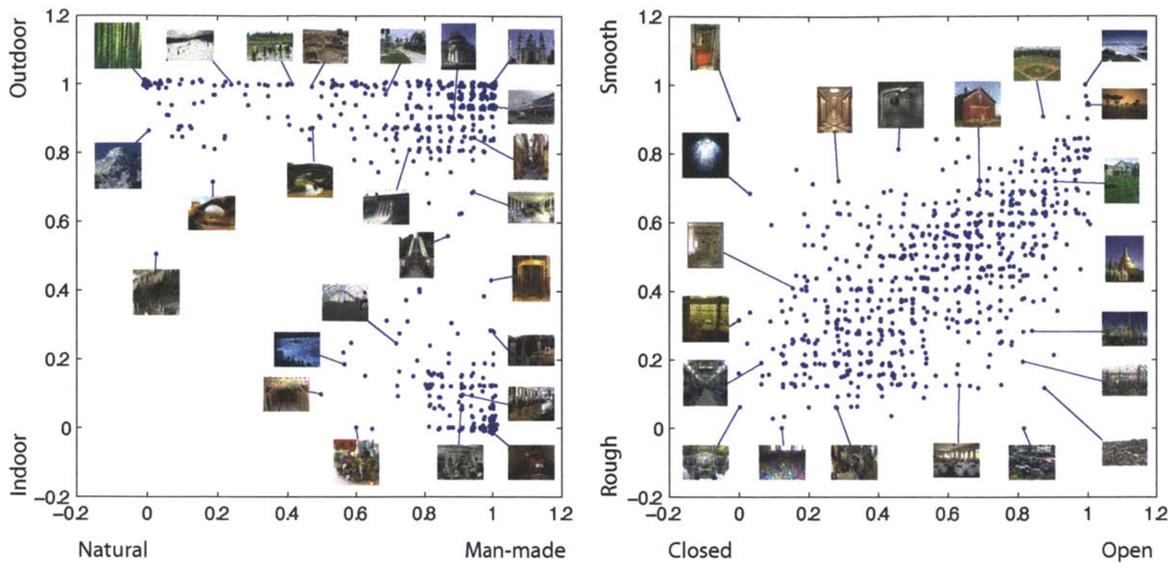
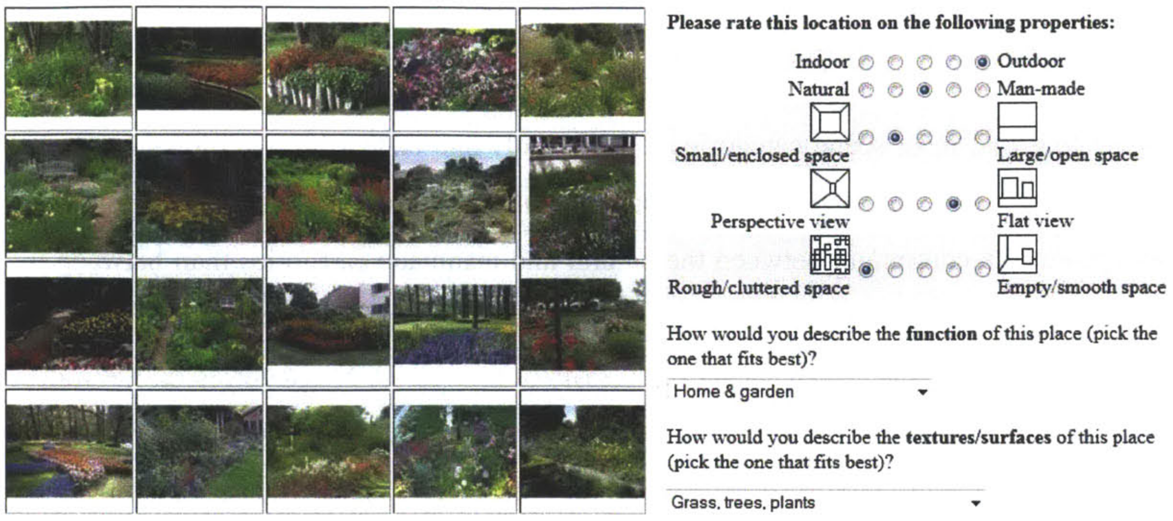


Figure 2-16: Organization of the scene categories along the spatial envelope properties.

Participants' performance was evaluated using two measures: 1) performance on the 4AFC task, and 2) whether they selected different images as the best and worst examples on a single trial. In general, participants performed well on the 4AFC task, with an average correct response rate of 97% (s.d. 0.13%). Participants also reliably selected different images as the best and worst examples of their category: participants marked an image as both best and worst on only 2% of trials (s.d. 0.10%); the likelihood of re-selecting an image by chance is 40%. We identified 19 participants (2% of total participants) who re-selected the same images as both best and worst on at least 25% of trials, which suggests that they were selecting images at random with no regard for the task. Together these participants had submitted 872 trials (1.13% of trials), which were dropped from further analysis.

A typicality score was obtained for each image in the dataset. The typicality score was calculated as the number of times the image had been selected as the best example of its category, minus a fraction (0.9) of the number of times it was selected as the worst example, divided by the number of times the image appeared throughout the experiment:

$$\text{typicality} = \frac{\text{\# of ``best'' votes} - 0.9 \times \text{\# of ``worst'' votes}}{\text{number of appearances}}.$$



Please rate this location on the following properties:

Indoor ○ ○ ○ ○ ● Outdoor

Natural ○ ○ ● ○ ○ Man-made

Small/enclosed space ○ ● ○ ○ ○ Large/open space

Perspective view ○ ○ ○ ● ○ Flat view

Rough/cluttered space ● ○ ○ ○ ○ Empty/smooth space

How would you describe the **function** of this place (pick the one that fits best)?

Home & garden

How would you describe the **textures/surfaces** of this place (pick the one that fits best)?

Grass, trees, plants

Figure 2-17: Graphical User Interface for the Amazon Mechanical Turk scene properties rating task.

Taking a fraction of the worst votes allows the number of best votes to be used as a tie-breaker for images that performed similarly. A typicality score near 1 means an image is extremely typical (it was selected as the best example of its category nearly every time it appeared in the experiment), and a typicality score near -1 means an image is extremely atypical (it was nearly always selected as a worst example). Examples of the most and least typical images from various categories are shown in Fig. 2-15.

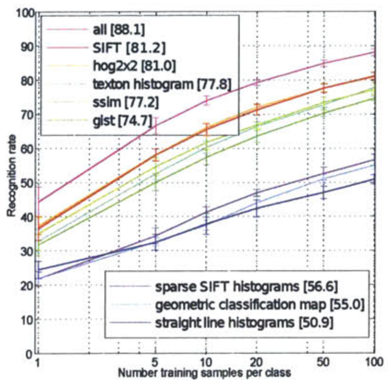
2.2.3 High-level properties of scene categories

To further characterize the database, we asked human observers to rank each category on the primary "scene spatial envelope" axes of [126]: indoor versus outdoor, natural versus manmade, openness, roughness, and expansion. We also asked them to classify the categories according to surface texture and social functions. Eight workers rated each scene category in a task on Amazon Mechanical Turk. Workers were shown a set of twenty highly-typical exemplars from the category and were asked to apply a rating to the image set; they were not given the category name. Workers rated the spatial envelope attributes on a 5-value scale, classified surface texture into one of 5 categories (water/ice, dirt/stone, vegetation, manmade outdoor, or indoor) and classified the social function of the place into one of 9 categories (home, sports/outdoors, commerce, art, education, religion, historical/cultural, industry, and transportation).

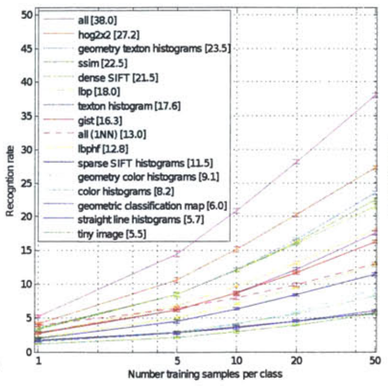
Responses on the indoor/outdoor and natural/man-made scales tended to be bimodal, suggesting that most scene categories fell into only one of the three top-level categories from the scene hierarchy (indoor, outdoor natural, or outdoor manmade), although there was more of a continuum between the natural and manmade categories than between the indoor and outdoor categories. Ratings on the spatial envelope properties were more normally distributed. Fig. 2-16 shows the distribution of all SUN categories on the indoor verse natural and roughness verse openness axes. In general, the correlation between human ratings and our classification of scenes into the 3-level hierarchy was quite high: the correlation between our indoor/outdoor sorting and human ratings was 0.97 and the correlation between our man-made/natural sorting and human ratings was 0.87. At the second level of the hierarchy, the correlation between our classification and the human ratings ranged from 0.52 to 0.86.

Most of the spatial envelope properties were correlated with each other. As might be expected, the indoor-outdoor property was negatively correlated (-0.45) with naturalness: indoor scenes are more likely to be man-made and outdoor scenes are more likely to be natural. Openness was correlated with high outdoor ratings (0.72) and high naturalness (0.48). Roughness was correlated with high indoor ratings (0.59) and high naturalness (0.41), and was negatively correlated with openness (0.63). Expansion was the only spatial envelope property tested that did not correlate well with any other spatial envelope property.

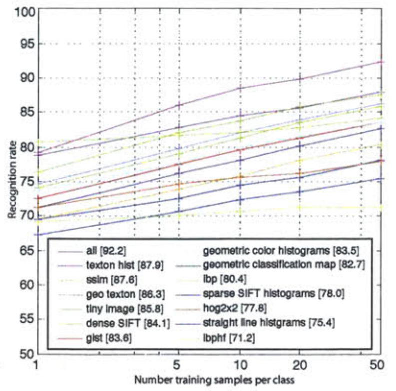
The social functions of scenes were not strongly correlated with their spatial envelope properties, with the exception of the "sports & outdoors" function, which was correlated with naturalness (0.60). Surface textures were primarily correlated with the indoor-outdoor and naturalness properties: the indoor texture was strongly correlated with the indoor-outdoor property (0.96) and negatively correlated with naturalness (-0.45), while the water, grass, and dirt textures were positively correlated with naturalness (0.52, 0.47, and 0.45, respectively). While these experiments examine high level properties of scene *categories*, there is naturally intra-category variation as well. The SUN attribute database [128] collects per-image scene attributes to examine such variations.



(a) 15 scene categories.



(b) 397 scene categories.



(c) Indoor-vs-outdoor.

Figure 2-18: (a) Classification accuracy on the 15 scene dataset[126, 104, 48]. (b) Classification accuracy on the 397 well-sampled categories from SUN database. (c) Classification accuracy for indoor-vs-outdoor task using SUN database.

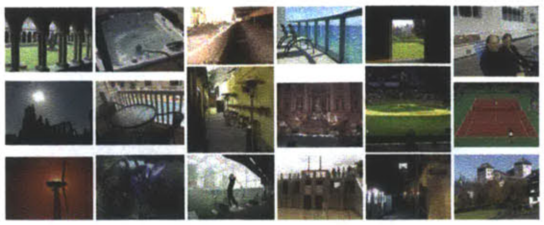
2.3 Computational semantic recognition

In this section we explore how discriminable the SUN categories and exemplars are with a variety of image features and kernels paired with 1 vs. all support vector machines.

2.3.1 Scene features

We selected or designed several state-of-the-art features that are potentially useful for scene classification: GIST, SIFT, and HOG (which are all local gradient-based approaches), SSIM (which relates images using their internal layout of local self-similarities), and Berkeley textons. As a baseline, we also include Tiny Images [168], color histograms and straight line histograms. To make our color and texton histograms more invariant to scene layout, we also build histograms for specific geometric classes as determined by [76]. The geometric classification of a scene is then itself used as a feature, hopefully being invariant to appearance but responsive to layout.

GIST: The GIST descriptor [126] computes a wavelet image decomposition. Each image location is represented by the output of filters tuned to different orientations and scales. We use a Gabor-like filters steerable pyramid with 8 orientations and 4 scales applied to the intensity (monochrome) image. To capture global image properties while keeping some spatial information, we take the mean value of the magnitude of the local features averaged


(a) Outdoor images mis-classified as indoor scenes.

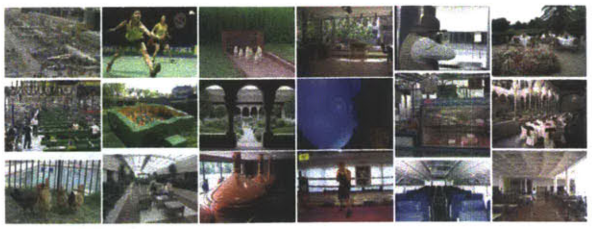

(b) Indoor images mis-classified as outdoor scenes.

Figure 2-19: Typical errors for the indoor-vs-outdoor classification.

over large spatial regions. The square output of each filter is averaged on a 4×4 grid. This results in an image descriptor of $8 \times 4 \times 16 = 512$ dimensions. GIST features [126] are computed using the code available online and we use a Gaussian RBF kernel.

HOG2×2: The histogram of oriented edges (HOG) descriptors is widely use for pedestrian and object detection. HOG decomposes an image into small squared cells (typically 8×8 pixels), computes an histogram of oriented gradients in each cell, normalizes the result using a block-wise pattern (with 2×2 square HOG blocks for normalization), and return a descriptor for each cell. HOG exists in two major variants: the original Dalal-Triggs variant [27] and the UoCTTI variant [52]. Dalal-Triggs HOG [27] works with undirected gradients only and does not do any compression, for a total of 36 dimensions. UoCTTI HOG [52] computes instead both directed and undirected gradients as well as a four-dimensional texture-energy feature, but projects the result down to 31 dimensions. In many applications, UoCTTI HOG tends to perform better than Dalal-Triggs HOG. Therefore, we use UoCTTI HOG in our experiments, computed using the code available online provided by [52]. In [184], we argue that stacking the features from multiple HOG cells into one feature is very important, because the higher feature dimensionality provides more descriptive power, and significantly improve the performance in our experiments. In our experiment, we tried different sizes of windows for stacking the HOG features, and only report the best performing one. 2×2 neighboring HOG descriptors are stacked together to form a 124 dimensional descriptor. The stacked descriptors spatially overlap. The descriptors are quantized into 300 visual words by *k*-means. With this visual word representation, three-level spatial histograms are computed on grids of 1×1, 2×2 and 4×4. Histogram intersection [104] is used to define the similarity of two histograms at the same pyramid level for two im-

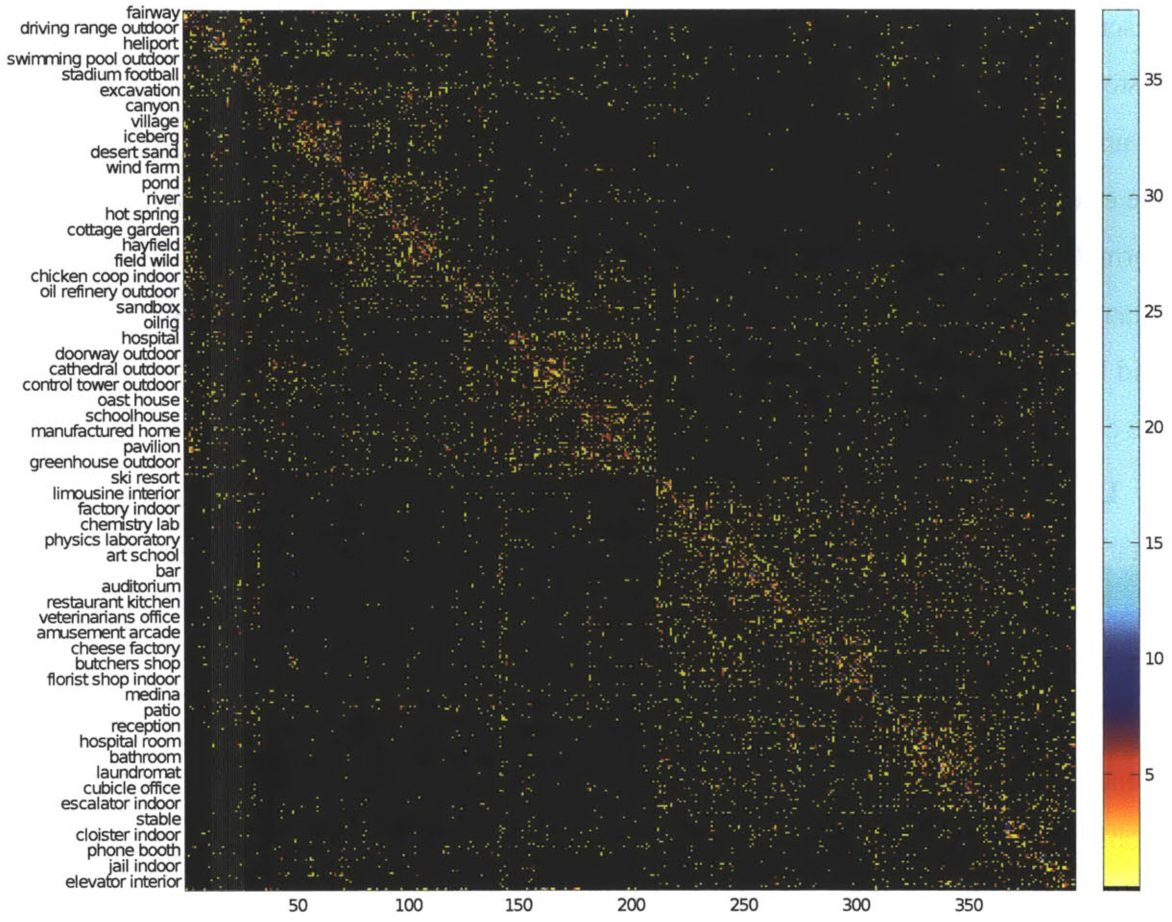


Figure 2-20: Pattern of confusion across categories. The classes have been ordered to reveal the blocky structure. For clarity, the elements in the diagonal have been set to zero in order to increase the contrast of the off-diagonal elements. On the Y axis we show a sample of the scene categories. Confusions seem to be coherent with semantic similarities across classes. The scenes seem to be organized as indoor (top), urban (center) and nature (bottom).

ages. The kernel matrices at the three levels are normalized by their respective means, and linearly combined together using equal weights.

Dense SIFT: As with HOG2×2, SIFT descriptors are densely extracted [104] using a flat rather than Gaussian window at two scales (4 and 8 pixel radii) on a regular grid at steps of 1 pixels. First, a set of orientation histograms are created on 4×4 pixel neighborhoods with 8 bins each. These histograms are computed from magnitude and orientation values of samples in a 16×16 neighboring region such that each histogram contains samples from a 4×4 subregion of the original neighborhood region. The magnitudes are further weighted by a Gaussian function with equal to one half the width of the descriptor window. The descriptor then becomes a vector of all the values of these histograms. Since there are 4×4 histograms each with 8 bins the vector has 128 elements. This vector is then normalized to unit length in order to enhance invariance to affine changes in illumination. To reduce the effects of non-linear illumination a threshold of 0.2 is applied and the vector is again normalized. The three descriptors are stacked together for each HSV color channels, and quantized into 300 visual words by k-means. Next, kernels are computed from spatial pyramid histograms at three levels by the same method above for HOG2$\times$2. SIFT descriptors are computed using the VLFeat library [177].

LBP: Local Binary Patterns (LBP) [124] is a multiresolution approach to gray-scale and rotation invariant texture classification based on local binary patterns and nonparametric discrimination of sample and prototype distributions. The method is based on recognizing that certain local binary patterns are fundamental properties of local image texture, and their occurrence histogram has proven to be a powerful texture feature. We can regard the scene recognition as a texture classification problem and therefore apply this model to our problem. Timo et al. also extended this approach to be a rotation invariant image descriptor, called Local Binary Pattern Histogram Fourier (LBP-HF)[7]. We try this descriptor to examine whether rotation invariance is suitable for scene recognition.

Texton Histograms: A traditional and powerful local image descriptor is to convolve the image with gabor-like filter bank [152]. Therefore, we use eight oriented even and odd symmetric Gaussian derivative filters and a center surround (difference of Gaussians) filter, as the popular image segmentation framework [8, 9]. We use a filterbank containing 8 even and odd-symmetric filters and one center-surround filter at 2 scales. The even-symmetric filter is a Gaussian second derivative, and the odd-symmetric filter is its Hilbert transform. We build a 512-entry universal texton dictionary [117] by clustering responses to the filter bank. For each image we then build a 512-dimensional histogram by assigning each pixel's set of filter responses to the nearest texton dictionary entry. We compute kernels from normalized χ^2 distances.

Sparse SIFT histograms: As in "Video Google" [157], we build SIFT features at Hessian-affine and MSER [118] interest points. We cluster each set of SIFTs, independently, into dictionaries of 1,000 visual words using k-means. An image is represented by

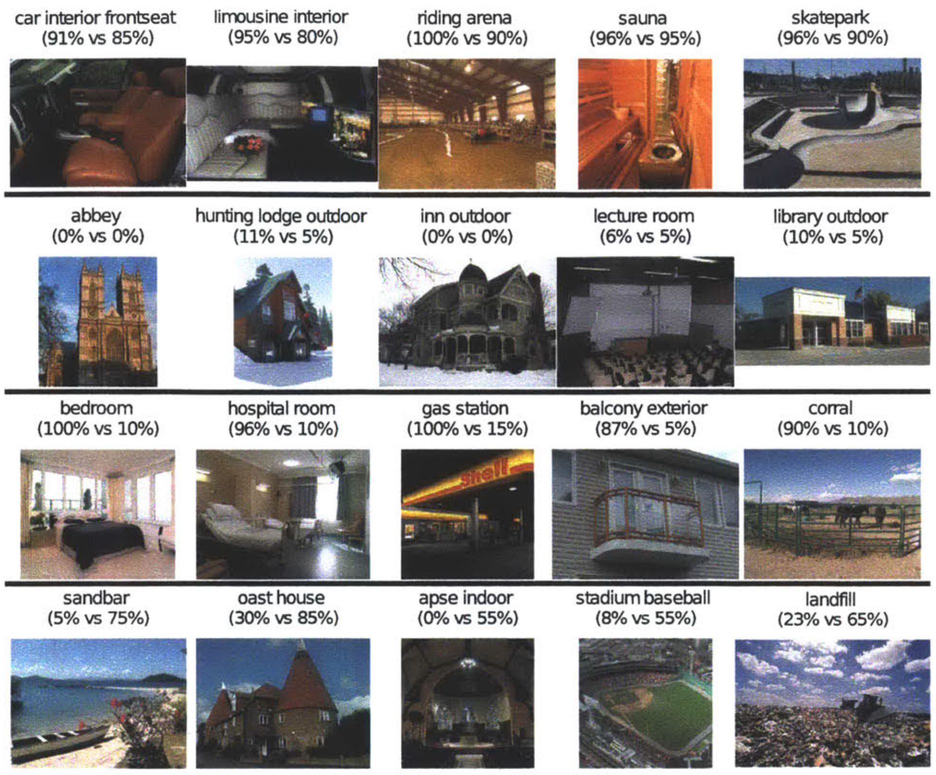


Figure 2-21: Categories with similar and disparate performance in human and "all features" SVM scene classification. Human accuracy is the left percentage and computer performance is the right percentage. From top to bottom, the rows are 1) categories for which both humans and computational methods perform well, 2) categories for which both perform poorly, 3) categories for which humans perform better, and 4) categories for which computational methods perform better. The "all features" SVM tended to outperform humans on categories for which there are semantically similar yet visually distinct confusing categories, e.g., sandbar and beach, baseball stadium and baseball field, landfill and garbage dump.

two histograms counting the number of sparse SIFTs that fall into each bin. An image is represented by two 1,000 dimension histograms where each SIFT is soft-assigned, as in [132], to its nearest cluster centers. Kernels are computed with χ^2 distance.

SSIM: Self-similarity descriptors [154] are computed on a regular grid at steps of five pixels. Each descriptor is obtained by computing the correlation map of a patch of 5×5 in a window with radius equal to 40 pixels, then quantizing it in 3 radial bins and 10 angular bins, obtaining 30 dimensional descriptor vectors. The descriptors are then quantized into 300 visual words by k-means. After that, kernels are computed from spatial histograms at three levels using χ^2 distance.

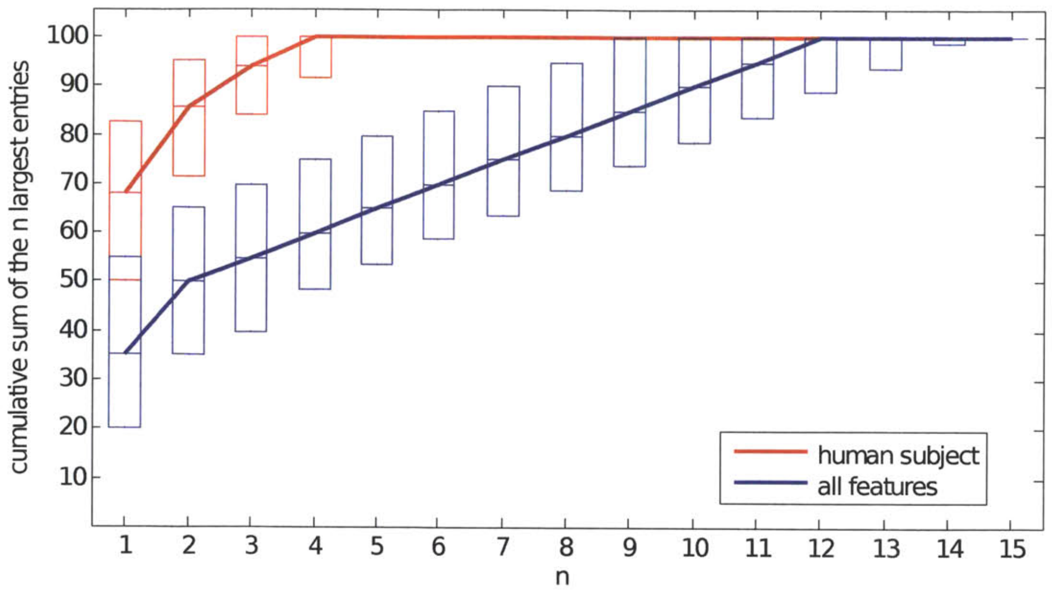


Figure 2-22: The cumulative sum of the n largest entries in each row of the confusion matrix. The box indicates the 25th to 75th percentiles at each n.

Tiny Images: The most trivial way to match scenes is to compare them directly in color image space. Reducing the image dimensions drastically makes this approach more computationally feasible and less sensitive to exact alignment. This method of image matching has been examined thoroughly by Torralba et al. [168] for the purpose of object recognition and scene classification. Inspired by this work we use 32 by 32 color images as one of our features. Images are compared with an RBF kernel.

Line Features: We find straight lines from Canny edges using the method described in Video Compass [99]. For each image we build two histograms based on the statistics of detected lines– one with bins corresponding to line angles and one with bins corresponding to line lengths. We use an RBF kernel to compare these unnormalized histograms. This feature was used in [66].

Color Histograms: We build joint histograms of color in CIE L*a*b* color space for each image. Our histograms have 4, 14, and 14 bins in L, a, and b respectively for a total of 784 dimensions. We compute distances between these histograms using χ^2 distance on the normalized histograms.

Geometric Probability Map: We compute the geometric class probabilities for image regions using the method of Hoiem et al. [76]. We use only the ground, vertical, porous,

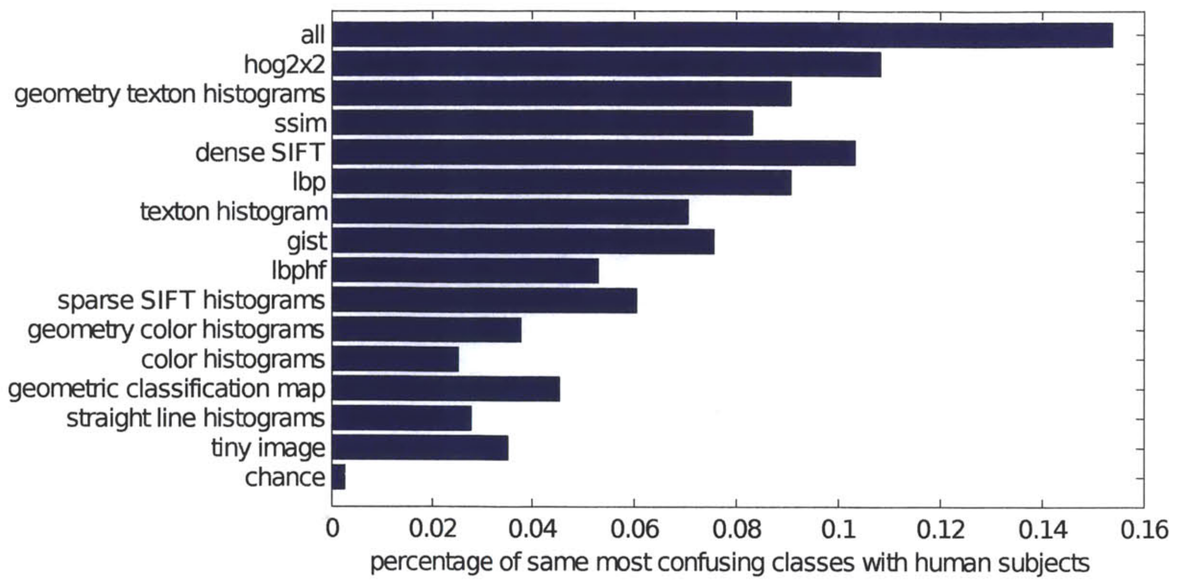


Figure 2-23: For each feature, we plot the proportion of categories for which the largest *incorrect* (off-diagonal) confusion is the same category as the largest human confusion.

and sky classes because they are more reliably classified. We reduce the probability maps for each class to 8x8 and use an RBF kernel. This feature was used in [66].

Geometry Specific Histograms: Inspired by "Illumination Context" [103], we build color and texton histograms for each geometric class (ground, vertical, porous, and sky). Specifically, for each color and texture sample, we weight its contribution to each histogram by the probability that it belongs to that geometric class. These eight histograms are compared with χ^2 distance after normalization.

2.3.2 Scene categorization

For comparison with previous papers, we show results on the 15-scene categories dataset [126, 104, 48] in Fig. 2-18(a). The performance on the 397-category SUN database is shown in Fig. 2-18(b). For each feature, we use the same set of training and testing splits. For trials with fewer training examples, the testing sets are kept unchanged while the training sets are decreased. The "all features" classifier is built from a weighted sum of the kernels of the individual features. The weight of each constituent kernel is proportional to the fourth power of its individual accuracy. As an additional baseline, we plot the performance of the "all features" kernel using one-nearest-neighbor classification. The 1-vs-all

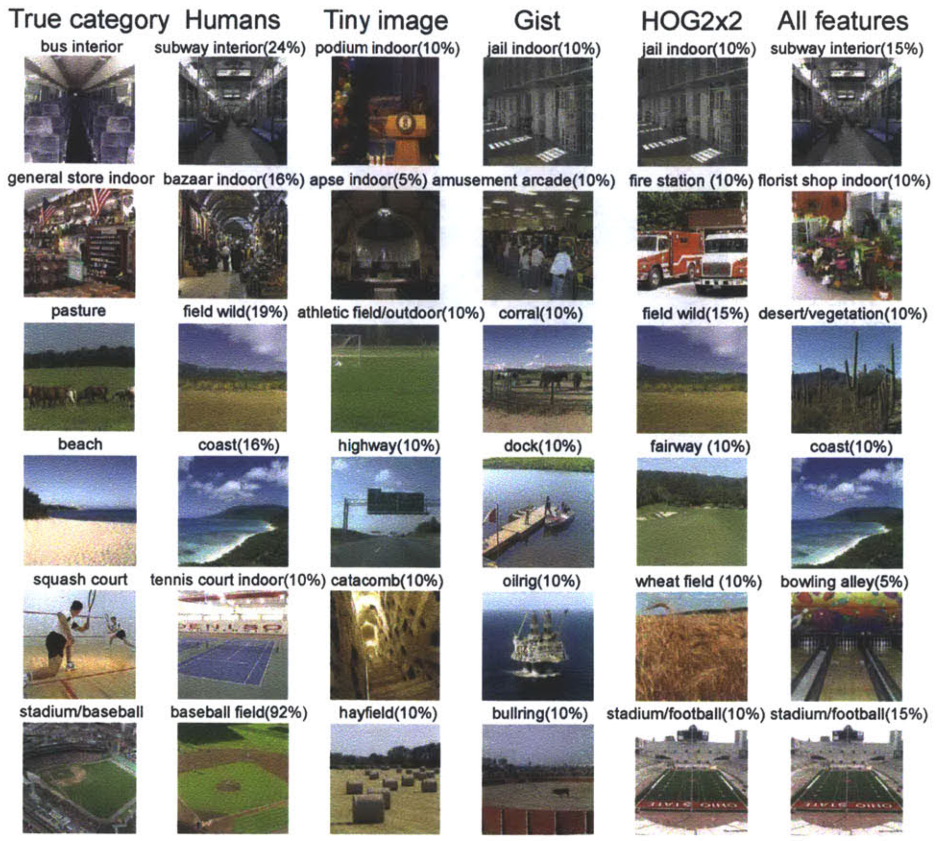
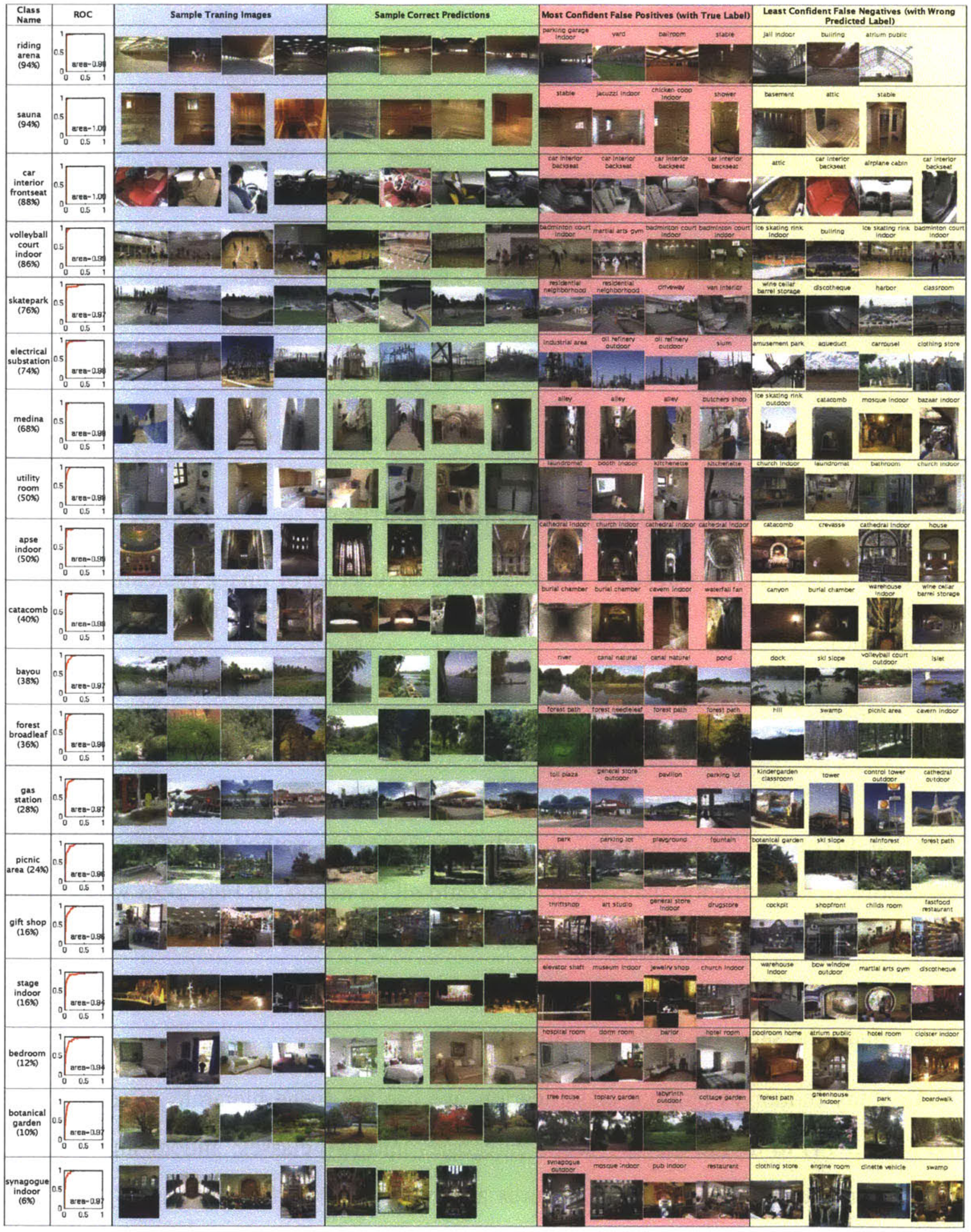


Figure 2-24: Most confused categories for the 397 categories classification task.

SVM has nearly three times higher accuracy. It is interesting to note that with increasing amounts of training data, the performance increase is more pronounced with the SUN dataset than the 15 scene dataset. The confusion matrix of the "all features" combined classifier is shown in Fig. 2-20. Classification results for selected categories are shown in Fig. 2-25. The best scene classification performance with all features, 38%, is still well below the human performance of 68%. In Fig. 2-21 we examine the categories for which human and machine accuracy is most similar and most dissimilar.

Also, humans have less confusion than any of the descriptors (the errors concentrate among fewer confusing categories). This is shown in Fig. 2-22. The plot shows the median value of the 397 cumulative sums of the n largest entries in each row of the confusion matrix. For n=1, the value corresponds to the median of the largest value of each row of the confusion matrix. For n=2, the value corresponds to the median of the sums of the two largest values of each row, and so on. Humans have far fewer confusing categories



more result at
`http://groups.csail.mit.edu/vision/SUN/Classification397/`

Figure 2-25: Selected SUN scene classification results using all features.

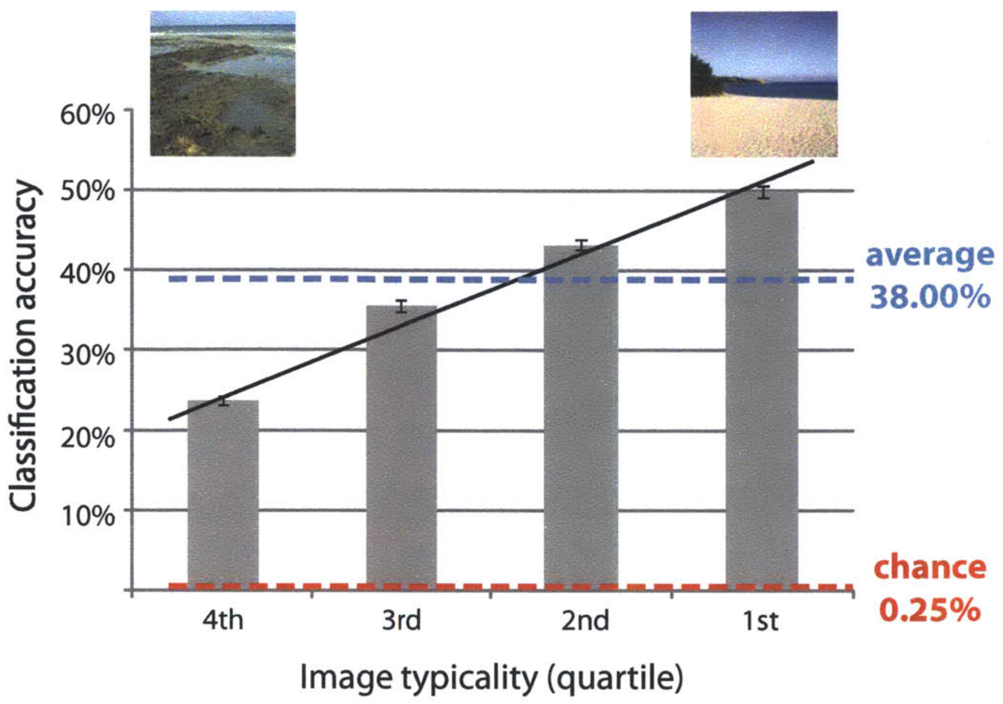


Figure 2-26: Performance of the SVM classifier as a function of image typicality. Images are sorted according to their typicality score from least typical (4th quartile) to most typical (1st quartile).

than the best performing descriptor. For humans, the 3 largest entries in each row of the confusion matrix sum to 95%, while the "all feature" SVM needs 11 entries to reach 95%. Note that this analysis does not relate directly to accuracy – a method might have relatively few entries in its confusion matrix, but they could all be wrong. In Fig. 2-23 we examine the similarity in scene classification *confusions* between humans and machines. The better performing features not only tend to agree with humans on correct classifications, they also tend to make the same mistakes that humans make.

To study indoor-vs-outdoor classification, we use the scene hierarchy (Section 4.1.2 and Fig. 3-14) to divide the 397 scene categories into two classes: indoor and outdoor. Then, we evaluate the same set of features and report their performance in Fig. 2-18(c). There are two categories, promenade deck and ticket booth, that are considered to be both indoor and outdoor in our scene hierarchy. Therefore, we exclude these two categories in our evaluation. We can see that the order of performance for different features are quite different, probably due to the great difference of the task compared to 397-way scene clas-

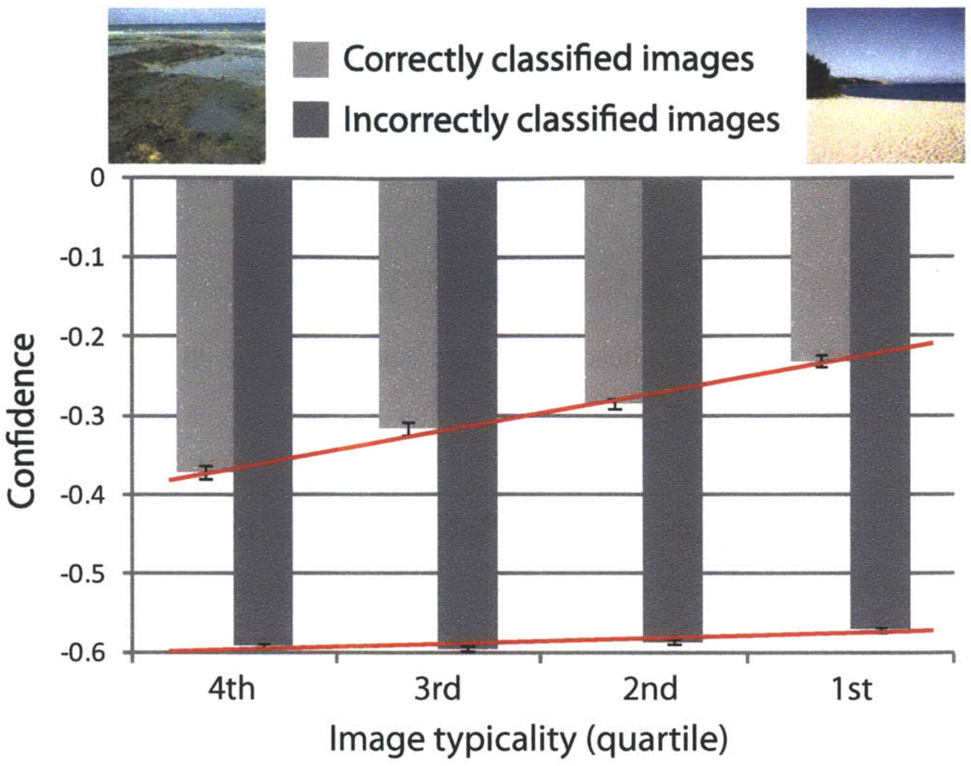


Figure 2-27: Confidence of the SVM classifier as a function of image typicality. Images are sorted according to their typicality score from least typical (4th quartile) to most typical (1st quartile).

sification. The overall performance is 92.2%, which suggests that this task is nearly solved. Fig. 2-19 shows some errors made by the classifier. We can see that the definition of indoor-vs-outdoor is ambiguous in some places, such as images which show both indoor elements and outdoor scenes through a window or a door. Therefore, the accuracy on this task might be actually higher than what the evaluation suggests.

2.3.3 Recognition and typicality of scenes

What is the scene classification performance as a function of scene typicality? Fig. 2-26 shows that classification performance for individual images varies with their typicality scores, using the combined kernel with all features: the most typical images were classified correctly about 50% of the time, and the least typical images were classified correctly only 23% of the time. Images were divided into four groups corresponding to the four quartiles of the distribution of typicality scores across the database. These groups contained 5020,

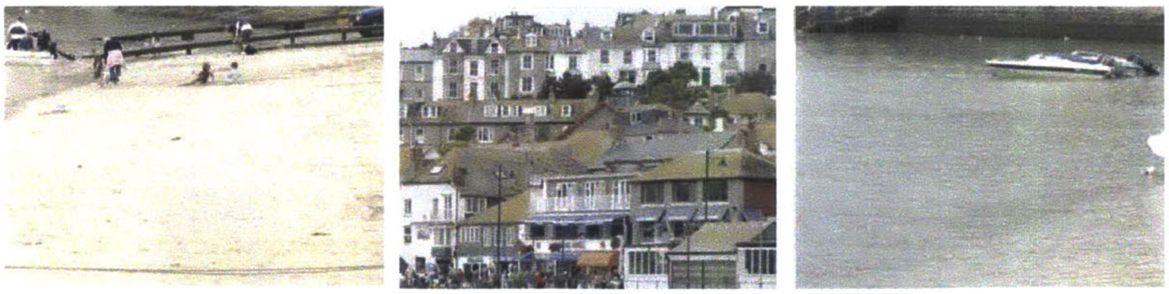


Figure 2-28: These scenes of a beach, a village, and a river are all from a single image (Fig. 5-19).

4287, 5655, and 4908 images (groups are listed in order from fourth quartile – lowest typicality – to first quartile). A one-way ANOVA comparing these quartile groups shows a significant effect of image typicality quartile on classification accuracy [4]; Bonferroni-corrected post-hoc tests show that the differences between each quartile are significant.

Image typicality is also related to the confidence of the SVM classifier. The confidence reflects how well the classifier believes the image matches its assigned category[5]. Fig. 2-27 shows the SVM confidence as a function of image typicality for correctly- and incorrectly-classified images. Confidence increases with increasing typicality, but this pattern is stronger in correctly-classified images. [6]

In summary, scenes which people rate as more typical examples of their category are more likely to be correctly classified by the algorithms based on global image descriptors. Although we cannot claim that the features used in these algorithms are the same features which humans use to perform the same classification task, this nevertheless indicates that more typical examples of a scene category contain more of the diagnostic visual features that are relevant for scene categorization. It also shows that typical images of scene categories can be reliably identified by state of the art computer vision algorithms.

[4] $F(3, 19846) = 278, p < .001$.

[5] Due to the difficulty of the one-versus-all classification task, confidence was low across all classifications, and even correctly-classified images had average confidence scores below zero.

[6] A 4×2 ANOVA gives significant main effects of image typicality ($F(3, 19842) = 79.8, p < .001$) and correct vs. incorrect classification ($F(1, 19842) = 6006, p < .001$) and a significant interaction between these factors ($F(3, 19842) = 43.5, p < .001$).

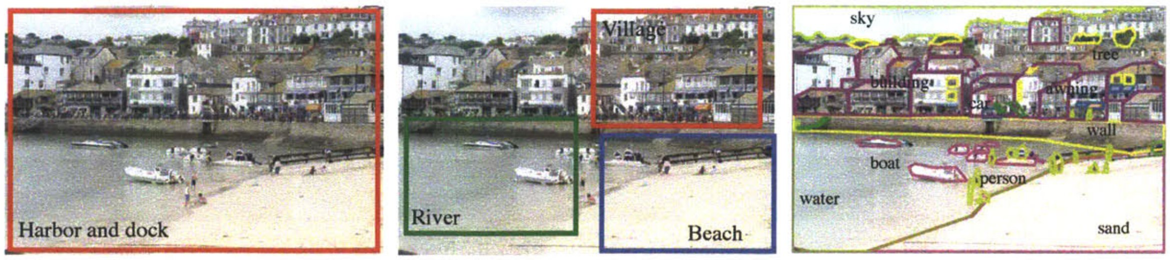


Figure 2-29: There are many complementary levels of image understanding. One can understand images on a continuum from the *global* scene level (left) to the *local* object level (right). Here, we introduce the intermediate concept of *local subscenes* (middle), and define a task called Scene Detection.

2.3.4 Scene detection

Imagine that you are walking in a street: scene recognition will tell you that you are in the street, and object recognition will allow you to localize people, cars, tables, etc. But there are additional detection tasks that lie in between objects and scenes. For instance, we want to detect restaurant terraces, or markets, or parking lots. These concepts also define localized regions, but they lack the structure of objects (a collection of parts in a stable geometric arrangement) and they are more organized than textures.

Here, we refer to these scenes within scenes as "subscenes" to distinguish that them from global scene labels. A single image might contain multiple scenes, where a scene is a bounded region of the environment that has a distinct functionality with respect to the rest. For instance, a street scene can be composed of store fronts, a restaurant terrace, and a park. To be clear, we are not describing a part-based model of scenes – subscenes are full-fledged, potentially independent scenes that create their own context. The objects and the actions that happen within subscenes have to be interpreted in the framework created by each local scene, and they might be only weakly related to the global scene that encompasses them. However, the dominant view in the literature is that one image depicts one scene category (with some exceptions [178, 19]). Also, while our approach takes scene representations and makes them more local, complementary work from [145] comes from the other direction and detects object arrangements called "visual phrases". Furthermore, scene detection is also related to scene viewpoint recognition [187].

As scenes are more flexible than objects, it is unclear what the right representation

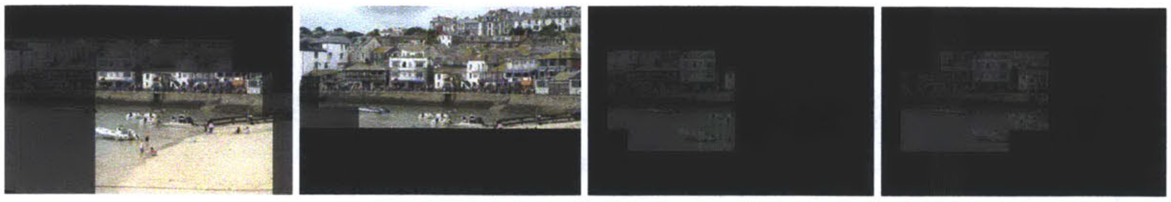

(a) beach (b) village (c) harbor&dock (d) river

Figure 2-30: Masks generated from the labels of Mechanical Turk workers. The brightness of each region is proportional to the degree of consistency in annotations.

will be in order to detect them in complex images. Here, we use the scene classification framework to directly classify image crops into subscene categories. We refer to this task as "scene detection". Our terminology is consistent with the object detection literature [44] where object *classification* or *recognition* involves classifying entire images, while object *detection* requires localizing objects within an image.

We choose a set of 47 scene categories that commonly co-occur within images (e.g. ocean with coast, alley with crosswalk, shelving with office area, river with harbor, village with plaza, etc.), and use our SUN database as training examples. It may seem odd to train a localized detector from entire images, but we do not expect global scenes and subscenes to vary significantly in appearance, only in image scale. To test scene detection, we create a database of 1000 images that have spatially localized subscenes (called the Scene Detection database). We use pairs of category related keywords to search for relevant images from online sources such as Flickr, Picasa, Google, and Bing. We manually filter the results to ensure that images 1) are large enough such that crops are still of sufficient resolution 2) depict relevant scenes and 3) are not distorted with respect to lighting, viewpoint, or post-processing. Unlike objects, scenes have unspecified spatial extent. The key idea to reliably annotate ground truth subscenes is to examine only the local image region without distraction from the surrounding context (for instance "sibling" or "parent" scenes of different categories). We do this by cropping out image regions at 3 different scales and assigning scene labels to those crops in isolation. We use a somewhat sparse set of 30 partially overlapped crops for each image in the Subscene Database. For each crop, we annotate its categories using Amazon's Mechanical Turk with three different workers. The annotators are presented with visual examples and text guidelines for all 47 categories, as

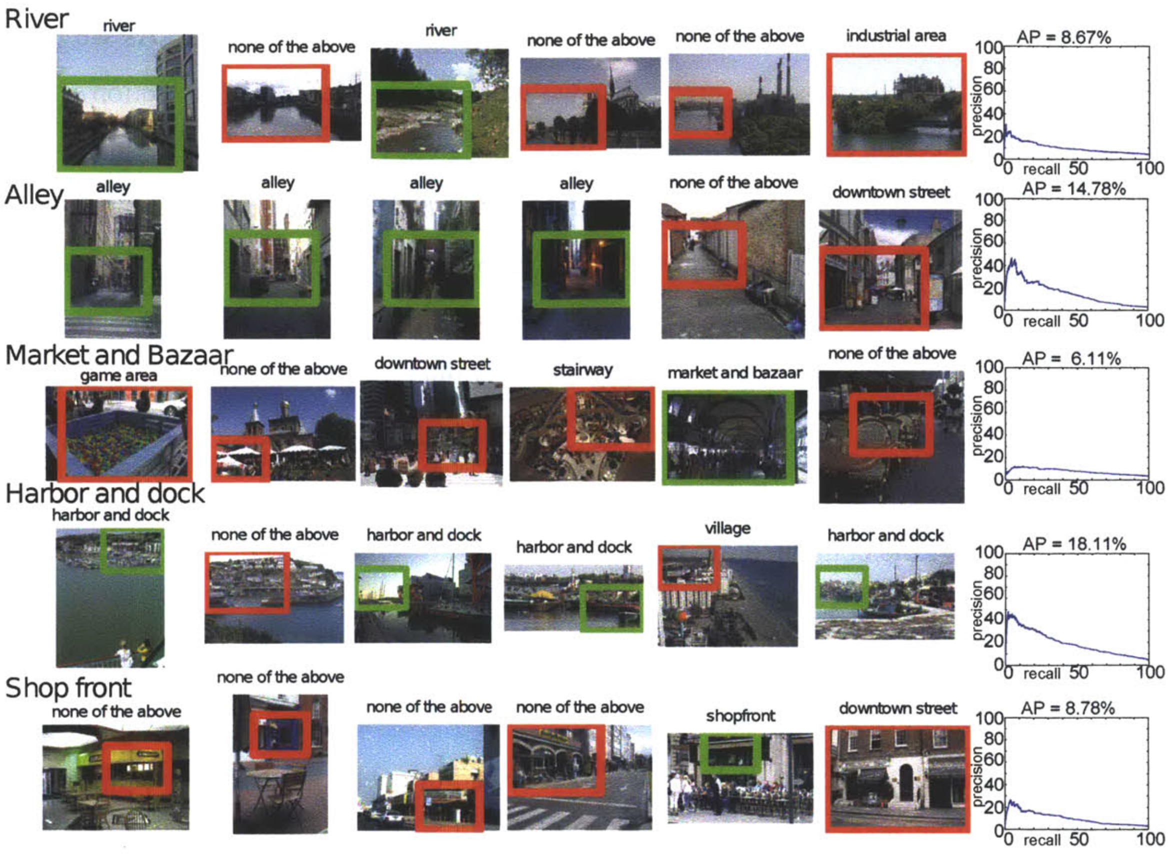


Figure 2-31: Scene detection results for some classes and their recall precision curves. Green boxes indicate correct detection, and red boxes indicate wrong detection.

well as a "none of the above" option for crops that do not fit any category or are not scenes. Fig. 2-30 visualizes the annotations.

We use the same algorithms as scene classification to train a detector. All testing is done on the scene detection dataset. To evaluate the detection performance, we use a simple criterion to plot the precision-recall curves: a detection is considered correct if the predicted label matches the human label. To generate testing images, we use sparse sliding windows because there usually aren't exact boundaries for subscenes. We uniformly generate about 30 crops at 3 different scales – the same windows that we have ground truth annotations for. We do not perform non-maximum suppression as in the object domain. Therefore, for each crop, there are 47 class prediction scores. We take the negative of the minimum score among all 47 class prediction scores as the score for the "non-scene" class, i.e. the 48th class.

In order to evaluate the detection performance, we use a simple criteria. As we have human labels for all the possible crops that we will consider during the detection stage, we consider a detection correct if the predicted label matches the human label. This task is harder than the recognition task as many crops are not considered to contain clearly defined scenes. In the test set there are a total of 26,626 crops, and 12,145 of those correspond to non-scenes. To report the performance, we use precision-recall curves: for each class and for each decision threshold we compute how many of the crops labeled according to each category are retrieved and with what precision. Fig. 2-31 shows examples of few precision-recall curves for a few selected classes (performances are typical). The average precision over all classes for the combined model is 19.224.

2.4 Conclusion

To advance the field of scene understanding, we need datasets that encompass the richness and variety of environmental scenes and knowledge about how scene categories are organized and distinguished from each other. In this work, we propose a quasi-exhaustive dataset of 908 scene categories. We evaluate state-of-the-art algorithms, and study several questions related to scene understanding. All images, object labels, scene definitions, and other data, as well as the source code, are available at `http://sundatabase.mit.edu/`.

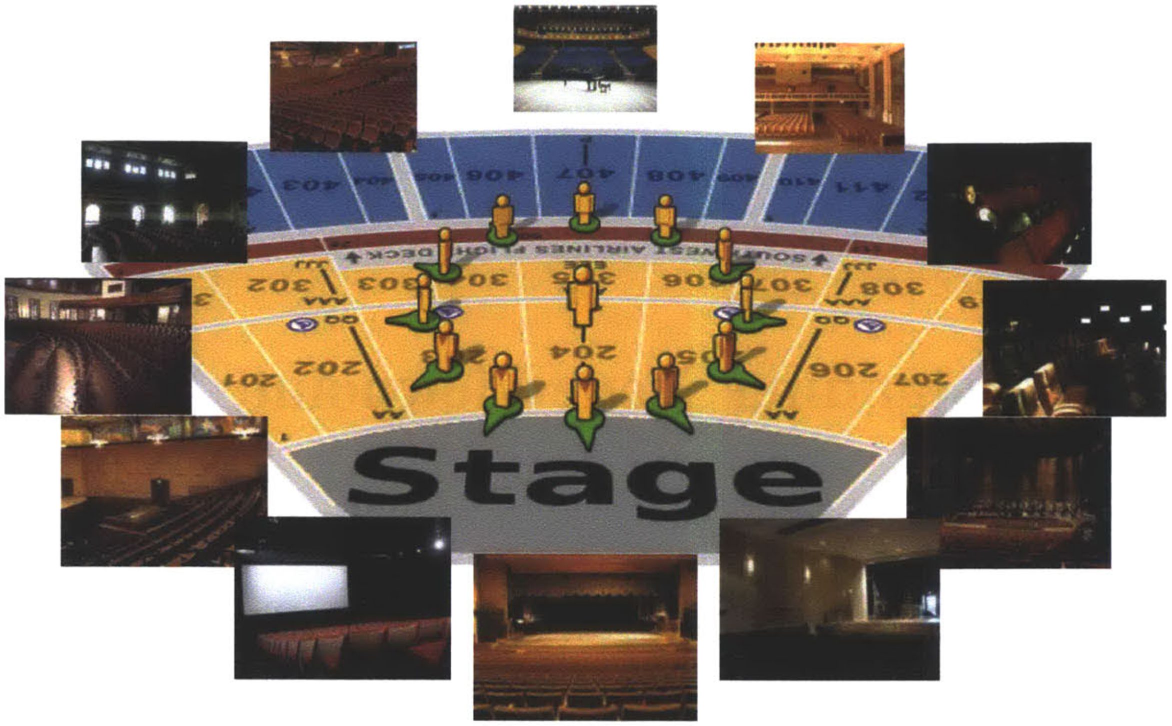
Chapter 3

Understanding Scene Viewpoint

The pose of an object carries crucial semantic meaning for object manipulation and usage (e.g., grabbing a mug, watching a television). Just as pose estimation is part of object recognition, viewpoint recognition is a necessary and fundamental component of scene recognition. For instance, as shown in Figure 5-19, a theater has a clear distinct distributions of objects - a stage on one side and seats on the other - that defines unique views in different orientations. Just as observers will choose a view of a television that allows them to see the screen, observers in a theater will sit facing the stage when watching a show.

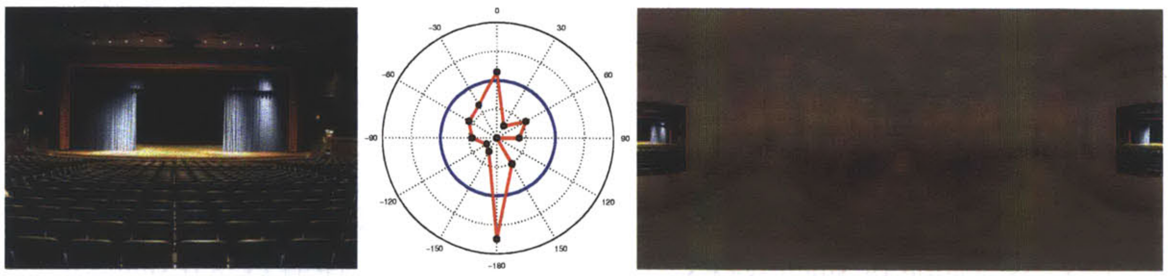
Although the viewpoint recognition problem has been well studied in objects [127], most research in scene recognition has focused exclusively on the classification of views. There are many works that emphasize different aspects of scene understanding [61, 68, 78, 75, 120, 122], but none of them make a clear distinction between views and places. In fact, current scene recognition benchmarks [104, 184] define categories that are only relevant to the classification of single views. However, recent evidence [125, 85] suggests that humans have a world-centered representation of the surrounding space, which is used to integrate views into a larger spatial context and make predictions about what exists beyond the available field of view.

The materials presented in this chaper are based on [187, 200] in collaboration with Antonio Torralba, Aude Oliva, Krista A. Ehinger, James Hays, Ping Tan, and Yinda Zhang.



(a) Despite belonging to the same place category (e.g., theater), the photos taken by an observer inside a place look very different from different viewpoints. This is because typical photos have a limited visual field of view and only capture a single scene viewpoint (e.g., the stage) at a time.

(b) We use panoramas for training place categorization and viewpoint recognition models, because they densely cover all possible views within a place.



(c) Given a limited-field-of-view photo as testing input (left), our model recognizes the place category, and produces a compass-like prediction (center) of the observers viewpoint. Superimposing the photo on an averaged panorama of many theaters (right), we can automatically predict the possible layout that extends beyond the available field of view.

Figure 3-1: Scene viewpoint recognition within a place.

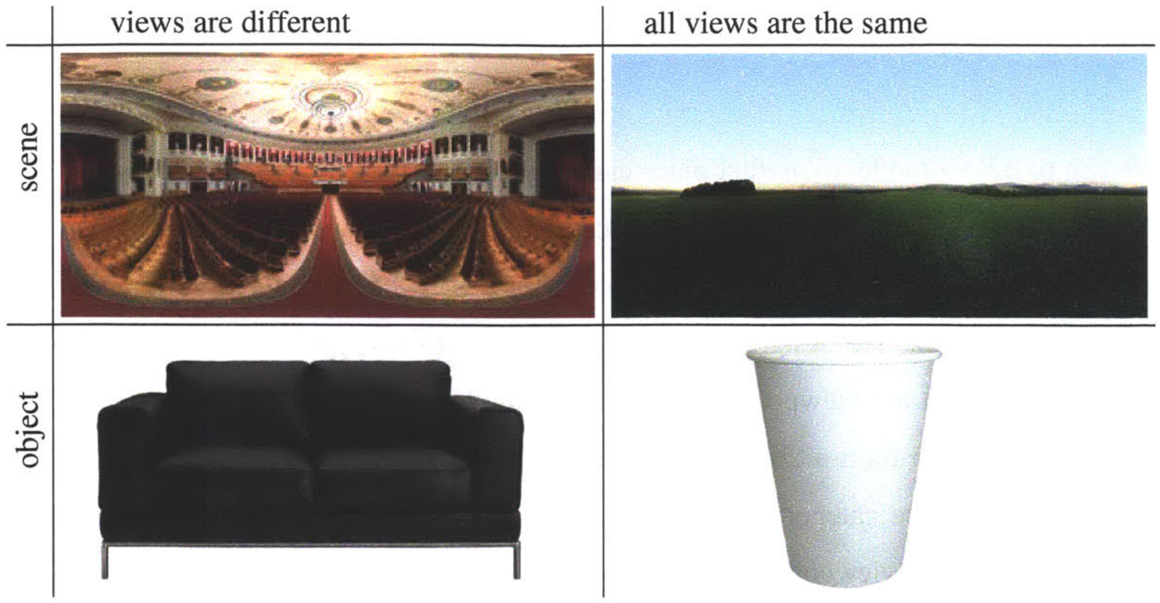


Figure 3-2: Illustration of viewpoint homogeneity scenes and objects.

The goal of this chapter is to study the viewpoint recognition problem in scenes. We aim to design a model which, given a photo, can classify the place category to which it belongs (e.g., a theater), and predict the direction in which the observer is facing within that place (e.g., towards the stage). Our model learns the typical arrangement of visual features in a 360°panoramic representation of a place, and learns to map individual views of a place to that representation. Now, given an input photo, we will be able to place that picture within a larger panoramic picture, as if we were to rotate the camera all around the observer.

We also study the symmetry property of places. Like objects, certain places exhibit rotational symmetry. For example, an observer standing in an open field can turn and see a nearly-identical view in every direction (isotropic symmetry). Similarly, a cup presents similar views when it is rotated. Other objects (e.g. sofa) and places (e.g. theaters) do not have rotational symmetry, but present some views that are mirror images of each other (left and right sides of the sofa, left and right views of the theater). Still other objects and places have no similar views (asymmetry). We design an algorithm to automatically discover the symmetry type of place categories and incorporate this information into its panoramic representation.

3.1 Overview

We aim to train a model to predict place category and viewpoint for a photo. We use 360-degree full-view panoramas for training, because they unbiasedly cover all views in a place. The training data are 26 categories of places[1], each of which contains many panoramas without annotation. We design a two-stage algorithm to first train and predict the place category, and then the viewpoint. We first generate a set of limited-field-of-view photos from panoramas to train a 26-way classifier to predict the place category (Section 3.2). Then, for each place category, we automatically align the panoramas and learn a model to predict the scene viewpoint (Section 3.3). We model the viewpoint prediction as a m-way classification problem to assign a given photo into one of the m viewpoints, uniformly sampled from different orientations on the $0°$-pitch line. Simultaneously with the alignment and classification, we automatically discover the symmetry type of each place category (Section 3.4). We test our model on two sets of photos: a set of views generated from our panoramas, and scene photos from the SUN database [184]. For the later dataset, we obtain viewpoint annotation from human raters, and illustrate the viewpoint biases that people display when taking photos of scenes (Section 3.5).

We will use the term *photo* to refer to a standard limited-field-of-view image as taken with a normal camera (Figure 3-5), and the term *panorama* to denote a 360-degree full-view panoramic image (Figure 5-19(b)). We use the term *place* to refer to panoramic image categories, and use the term *scene category* to refer to semantic categories of photos. We make this distinction because a single panoramic place category can contain views corresponding to many different scene categories (for example, *street* and *shopfront* are treated as two different scene categories in [184], but they are just different views of a single place category, *street*).

[1] The panorama dataset covers 15 indoor categories: church, hotel room, subway station, theater, train interior, corridor, living room, shop, restaurant, lobby atrium, museum, expo showroom, old building, cave and workshop; and 11 outdoor categories: beach, street, wharf, coast, lawn, plaza courtyard, field, mountain, forest, park, and ruin.

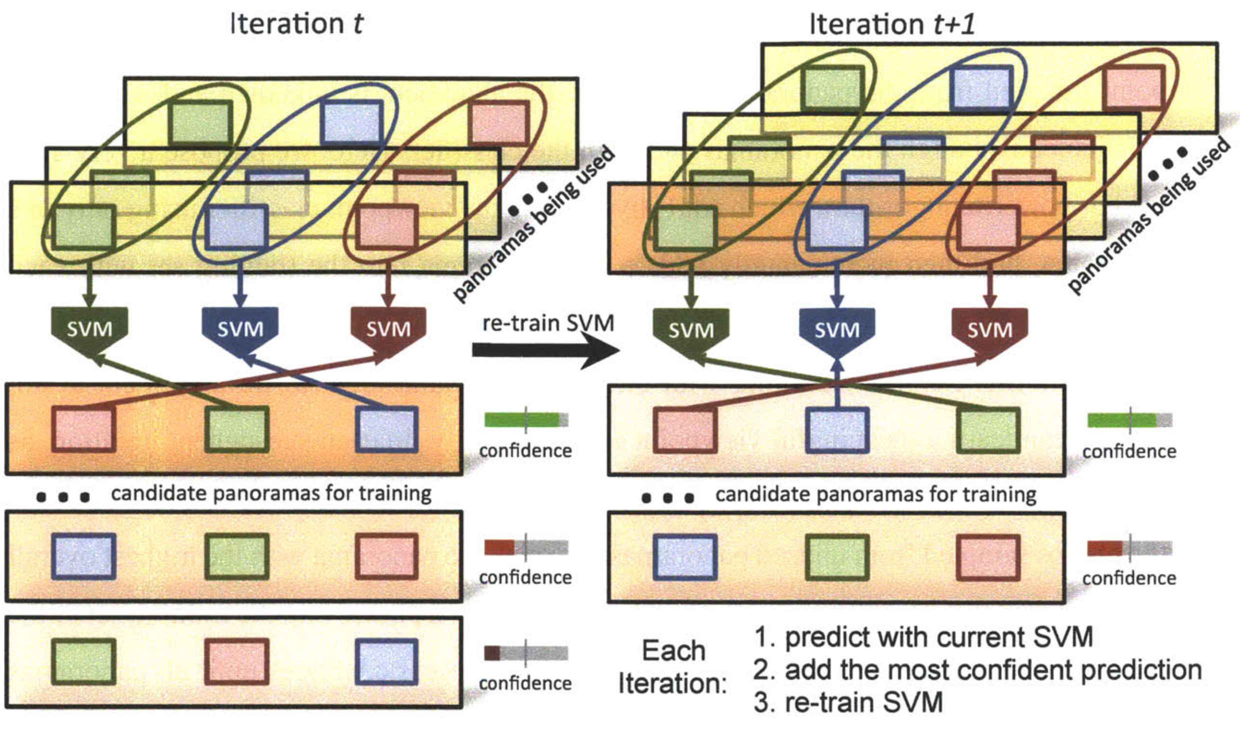


Figure 3-3: Incremental algorithm for simultaneous panorama alignment and viewpoint classifier training. Each long rectangle denotes a panorama, and each smaller rectangle inside the panorama denotes a limited-field-of-view photo generated from the panorama.

3.2 Place category classifier

We use a non-linear kernel Support Vector Machine (SVM) to train a 26-way classifier to predict the place category of a photo. The training data are the 12 limited-field-of-view photos generated from each panorama in each category. Although we regard all different viewpoints from the same category as the same class in this stage, the kernel SVM with non-linear decision boundary nevertheless gives very good partition of positives and negatives. Refer to Section 3.8.6 for details on the dataset construction, features, kernels and classifiers that we used.

3.3 Panorama alignment & viewpoint classifier

For each place category, we align the panoramas for training, and train the viewpoint recognition model. For each category, if we know the alignment of panoramas, we can train a

m-way classifier for m viewpoints, using the limited-field-of-view photos for each viewpoint sampled from the panoramas. However, the alignment is unknown and we need to simultaneously align the panoramas and train the classifier. Here, we propose a very simple but powerful algorithm, which starts by training the viewpoint classifier using only one panorama, and then incrementally adds a new panorama into the training set with each iteration.

At the first iteration, training with only one panorama requires no alignment at all, and we can learn a meaningful viewpoint classifier easily. In each subsequent iteration, as illustrated in Figure 5-23, we use the current classifier to predict the viewpoint for the rest of the photos sampled from unused panoramas. We pick the panorama with the highest overall confidence in the classifier prediction, add all its generated photos into the training set of the classifier, and retrain the classifier again. The same process continues until all panoramas have been used for training. This produces a viewpoint classifier and an alignment for all panoramas at the same time.

This process exploits the fact that the most confident prediction of a nonlinear kernel SVM classifier is usually correct, and therefore we maintain a high-quality alignment for re-training a good viewpoint classifier at each iteration. Of course, starting with a different panorama results in a different alignment and model. Therefore, we exhaustively try each panorama as the starting point and use cross validation to pick the best model.

In each iteration, the viewpoint classifier predicts results for all m limited-field-of-view photos generated from the same panorama, which have known relative viewpoints. Hence, we can use this relative viewpoint as a hard constraint to help choose the most confident prediction and assign new photos into different viewpoints for re-training. Furthermore, in each iteration, we also adjust all the previously used training panoramas according to the current viewpoint classifier's predictions. In this way, the algorithm is able to correct mistakes made in earlier iterations during the incremental assignment.

When selecting the most confident prediction, we use the viewpoint classifier to predict on both the original panorama and a horizontally-flipped version of the same image, and pick the one with higher confidence[2]. This allows the algorithm to handle places which

[2]In order to avoid adding an artificial symmetry structure to the data, only one of the original panorama

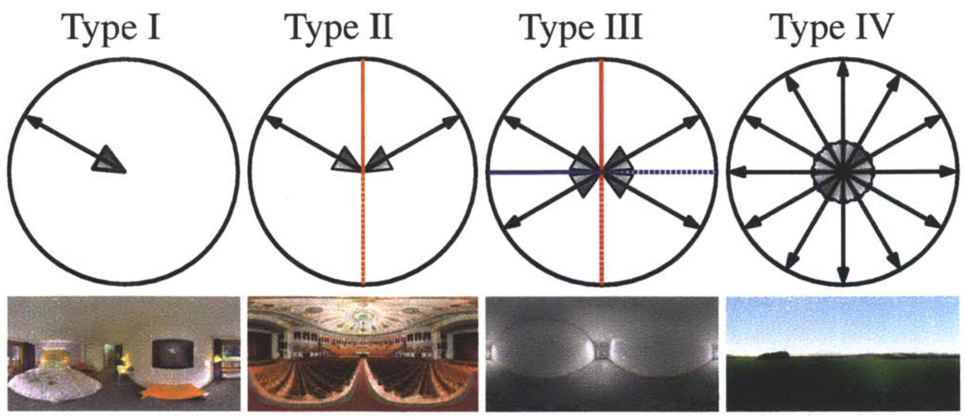


Figure 3-4: Four types of symmetry structure found in place categories: Type 1 (Asymmetry), Type 2 (Bilateral symmetry with one axis); Type 3 (Bilateral symmetry with two axis); Type 4 (Isotropic symmetry). Each circle represents a panorama as seen from above, arrows represent camera viewpoints that are similar, and **red** and **blue** lines denote symmetry axes. The second row shows example images for each type of symmetry. See Section 3.4 for a detailed description.

have two types of layouts which are 3D mirror reflection of each other. (For example, in a hotel room, the bed may be located to the left of the doorway or to the right of the doorway – the spatial layout of the room is the same, only flipped.) Because we give the algorithm the freedom to horizontally flip the panorama, these two types of layout can be considered as just one layout to train a better model with better alignment. The accuracy of the automatic panorama alignment procedure can be foreseen in Table 3.1 (see Sections 3.6 and 3.8.6, and Table 3.1 caption for details).

Testing Given a view of a place, our model can infer the semantic category of the place and identify the observer's viewpoint within that place (see Sections 3.6 and 3.8.6 for Implementation and Evaluation details, and Table 2).

We can even extrapolate the possible layout that extends beyond the available field of view, as illustrated in Figure 5-19(c) and 3-8. We represent the extrapolated layout using either the average panorama of the place category, or the nearest neighbour from the training examples. We can clearly see how correct view alignment allows us to predict visual information, such as edges, that extend beyond the boundaries of the test photo.

and the flipped panorama is allowed to participate in the alignment.

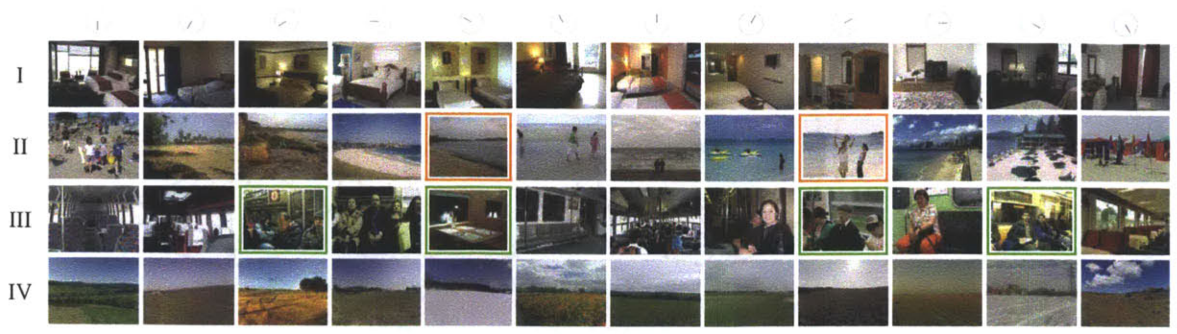


Figure 3-5: Example images across different viewpoints. Each row shows typical photos from 12 different viewpoints in a category: hotel room (Type I), beach (Type II), street (Type III), field (Type IV). The grey compasses on the top indicate the viewpoint depicted in each column. We can see how different views can share similar visual layouts due to symmetry. For example, the two photos with orange frame are a pair with type II symmetry, and the four photos with green frame are a group with type III symmetry.

3.4 Symmetry discovery and view sharing

Many places have a natural symmetry structure, and we would like to design the model to automatically discover the symmetry structure of the place categories in the data set. A description of place symmetry is a useful contribution to scene understanding in general, and it also allows us to borrow views across the training set, increasing the effective training set size. This does not solve the ambiguity inherent in recognizing views in places with underlying symmetry, but it will reduce the model's errors to other, irrelevant views. This is illustrated in Figure 3-9(c): the model trained with no sharing of training examples (top figure) has more errors around 90°, and hence less frequent detections on 0°and 180°. With sharing of training examples (bottom figure), the errors at 90° are reduced, and the frequency of detections on 0°and 180° are increased, which means that the accuracy is increased. Finally, understanding the symmetry structure of places allows us to train a simpler model with fewer parameters, and a simpler model is generally preferred to explain data with similar goodness of fit. Here, we consider four common types of symmetry structure found in place categories, shown in Figure 3-4:

Type I: Asymmetry. There is no symmetry in the place. Every view is unique.

Type II: Bilateral Symmetry with One Axis. Each view matches a horizontally-flipped view that is mirrored with respect to an axis.

Type III: Bilateral Symmetry with Two Axes. Same as Type III but with two axes that are perpendicular to each other. This type of symmetry also implies 180° rotational symmetry.

Type IV: Isotropic Symmetry. Every view looks the same.

To allow view sharing with symmetry, we need to modify the alignment and viewpoint classifier training process proposed in the previous section 4, for each type of symmetry as follows:

Type I: The algorithm proposed in the previous section can be applied without any modification in this case.

Type II: Assuming we know the symmetric axis, we need to train a m-way classifier as with Type I. But each pair of symmetric views can share the same set of training photos for training, with the appropriate horizontal flips to align mirror-image views. Denote A and B as symmetric views under a particular axis, and the photos $\{I_A\}$ and $\{I_B\}$ are their respective training examples. And denote $h(I)$ as a function to horizontally flip a photo I. Then, we can train model of viewpoint A using not only $\{I_A\}$, but also $\{h(I_B)\}$. Same for B which we will train model of viewpoint B using photos $\{h(I_A), I_B\}$. If the viewpoint happens to be on the symmetric axis, i.e. $A = B$, we can train the model using photos $\{I_A, h(I_A)\}$. However, all this assumes that the symmetric axis is known. Therefore, we use exhaustive approach to learn one extra degree of freedom for the place category – the location of the symmetry axis. In each iteration, for each of the $\frac{m}{2}$ possible symmetric axes (axes are assumed to align with one of the m viewpoints), we re-train the model based on the new training set and the symmetric axes. We use the classifier to predict on the same training set to obtain the confidence score, and choose the axis that produces the highest confidence score from the trained viewpoint classifier in each iteration.

Type III: This type of place symmetry indicates that each view matches the opposite view that is 180° away. Therefore, instead of training a m-way viewpoint classifier, we only need to train a $(\frac{m}{2})$-way viewpoint classifier. Denote that views A and B are symmetric under one axis, A and C are symmetric under the other perpendicular axis, A and D are opposite

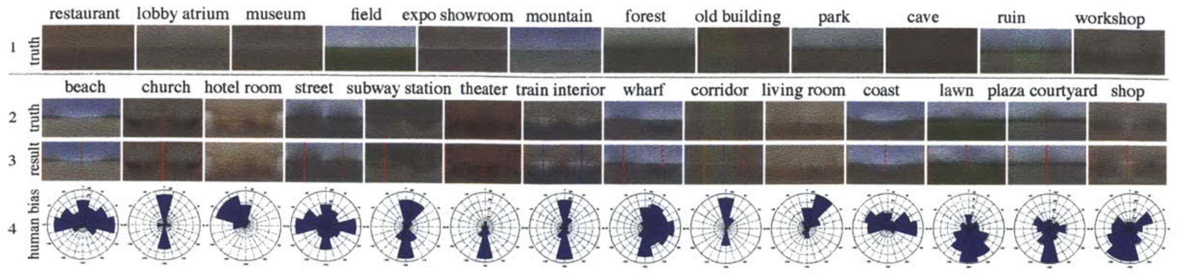


Figure 3-6: The first three rows show the average panoramas for the 26 place categories in our dataset. At top are the Type IV categories (all views are similar, so no alignment is needed). Below are the 14 categories with Symmetry Types I, II and III, aligned using manual (top) or automatic (bottom) alignment. Discovered symmetry axes are drawn in red and blue lines, corresponding to the red and blue lines of symmetry shown in Figure 3-4. The 4th row shows the view distribution in the SUN dataset for each place category; the top of each polar histogram plot corresponds to the center of the averaged panorama.

views that are 180° away, and $\{I_A\}, \{I_B\}, \{I_C\}, \{I_D\}$ are their respective training examples. Then, we will train a viewpoint model for A using examples $\{I_A, h(I_B), h(I_C), I_D\}$. If the view is on one of the symmetric axis, i.e. $A = B$, we have $C = D$, and we will train the model for A using examples $\{I_A, h(I_A), h(I_D), I_D\}$. Because we know that the two axes of symmetry are perpendicular, there is actually only one degree of freedom to learn for the axes during training, which is the same as Type II and the same exhaustive approach for symmetric axis identification is applied. But instead of $\frac{m}{2}$ possible symmetric axes, there are only $\frac{m}{4}$ possibilities due to symmetry.

Type IV: The viewpoint cannot be identified given the isotropic symmetry, so there is no need for alignment or a viewpoint classification model. The optimal prediction is just random guessing.

To automatically decide which symmetry model to select for each place category, i.e. discover the symmetry structure, we use cross validation on the training set to pick the best symmetry structure. If there is a tie, we choose the model with higher symmetry because it is simpler. We then train a model with that type of symmetry using all training data. Results (see Sections 3.6 and 3.8.6 for implementation and estimation details) are displayed in Tables 3.1 and 3.2. Discover which type of symmetry (I, II, III or IV) each place category displays, allows to reduce the model's errors to irrelevant views.

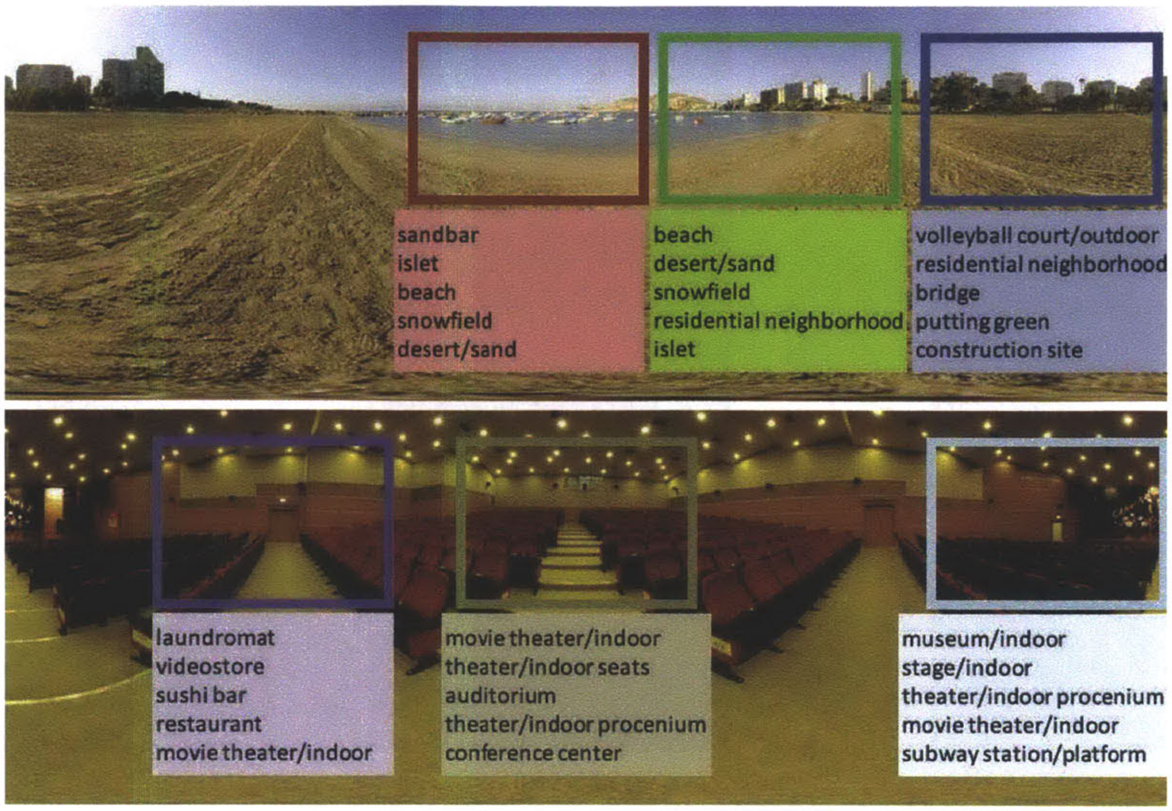


Figure 3-7: Prediction of SUN categories on different viewpoints. Each rectangle denotes a view in the panorama and the text below gives the top five predictions of the SUN category classifier, using [184] as training data. (The rectangles are purely for illustration – proper image warping was used in the algorithm.)

3.5 Canonical view of scenes

It is well known that objects usually have a certain "canonical view" [127]. Recent work also suggests that people also show preferences for particular views of places [38]. Here we further study the canonical views of places. More specifically, we are interested in where people take photos in a place, i.e. which viewpoint they choose for a scene when they want to take a photo.

To study the preferred canonical viewpoint of scenes, we use the publicly available SUN dataset for scene classification from [184], manually selecting scene categories that corresponded to our panoramic scene categories. We obtain the viewpoint information for each photo by running a view-matching task on Amazon Mechanical Turk. Workers were shown a SUN photo and a representative panorama from the same general place category.

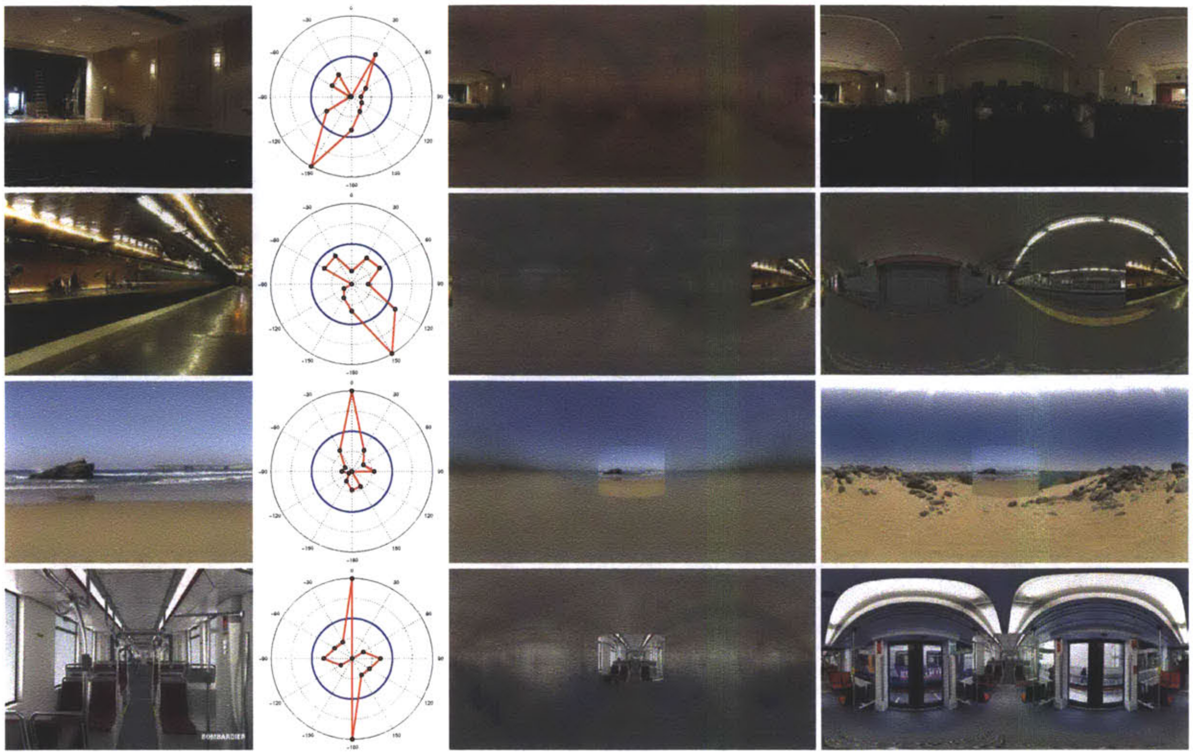


Figure 3-8: Visualization of results. In each row, the first image is the input photo used to test the model. The second figure visualizes of the scores from SVM indicating the most likely orientation of the photo. The blue circle is the zero crossing point, and the red polyline is the predicted score for each orientation. The next two images illustrate the predicted camera orientation and the extrapolated scene layout beyond the input view (the left image extrapolates the layout using the average panorama from the place category; the right image uses the nearest neighbor from the training set.) Please refer to the supplementary materials for more results.

The panorama was projected in an Adobe Flash viewer which showed a 65.5° view of scene (as it might appear through a standard camera), and workers could rotate the view within the viewer to match the view shown in the SUN photo.

For each photo, we obtained votes from several different workers with quality control. Specifically, turkers were asked to rotate the view in the Flash viewer to match the individual image shown. Figure 3-5 shows examples of SUN database images which workers mapped to each viewpoint of a corresponding panoramic place category. In each category, some views were more common in the SUN database than others, and we visualize this view bias in the 4th row of Figure 3-6. There are clearly biases specific to the place category, such as a preference for views which show the bed in the hotel room category.

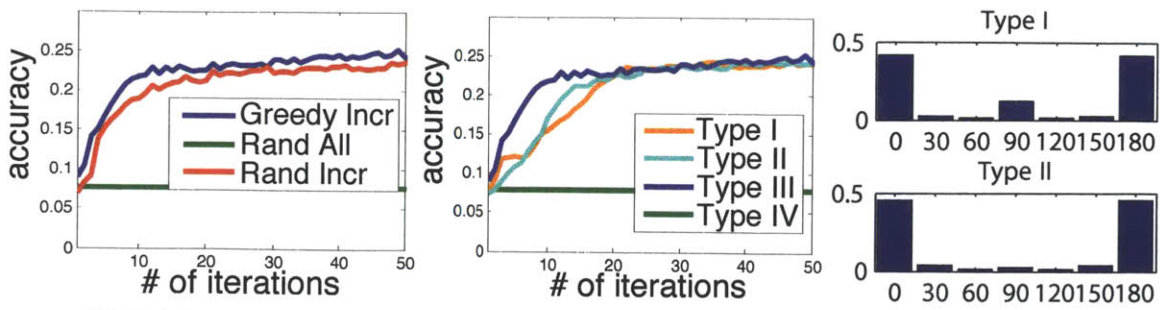
To further illustrate how each view is correlated with specific scene categories, we train a scene classifier using the SUN database from [184], which covers 397 common scene categories. We use this 397-way classifier to predict on each viewpoint and get an average response from all photos generated from our panorama dataset. We show some examples in Figure 3-7. For instance, the beach views which show the sea are predicted to be beach-related categories (sandbar, islet, beach, etc.), while the opposite views are predicted to be non-beach categories (volleyball court and residential neighborhood).

3.6 Implementation details

Dataset construction We downloaded panoramas from [1], and selected 26 place categories for which there were many different exemplars available. The final dataset has a total of 6161 panoramas. The panoramas have a resolution of 9104×4552 pixels and cover a full $360° \times 180°$ visual angle using equirectangular projection [161]. To obtain viewpoint ground truth, panoramas from the 14-place category of symmetry types I, II, and III were manually aligned by the authors, by selecting a consistent key object or region for all the panoramas of the category. Row 2 of Figure 3-6 shows the resulting average panorama for each place category after manual alignment. No alignment was needed for the 4th symmetry type (row 1 of Figure 3-6). The averaged panoramas reveal some common key structures for each category, such as the aligned ocean in the beach panoramas and the corridor structure in the subway station.

We tested the algorithm on two sets of limited-field-of-view photos. First, we generated 12 photos from each panorama, for a total of 73,932 photos. Each photo covers a horizontal angle of 65.5°, which corresponds to 28mm focal length for a full-frame Single-Lens Reflex Camera (typical parameters for digital compact cameras). We also constructed a second testing dataset using the photos from the SUN dataset [184] (Section 3.5).

Features and classifiers We select several popular state-of-art features used in scene classification tasks, including GIST [126], dense SIFT [113, 104], HOG [27, 52, 184], texton [117], geometric map (GeoMap) and texton statistics (GeoTexton) [66], self-similarity



(a) Training strategies. (b) Different symmetries. (c) Angle deviation.

Figure 3-9: Illustration for the algorithm behavior for the place category *street*. (a) compares different training strategies: our proposed greedy incremental algorithm, a random incremental algorithm that adds a random panorama, and a totally random algorithm that adds everything at once. (b) illustrates the performance of algorithms with different symmetry assumptions. (c) shows that although incorporating symmetry does not resolve ambiguity (0° and 180°), it does reduce random mistakes (e.g. 90°).

(SSIM)[151], local binary pattern (LBP) [124], and 32×32 tiny image [168] as baseline. For dense SIFT, HOG, texton and SSIM, we construct a visual words vocabulary of 300 centroids using k-means to have a 1×1, 2×2, and 4×4 spatial pyramid histogram [104] as descriptor of the photo. Based on the benchmark of [184], we choose to use histogram intersection kernel for HOG, dense SIFT and LBP, RBF kernel for GeoMap and TinyImage, χ^2 kernel for texton, GeoTexton and SSIM, and e^{χ^2} kernel for GIST. We use a linear weighted sum of the kernel matrices as the final combined kernel, using the weights suggested by [184]. The same features and kernels are also used by the viewpoint prediction classifier. We use One-versus-Rest SVM as our multi-class classifier for both place category classification and viewpoint classification, because, empirically, it outperforms all other popular classifiers and regressors in our experiments.

Algorithm behavior analysis Figure 3-9 shows an example run on the street place category which characterizes the behavior of our algorithm. Figure 3-9(a) shows how different strategies of iterative training affect the convergence results: incremental addition of training examples gives much better results than starting with all examples (Rand All), and greedy adding the most confident examples (Greedy Incr) gives better performance than adding examples in a random order (Rand Incr). Figure 3-9(b) compares the four types of symmetry structure with respect to the testing accuracy. We can see that view sharing

Evaluation	HOG-S	HOG-L	Tiny	HOG	COM	**Final**	Chance
Accuracy	41.8	45.0	25.9	56.4	62.2	**69.7**	8.3
Deviation	65.6°	62.5°	73.4°	48.0°	41.4°	**34.5°**	90.0°

Table 3.1: Performance of automatic panorama alignment. HOG-S and HOG-L are two baseline algorithms for comparison (Section 3.7.2). Tiny, HOG and COM are the results that do NOT make use of symmetry and view sharing (Section 3.3), using various features presented in Section 3.6. Final is our complete model with symmetry and view sharing (Section 3.4).

from symmetry structure is most beneficial when the number of training examples is low (in early iterations), but as more training examples are introduced in later iterations, all models converge to similar accuracy. Figure 3-9(c) shows the histogram of angle deviation from the truth. We can clearly see the ambiguity due to the symmetry structure of the street place category. Type I (top figure) performs slightly worse than Type II (bottom figure), by making more errors on angles between 0° and 180°. Our algorithm is related to curriculum learning, k-means and Expectation-Maximization (EM), and can be interpreted as Latent Structural SVM with Concave-Convex Procedure. Please refer to the supplementary materials for mathematical analyses and further discussions.

3.7 Evaluation

3.7.1 Evaluation methods

Place categorization performance is evaluated by determining the accuracy per category. Viewpoint prediction can also be evaluated by classification accuracy if the panoramas are manually aligned.

However, because we use unsupervised alignment of the panoramas to train the viewpoint predictor from the aligned result, we cannot know which categories in the alignment result correspond to which categories in the labelled truth for evaluation. Similar to Unsupervised Object Discovery [172], we use an *oracle* to assign each aligned viewpoint to one of the m view directions from the truth, and then evaluate the accuracy based on the resultant direction-to-direction mapping. Due to circular constraints, the total number of all possible solutions is only m, so we can try all of them to search for the best direction-

Test Set	Manual Alignment				Automatic Alignment					
	Accuracy 1		Accuracy 2		Accuracy		Angle Deviation			
	Place	Both	Place	Both	Place	Both	I	II	III	IV
Panorama	48.5	23.6	51.9	23.8	51.9	**24.2**	55°	51°	86°	90°
SUN[184]	22.2	13.5	24.1	13.0	24.1	**13.9**	29°	30°	38°	90°
Chance	3.8	0.3	3.8	0.3	3.8	0.3	90°	90°	90°	90°

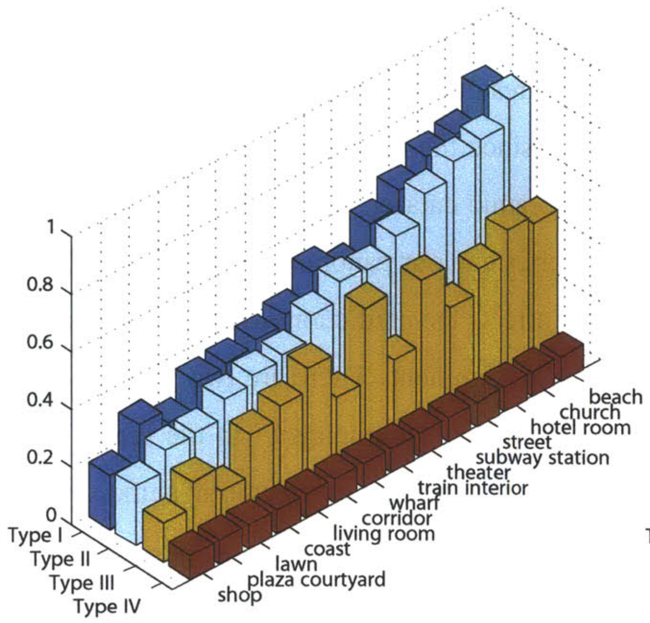
Table 3.2: Testing accuracy and average viewpoint deviation. We compare the performance of our automatic alignment algorithm with manual alignment. For the manual alignment, we design two algorithms for comparison (Section 3.7.3): a 1-step algorithm (Accuracy 1) and a 2-step algorithm (Accuracy 2). For each algorithm, "Place" is the accuracy of place classification, and "Both" is the accuracy requiring both correct place category prediction and correct viewpoint prediction, and is the final result. We also show the average angle deviation assuming different symmetry types.

to-direction mapping for evaluation.

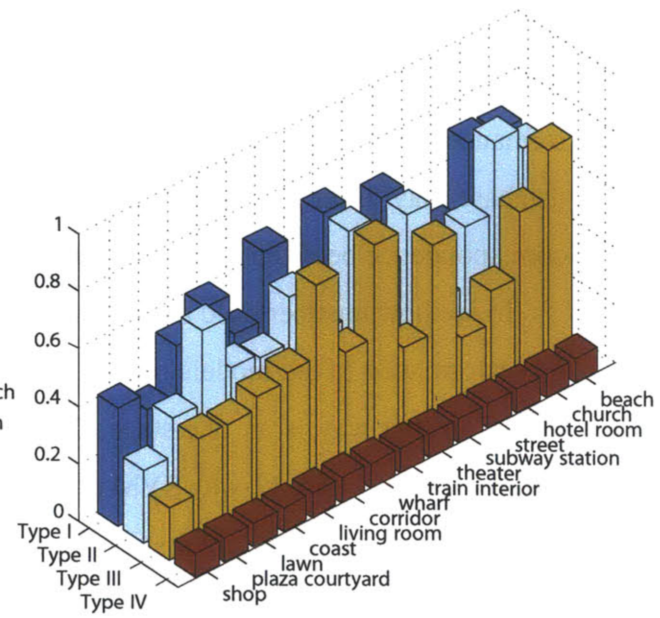
Besides viewpoint prediction, another natural evaluation metric is the average viewpoint deviation from the truth to the prediction. However, due to the symmetry structure of certain types of places, the viewpoint may be ambiguous, and the average angle deviation is not always a good metric. For example, in a street scene, there are usually two reasonable interpretations for a given view, 180° apart, so the viewpoint deviations cluster at 0° and 180°(as shown in Figure 3-9(c)). This means the average viewpoint deviation is about 90°, the expected value for chance performance. In addition to the prediction results, we also need to evaluate the automatic alignment results during training. Again, we use the accuracy with the oracle, and average viewpoint deviation for this evaluation.

3.7.2 Evaluation on training

We summarize the automatic alignment results obtained during training in Table 3.1. We use the manual panorama alignment produced by an author as the ground truth for evaluation. We design a baseline algorithm of panorama alignment for comparison: For each pair of panoramas, we try all m alignment possibilities. For each possibility, we extract HOG features [27, 52], and use the sum of squared differences of the two feature maps to look for the best alignment between these two panoramas. Then we use the panorama with the minimum difference from all other panoramas as the centroid, and align the other panoramas to this common panorama. We tried two different cell sizes for HOG – 8 pixels



(a) Test on Panorama Dataset



(b) Test on SUN Dataset

Figure 3-10: Viewpoint prediction accuracy assuming different symmetry types.

and 40 pixels, labelled HOG-S and HOG-L in the table.

We also study the performance of different image features using our algorithm, including TinyImage and HOG, and COM (the combined kernel from all features). Furthermore, we compare the performance between no sharing of training views (i.e. always assuming Type I symmetry), and sharing using the discovered symmetry type (i.e. algorithm decides the symmetry type). We can see in Table 1 that our complete model (final) using all features and automatic discovered symmetry performs the best (accuracy of 69.7 % vs. 62.2 % without symmetry). The average panoramas from the automatic alignment algorithm, shown in the third row (result) of Figure 3-6, look very similar to the averages produced by manual alignment in the second row (truth).

3.7.3 Evaluation on testing

We evaluate our testing result on two test sets: limited-field-of-view photos generated from our panorama dataset, and real-world, camera-view photos from SUN dataset. The performance is reported in Table 3.2, which shows accuracy and average viewpoint deviation. We evaluate the place categorization accuracy without taking viewpoint into account in the table as “Place”. “Both” is the accuracy requiring both correct place category prediction and correct viewpoint prediction, and it is the final result. Note that we expect lower performance when testing on the SUN photos [184], since the dataset statistics of the SUN database may differ from the statistics of our panoramic dataset, and there is not an exact correspondence between the SUN database categories and our panoramic place categories.

To see how unsupervised automatic alignment affects the prediction result, we compare its performance to the models trained using manual alignment. There are two ways to do this. The first way is an one-stage algorithm (denoted as “Accuracy 1” in the table), in which we train a 26×12-way SVM to predict the place category and viewpoint at the same time. The second way is a two-stage algorithm (“Accuracy 2”) to first train a 26-way SVM to predict the place category, and then, for each category, train a 12-way SVM to predict the viewpoint. Compared to the models trained from manual alignment, our model trained with unsupervised automatic alignment performs slightly better on our panorama dataset, but slightly worse on the SUN dataset. Overall, our automatic alignment performs very well, producing results comparable to manual alignment (see Table 2). Furthermore, we can evaluate the view prediction accuracy assuming the place categorization is correct. We compare the four types of symmetry structure for each place category in Figure 3-10. We can see that imposing correct symmetry structure is usually helpful, even when the imposed symmetry is incomplete. Imposing incorrect types of symmetry always hurts performance.

3.8 Application: dramatic view extrapolation

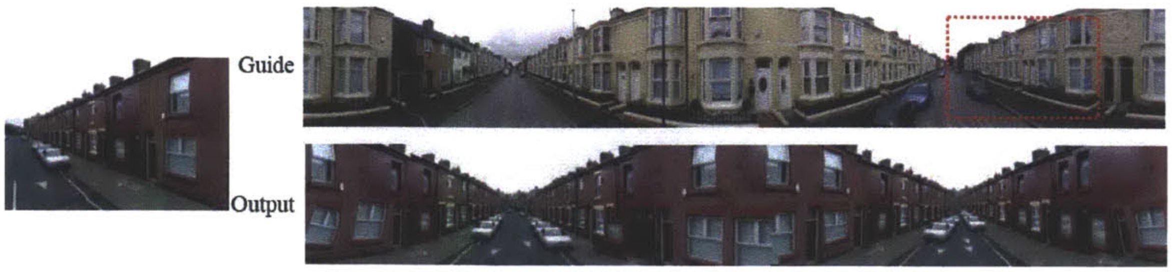


Figure 3-11: Our method can extrapolate an image of limited field of view (left) to a full panoramic image (bottom right) with the guidance of a panorama image of the same scene category (top right). The input image is roughly aligned with the guide image as shown with the dashed red bounding box.

When presented with a narrow field of view image, humans can effortlessly imagine the scene beyond the particular photographic frame. In fact, people confidently remember seeing a greater expanse of a scene than was actually shown in a photograph, a phenomena known as "boundary extension" [84]. In the computational domain, numerous texture synthesis and image completion techniques can modestly extend the apparent field of view (FOV) of an image by propagating textures outward from the boundary. However, no existing technique can significantly extrapolate a photo because this requires implicit or explicit knowledge of scene layout. Recently, Xiao et al. [184] introduced the first large-scale database of panoramic photographs and demonstrated the ability to align typical photographs with panoramic scene models. Inspired by this, we ask the question: is it possible to dramatically extend the field of view of a photograph with the guidance of a representative wide-angle photo with similar scene layout?

Specifically, we seek to extrapolate the FOV of an input image using a panoramic image of the same scene category. An example is shown in Figure 3-11. The input to our system is an image (Figure 3-11, left) roughly registered with a guide image (Figure 3-11, top). The registration is indicated by the red dashed line. Our algorithm extrapolates the original input image to a panorama as shown in the output image on the bottom right. The extrapolated result keeps the scene specific structure of the guide image, *e.g.* the two vertical building facades along the street, some cars parked on the side, clouds and sky on the top, etc. At the same time, its visual elements should all come from the original input image so that it appears to be a panorama image captured at the same viewpoint. Essentially, we need to learn the shared scene structure from the guide panorama and apply it to the input image to create a novel panorama.

We approach this FOV extrapolation as a constrained texture synthesis problem and address it under the framework of shift-map image editing [135]. We assume that panorama images can be synthesized by combining multiple shifted versions of a small image region with limited FOV. Under this model, a panorama is fully determined by that region and a shift-map which defines a translation vector at each pixel. We learn such a shift map from a guide panorama and then use it to constrain the extrapolation of a limited FOV input image. Such a guided shift-map can capture scene structures that are not present in the small image region, and ensures that the synthesized result adheres to the layout of the guide image.

Our approach relies on understanding and reusing the long range self-similarity of the guide image. Because a panoramic scene typically contains surfaces, boundaries, and objects at multiple orientations and scales, it is difficult to sufficiently characterize the self-similarity using only patch translations. Therefore we generalize the shift-map method to optimize a general similarity transformation, including scale, rotation, and mirroring, at each pixel. However, direct optimization of this "similarity-map" is computationally prohibitive. We propose a hierarchical method to solve this optimization in two steps. In the first step, we fix the rotation, scaling and reflection, and optimize for the best translation at each pixel. Next, we combine these intermediate results together with a graph optimization similar to photomontage [5].

3.8.1 Related work

Human vision. Intraub and Richardson [84] presented observers with pictures of scenes, and found that when observers drew the scenes according to their memory, they systematically drew more of the space than was actually shown. Since this initial demonstration, much research has shown that this effect of "boundary extension" appears in many circumstances beyond image sketching. Numerous studies have shown that people make predictions about what may exist in the world beyond the image frame by using visual associations or context [13] and by combining the current scene with recent experience in memory [114]. These predictions and extrapolations are important to build a coherent percept of the world [74].

Environment map estimation from a single image. Rendering techniques rely on panoramic environment maps to realistically illuminate objects in scenes. Techniques such as [102, 88] estimate environment maps from single images in order to manipulate material properties and insert synthetic objects in existing photographs. In both cases, the synthesized environment maps are not very realistic, but they do create plausible models of incident light. Our technique could be used to generate higher quality environment maps for more demanding rendering scenarios (e.g. smooth and reflective objects).

Inpainting. Methods such as [16, 108, 15] solve a diffusion equation to fill in narrow image holes. Because they do not model image texture in general, these methods cannot convincingly synthesize large missing regions. Further, they are often applied to fill in holes with known, closed boundaries and are less suitable for FOV extension.

Texture synthesis. Example based texture synthesis methods such as [37, 36] are inherently image extrapolation methods because they iteratively copy patches from known regions to unknown areas. More sophisticated optimization methods [100] preserve texture structure better and reduce seam artifacts. These techniques were applied for image completion with structure-based priority [25], hierarchical filtering [35] and iterative optimization [182]. Hertzmann et al. [73] introduced a versatile "image analogies" framework to transfer the stylization of an image pair to a new image. Shift-map image editing [135] formulates image completion as a rearrangement of image patches. Kopf et al. [96] extrapolate image boundaries by texture synthesis to fill the boundaries of panoramic mosaics. Poleg and Peleg [133] extrapolate individual, non-overlapping photographs in order to compose them into a panorama. These methods might extrapolate individual images by as much as 50% of their size, but we aim to synthesize outputs which have 500% the field of view of input photos.

Hole-filling from image collections. Hays and Efros [65] fill holes in images by finding similar scenes in a large image database. Whyte et al. [183] extend this idea by focusing on instance-level image completion with more sophisticated geometric and photometric

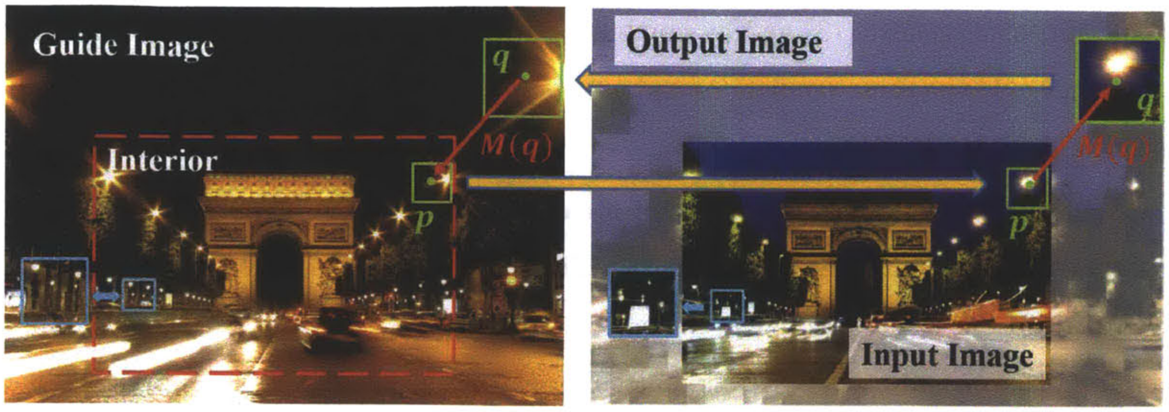


Figure 3-12: Baseline method. Left: we capture scene structure by the motion of individual image patches according to self-similarity in the guide image. Right: the baseline method applies these motions to the corresponding positions of the output image for view extrapolation.

image alignment. Kaneva et al. [93, 92] can produce infinitely long panoramas by iteratively compositing matched scenes onto an initial seed. However, these panoramas exhibit substantial semantic "drift" and do not typically create the impression of a coherent scene. Like all of these methods, our approach relies on information from external images to guide the image completion or extrapolation. However, our singular guide scene is provided as input and we do not directly copy content from it, but rather learn and recreate its layout.

3.8.2 Overview

Our goal is to expand an input image I_i to I with larger FOV. Generally, this problem is more difficult than filling small holes in images because it often involves more unknown pixels. For example, when I is a full panorama, there are many more unknown pixels than known ones. To address this challenging problem, we assume a guide image I_g with desirable FOV is known, and I_i is roughly registered to I_g^i (the "interior" region of I_g). We simply reuse I_i as the interior region of the output image I. Our goal is to synthesize the exterior of I according to I_i and I_g.

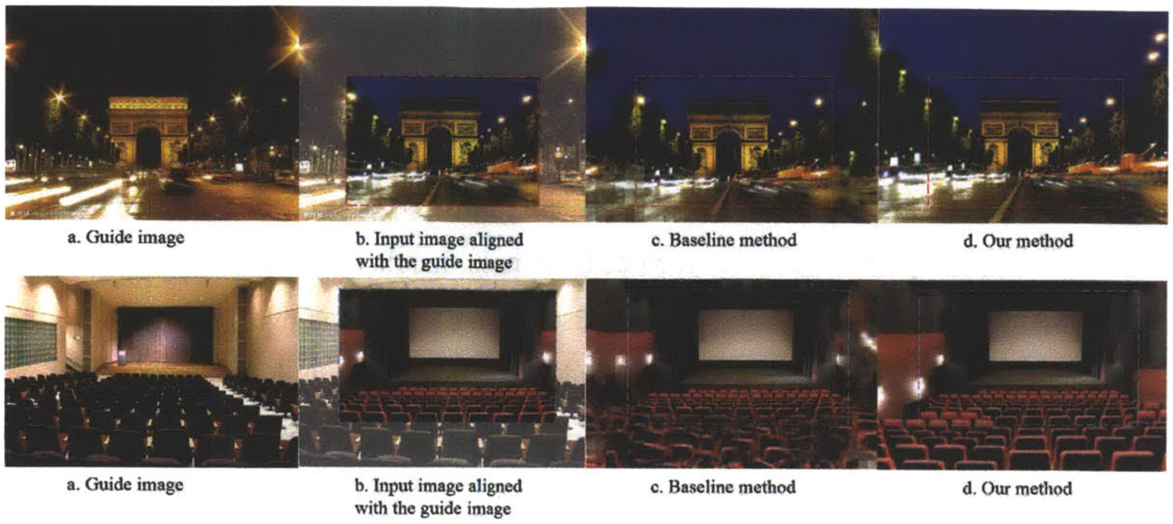


Figure 3-13: Arch (the first row) and Theater example (the second row). (a) and (b) are the guide image and the input image respectively. (c) and (d) are the results generated by the baseline method and our guided shift-map method.

3.8.3 Baseline method

We first describe a baseline algorithm. Intuitively, we need to learn the transformation between I_g^i and I_g, and apply it to I_i to synthesize I. This transformation can be modeled as the motions of individual image patches. Following this idea, as illustrated in Figure 3-12, for each pixel q in the exterior region of the guide image, we first find a pixel p in the interior region, such that the two patches centered at q and p are most similar. To facilitate matching, we can allow translation, scaling, rotation and reflection of these image patches. This matching suggests that the pixel q in the guide image can be generated by transferring p with a transformation $M(q)$, *i.e.* $I_g(q) = I_g(q \circ M(q))$. Here, $p = q \circ M(q)$ is the pixel coordinate of q after transformed by a transformation $M(q)$. We can find such a transformation for each pixel of the guide image by brute force search. As the two images I_i and I_g are registered, these transformations can be directly applied to I_i to generate the image I as $I(q) = I_i(q \circ M(q))$.

To improve the synthesis quality, we can further adopt the texture optimization [100] technique. Basically, we sample a set of grid points in the image I. For each grid point, we copy a patch of pixels from I_i centered at its matched position, as the blue and green boxes shown in Figure 3-12. Patches of neighboring grid points overlap with each other. Texture optimization iterates between two steps to synthesize the image I. First, it finds an optimal matching source location for each grid point according to its current patch. Second, it copies the matched patches over and averages the overlapped patches to update the image.

However, as shown in Figure 3-13 (c), this baseline does not generate appealing results. The results typically show artifacts such as blurriness, incoherent seams, or semantically incorrect content. This is largely because this baseline method is overly sensitive to the registration between the input and the guide image. In most cases, we can only hope to have a rough registration such that the alignment is semantically plausible but not geometrically perfect. For example, in the theater example shown in Figure 3-13, the registration provides a rough overlap between regions of chairs and regions of screen. However, precise pixel level alignment is impossible because of the different number and style of chairs. Such misalignment leads to improper results when the simple baseline method attempts to strictly recreate the geometric relationships observed in the guide image.

3.8.4 Generalized shift-map

To handle the fact that registration is necessarily inexact, we do not directly copy transformations computed from I_g according to the registration of I_i and I_g. Instead, we formulate a graph optimization to choose an optimal transformation at each pixel of I. Specifically, this optimization is performed by minimizing the following energy,

$$E(M) = \sum_q E_d(M(q)) + \sum_{(p,q)\in N} E_s(M(p),M(q)). \tag{3.1}$$

Here, q is an index for pixels, N is the set of all neighboring pixels. $E_d(\cdot)$ is the data term to measure the consistency of the patch centered at q and $q \circ M(q)$ in the guide image I_g. In other words, when the data term is small, the pixel q in the guide image I_g can be synthesized by copying the pixel at $q \circ M(q)$. Since we expect I to have the same scene structure as I_g (and I_i is registered with I_g^i), it is therefore reasonable to apply the same copy to synthesize q in I. Specifically,

$$E_d(M(q)) = ||R(q,I_g) - R(q \circ M(q),I_g)||_2. \tag{3.2}$$

$R(x, I)$ denotes the vector formed by concatenating all pixels in a patch centered at the pixel x of the image I.

$E_s(\cdot,\cdot)$ is the smoothness term to measure the compatibility of two neighboring pixels in the result image. The smoothness cost penalizes incoherent seams in the result image. It is defined as the following,

$$\begin{aligned} E_s(M(p), M(q)) &= \|I(q \circ M(q)) - I(q \circ M(p))\|_2 \\ &+ \|I(p \circ M(q)) - I(p \circ M(p))\|_2. \end{aligned} \tag{3.3}$$

If $M(q)$ is limited to translations, this optimization has been solved by the shift-map method [135]. He et al. [67] further narrowed down $M(q)$ to a small set of representative translations $\mathscr{M}$ obtained by analyzing the input image. Specifically, a translation M will be present in the representative translation set only if many image patches can find a good match by that translation. This set $\mathscr{M}$ captures the dominant statistical relationships between scene structures. In our case, we cannot extract this set from the input image I_i, because its FOV is limited and it does not capture all the useful structures. So we estimate such a set from the guide image I_g, and apply it to synthesize the result I from the input I_i, as shown in Figure 3-15. In this way, it ensures I to have the same structure as I_g. As our set of representative translations $\mathscr{M}$ is computed from the guide image, we call our approach the *guided shift-map method.*

However, in real images, it is often insufficient to just shift an image region to re-synthesize another image. Darabi et al. [28] introduced more general transformations such as rotation, scaling and reflection for image synthesis. So we also include rotation, scaling and reflection which makes $M(q)$ a general similarity transformation. This presents a challenging optimization problem.

3.8.5 Hierarchical optimization

Direct optimization of Equation 3.1 for general similarity transformations is difficult. Pritch et al. [135] introduced a multi-resolution method to start from a low resolution image and gradually move to the high resolution result. Even with this multi-resolution scheme, the

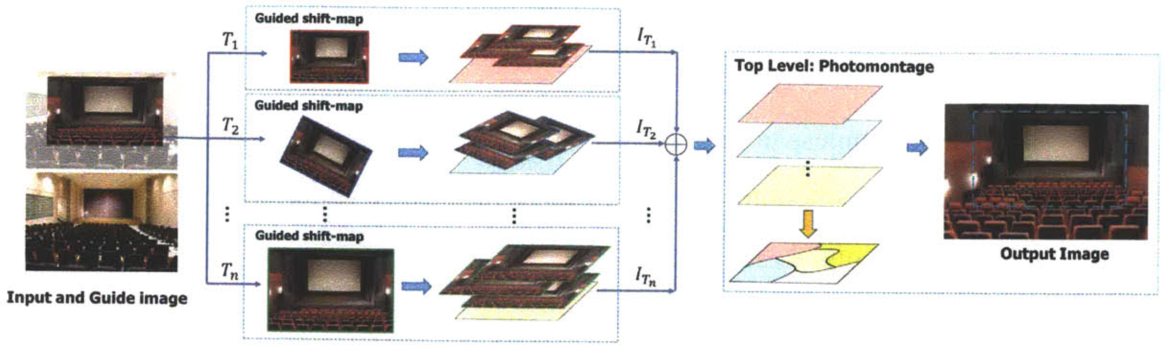


Figure 3-14: Pipeline of hierarchical optimization. We discretize a number of rotation, scaling and reflection. For each of the discretizd transformation T_i, we compute a best translation at each pixel by the guided shift-map method to generate I_{T_i}. These intermediate results are combined in a way similar to the Interactive Digital Photomontage [5] to produce the final output.

search space for $M(q)$ is still too large for general similarity transformations. We propose a hierarchical method to solve this problem in two steps. As shown in Figure 3-14, we first fix the rotation, scaling and reflection parameters and solve an optimal translation map. In the second step, we merge these intermediate results to obtain the final output in a way similar to Interactive Digital Photomontage [5].

Guided shift-map at bottom level

We represent a transformation T by three parameters r, s, m for rotation, scaling, and reflection respectively. We uniformly sample 11 rotation angles from the interval of $[-45^o, 45^o]$, and 11 scales from $[0.5, 2.0]$. Vertical reflection is indicated by a binary variable. In total, we have $11 * 11 * 2 = 242$ discrete transformations. For each transformation T, we use the guided shift-map to solve an optimal translation at each pixel. We still use $M(q)$ to denote the translation vector at a pixel q. For better efficiency, we further narrow down the transformation T to $20 \sim 50$ different choices. Specifically, we count the number of matched patches (by translation) for each discretized T, and only consider those T with larger number of matches.

Building representative translations As observed in [67], while applying shift-map image editing, it is preferable to limit these shift vectors to a small set of predetermined representative translations. So we use I_g to build a set of permissible translation vectors

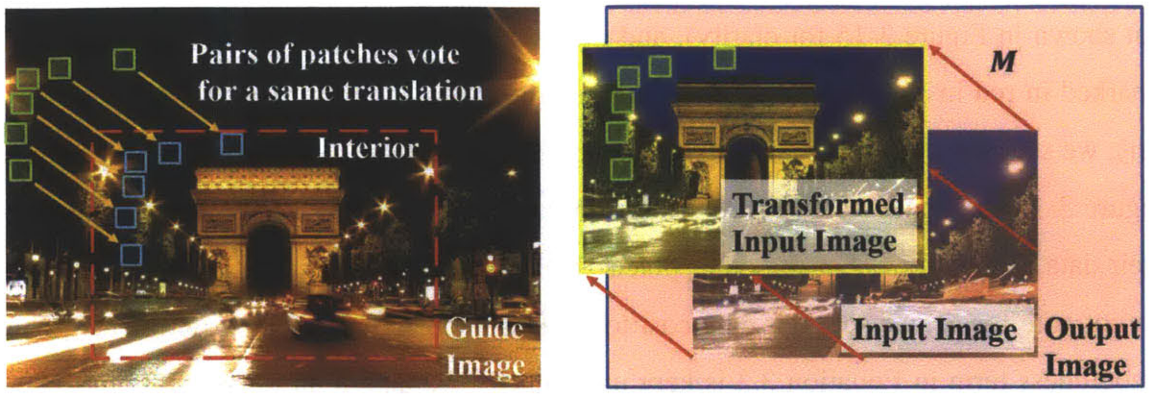


Figure 3-15: Left: in the guide image, the green patches vote for a common shift vector, because they all can find a good match (blue ones) with this shift vector; Right: The red rectangle is the output image canvas. The yellow rectangle represents the input image shifted by a vector voted by the green patches in the guide image. The data cost within these green patches is 0. The data cost is set to C for the other pixels within the yellow rectangle, and set to infinity for pixels outside of the yellow rectangle.

and apply them to synthesize I from I_i.

For each pixel q in the exterior of I_g, we search for its K nearest neighbors from the interior I_g^i transformed by T, and choose only those whose distance is within a fixed threshold. Each matched point p provides a shift vector $p-q$. We build a histogram of these vectors from all pixels in I_g. After non-maximum suppression, we choose all local maximums as candidate translations. For efficiency consideration, we choose the top 50 candidate translations to form the set of representative translations $\mathcal{M}_T$. In most experiments, more than 80% of the exterior pixels can find a good match according to at least one of these translations.

For the K nearest neighbor search, we measure the similarity between two patches according to color and gradient layout using 32×32 color patches and 31-dimensional HOG [52] features, respectively. We normalize the distance computed by color and HOG feature respectively according to the standard deviation of the observed distance.

Graph optimization We choose a translation vector at each pixel from the candidate set $\mathcal{M}_T$ by minimizing the graph energy Equation 3.1 with the guidance condition $M(q) \in \mathcal{M}_T$ for any pixel q. We further redefine the data term in Equation 3.2 as illustrated in Figure 3-15. For any translation $M \in \mathcal{M}_T$, the input image I_i is first transformed by T (which is not shown in Figure 3-15 for clarity), and then shifted according to M. For all the pixels (marked in red in Figure 3-15) that cannot be covered by the transformed I_i (yellow border), we set their data cost to infinity. We further identify those pixels (marked in green in Figure 3-15) that have voted for M when constructing the shift vector histogram, and set their data cost to zero. For the other pixels that can be covered by the transformed I_i but do not vote for M, we set their data cost to a constant C. $C = 2$ in our experiments. The smoothness term in Equation 3.3 is kept unchanged. We then minimize Equation 3.1 by alpha-expansion to find the optimal shift-map under the transformation T. This intermediate synthesis result is denoted by I_T.

Photomontage at top level

Once we have an optimal shift-map resolved for each transformation T, we seek to combine these results with another graph optimization. At each pixel, we need to choose an optimal transformation T (and its associated shift vector computed by the guided shift-map). This is solved by the following graph optimization

$$\mathbb{E}(T) = \sum_q \mathbb{E}_d(T(q)) + \sum_{(p,q)\in N} \mathbb{E}_s(T(p),T(q)). \tag{3.4}$$

Here, $T(q) = (r,s,m)$ is the selected transformation at a pixel q. The data term at a pixel q evaluates its synthesis quality under the transformation $T(q)$. We take all data costs and smoothness costs involving that pixel from Equation 3.1 as the data term $\mathbb{E}_d(T(q))$. Specifically,

$$\mathbb{E}_d(T(q)) = E_d^T(M^T(q)) + \sum_{p\in N(q)} E_s^T(M^T(p),M^T(q)).$$

Here, $M^T(q)$ is the optimal translation vector selected for the pixel q under the transformation T. $E_d^T(\cdot)$ and $E_s^T(\cdot,\cdot)$ are the data term and smoothness terms of the guide shift-map method under the transformation T. $N(q)$ is the set of pixels which neighbor q.

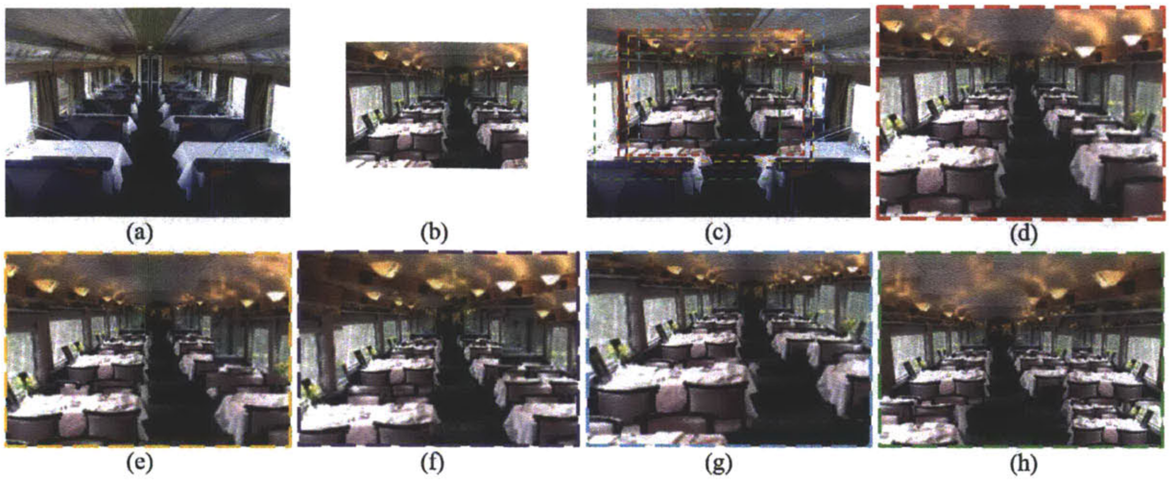


Figure 3-16: We evaluate our method with different registration between I_i and I_g. (a) and (b) are the guide and input images. (c) shows five different registrations. The red dashed line shows the manual registration. The others are generated by randomly shifting the manual registration for 5%, 10%, 15% and 20% of the image width. (d)–(h) are the five corresponding results. These results are framed in the same color as their corresponding dashed line rectangles.

The smoothness term is defined similar to Equation 3.3,

$$\begin{aligned}\mathbb{E}_s(T(p),T(q)) = &\|I_{T(p)}(q) - I_{T(q)}(q)\|_2 \\ &+ \|I_{T(p)}(p) - I_{T(q)}(p)\|_2.\end{aligned}$$

We then minimize the objective function Equation 3.4 by alpha-expansion to determine a transform T at each pixel. The final output at a pixel q is generated by transforming I_i with $T(q)$ and $M^T(q)$ and copying the pixel value at the overlapped position.

3.8.6 Experiments

We evaluate our method with a variety of real photographs. Given an input image I_i, we find a suitable I_g from the SUN360 panorama database [187] of the same scene category as I_i or we use an image search engine. We then provide a rough manual registration to align I_i and I_g and run our algorithm to generate the results.

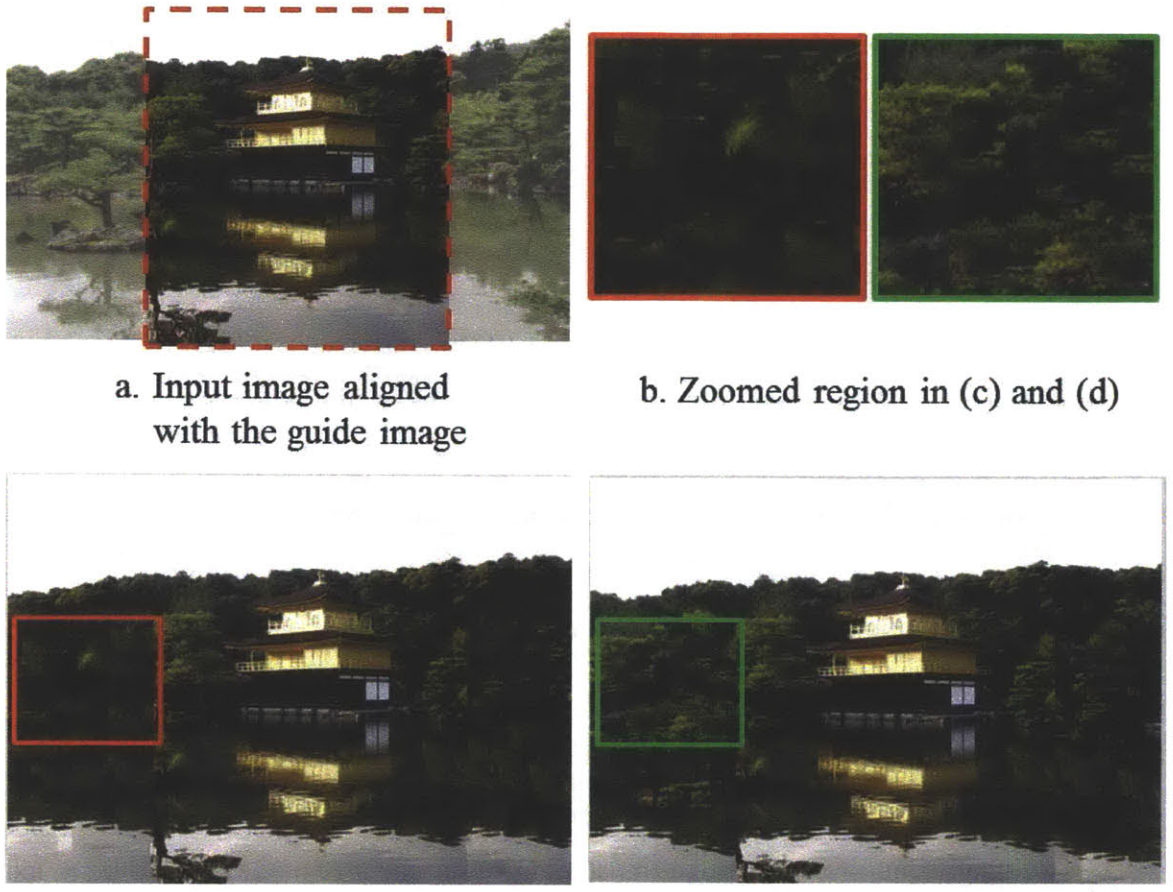
Comparison with the baseline method Figure 3-13 shows two examples comparing our method with the baseline method. Our method clearly outperforms the baseline method. In the theater example, although rough registration aligns semantically similar regions to the guide image I_g, directly applying the offset vectors computed in I_g to the I generates poor results. In comparison, our method synthesizes correct regions of chair and wall by accommodating the perspective-based scaling between exterior and interior in the $\mathscr{M}_T$. In the Arch example, some parts of the tree in the exterior region of the guide image match to patches in the sky in the interior region due to the similarity of patch feature (both HOG and color). As a result, part of the tree region is synthesized with the color of sky in the baseline method. Our method can avoid this problem by choosing the most representative motion vectors in the guide image and thus avoid such outliers. Both examples show that our method is more robust than the baseline method and does not require precise pixel level alignment. We also tested PatchMatch with the baseline method described in Section 3.8.3. While PatchMatch allows an almost perfect reconstruction of the guide image from its interior region, the resulting self-similarity field does not produce plausible extrapolations of the input image. In general, as more transformations are allowed, reconstruction of the guide image itself strictly improves (Equation 3.1), but the likelihood that these best transformations generalize to another scene decreases. In choosing which transformations to allow, there is a trade-off between expressiveness and robustness, and the similarity transforms we use seem to perform the best empirically. Typically, our method takes about 10 minutes to synthesize a 640 by 480 image. Most of the time is spent on K nearest neighbor search, for which numerous acceleration techniques are available.

Robustness to registration errors Our method requires the input image to be registered to a subregion of the guide image. Here, we evaluate the robustness of our method with respect to registration errors. Figure 3-16 shows an example with deliberately added registration error. We randomly shift the manually registered input image for 5–20% of the image width (600 pixels). The results from these different registrations are provided in the Figure 3-16 (d)–(h). All results are still plausible, with more artifacts when the registration error becomes larger. Generally, our method still works well for a registration error below

5% of image width. In fact, for this dining car example and most scenes, the "best" registration is still quite poor because the tables, windows, and lights on the wall cannot be aligned precisely. Our method is robust to moderate registration errors, as we optimize the transformations with the graph optimization.

Matching with HOG features Unlike most texture transfer methods, our approach compares image content with HOG features in addition to raw color patches. Figure 3-17 shows an example of how the recognition-inspired HOG can help our image extrapolation. Some patches in the foliage are matched to patches in the water in the guide image when the HOG feature is not used. This causes some visual artifacts in the result as shown in Figure 3-17 (c). The result with HOG feature is free from such problems as shown in Figure 3-17 (d). Please refer to the zoomed view in (b) for a clearer comparison.

Panorama Synthesis When I_g is a panoramic image, our method can synthesize I_i to a panorama. However, synthesizing a whole panorama at once requires a large offset vector space for voting to find representative translations. Also the size of $\mathscr{M}_T$ has to be much larger in order to cover the whole panorama image domain. Both of these problems require huge memory and computation. To solve this problem, we first divide the panoramic guide image I_g into several sub-images with smaller but overlapping FOV. We denote these sub-images as $I_{g1}, I_{g2}, ..., I_{gn}$. The input image is register to ONE of these sub-images, say I_{gr}. We then synthesize the output for each of these sub-image one by one. For example, for the sub-image I_{g1}, we find representative translations by matching patches in I_{g1} to I_{gr}. We then solve the hierarchical graph optimization to generate I_1 from the input image. Finally, we combine all these intermediate results to a full panorama by photomontage, which involves another graph cut optimization. This "divide and conquer" strategy generates good results in our experiments. One such example is provided in Figure 3-11. The success of this divide and conquer approach also demonstrates the robustness of our method, because it requires that all the sub-images be synthesized correctly and consistently with each other. Figure 3-18 shows more panorama results for outdoor, indoor, and street scenes. The first column is the input image. On the right hand side of each input image are the guide image



a. Input image aligned with the guide image

b. Zoomed region in (c) and (d)

c. Synthesis with only color feature

d. Synthesis with all features

Figure 3-17: Synthesis with different patch feature. The result obtained with HOG feature is often better than that from color features alone.

(upper image) and the synthesized result (lower image). In all the panorama synthesis experiments, the 360° of panorama is divided into 12 sub-images with uniformly sampled viewing direction from $0^\circ \sim 360^\circ$. The FOV of each sub-image is set to 65.5°. This ensures sufficient overlapping between two nearby sub-images. The FOV of the input images are around $40^\circ \sim 65.5^\circ$ degrees.

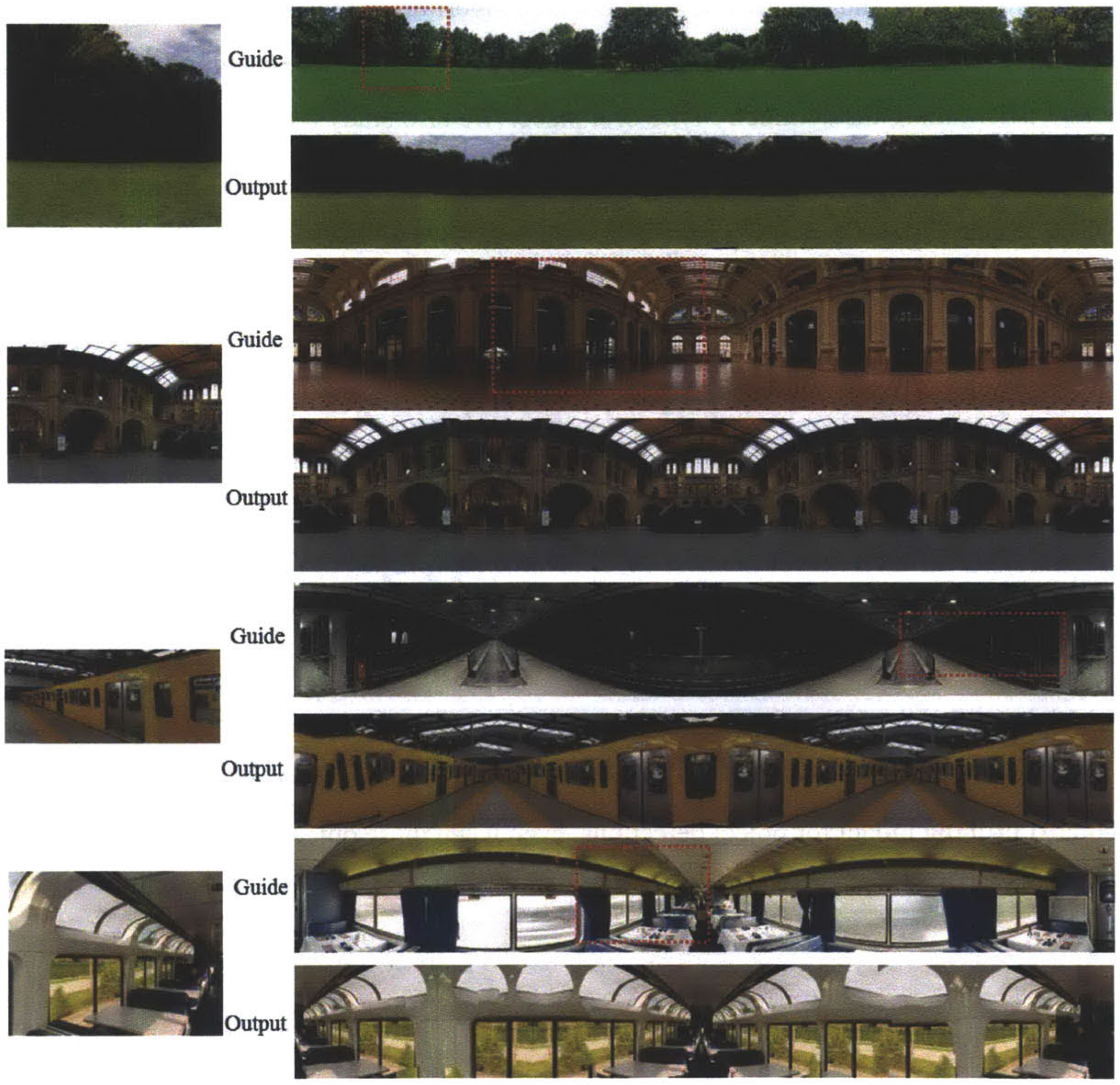


Figure 3-18: Panorama synthesis result. The left column is the input image. On the right are the guide image and the synthesized image.

3.9 Conclusion

We introduce a new problem in scene recognition: inferring not only the semantic category of a place, but also the observer's viewpoint within that place. We construct a new panorama dataset, and design a new incremental algorithm for automatic panorama alignment. We introduce the concept of scene symmetry and demonstrate a method for automatically determining the symmetry type of a place. We also study the view biases exhibited by people taking photos of places, and show how different viewpoints of a single place category may be classified into different semantic scene categories. All data and source code will be publicly available to encourage further research in this problem.

Since this is a first attempt at a new problem, we are simplifying the question by considering only viewpoint and ignoring the observer's position within a place. However, the principle ideas of this algorithm can be extended to address observer location and intra-category variations for alignment, by modifying the algorithm from one alignment per category to a mixture model.

By placing individual views within the context of a larger 3d place, our algorithm is able to represent elements of a scene which are beyond the boundaries of the available view, which has many useful applications. For example, if the objects in the training panoramas are annotated, the extrapolated layout generated by our algorithm can serve as semantic context priming model for object detections, even when those objects are outside the available field of view.

Chapter 4

Understanding 3D Scene Structure

Although an image is a 2D array, we live in a 3D world where scenes have volume, affordances, and are spatially arranged with objects occluding each other. The ability to reason about these 3D properties would be useful for tasks such as navigation and object manipulation. The dominant research focus for 3D reconstruction is in obtaining more accurate depth maps or 3D point clouds. However, even with a depth map, we are still unable to manipulate an object because there is no high-level representation of the 3D world. Essentially, 3D reconstruction is not just a low-level task. Obtaining a depth map to capture a distance at each pixel is analogous to inventing a digital camera to capture the color value at each pixel. The gap between low-level depth measurements and high-level shape understanding is just as large as the gap between pixel colors and high-level semantic perception. Moving forward, we need a higher-level intelligence for 3D reconstruction.

For many objects, their 3D shape can be entirely explained by a simple geometric primitive, such as a cuboid. In Section 4.1, we propose a 3D object detector that recovers a parameterization of the object's 3D shape, along with the camera pose. Beyond a single cuboid, we also propose a context model to reason multiple cuboid interactions in 3D (Section 4.2). To obtain the ground truth for training and evaluation, we design a novel web-based annotation tool to label 3D cuboids (Section 4.3).

The materials presented in this chaper are based on [194, 89, 195] in collaboration with Antonio Torralba, Bryan C. Russell, and Hao Jiang.

4.1 From 2D to 3D

Extracting a 3D representation from a single-view image depicting a 3D object has been a long-standing goal of computer vision [138]. Traditional approaches have sought to recover 3D properties, such as creases, folds, and occlusions of surfaces, from a line representation extracted from the image [119]. Among these are works that have characterized and detected *geometric primitives*, such as quadrics (or "geons") and surfaces of revolution, which have been thought to form the components for many different object types [17]. While these approaches have achieved notable early successes, they could not be scaled-up due to their dependence on reliable contour extraction from natural images.

In this work we focus on the task of detecting *rectangular cuboids*, which are a basic geometric primitive type and occur often in 3D scenes (e.g. indoor and outdoor man-made scenes [189, 190, 191]). Moreover, we wish to recover the shape parameters of the detected cuboids. The detection and recovery of shape parameters yield at least a partial geometric description of the depicted scene, which allows a system to reason about the affordances of a scene in an object-agnostic fashion [62, 83]. This is especially important when the category of the object is ambiguous or unknown.

There have been several recent efforts that revisit this problem [62, 69, 70, 106, 130, 181, 195, 198, 201]. Although there are many technical differences amongst these works, the main pipeline of these approaches is similar. Most of them estimate the camera parameters using three orthogonal vanishing points with a Manhattan world assumption of a man-made scene. They detect line segments via Canny edges and recover surface orientations [78] to form 3D cuboid hypotheses using bottom-up grouping of line and region segments. Then, they score these hypotheses based on the image evidence for lines and surface orientations [78].

In this section we look to take a different approach for this problem. As shown in Figure 5-19, we aim to build a 3D cuboid detector to detect individual boxy volumetric structures. We build a discriminative parts-based detector that models the appearance of the corners and internal edges of cuboids while enforcing spatial consistency of the corners and edges to a 3D cuboid model. Our model is trained in a similar fashion to recent work

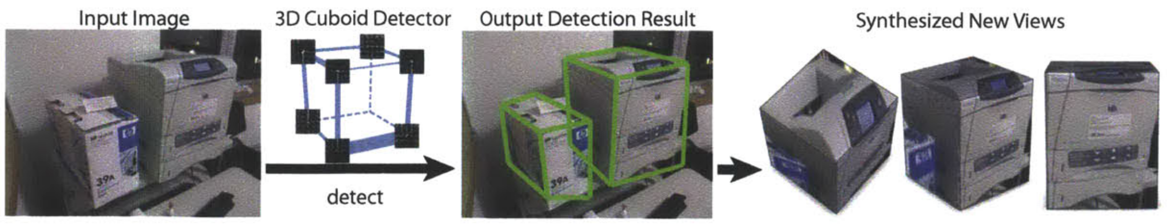


Figure 4-1: Problem summary. Given a single-view input image, our goal is to detect the 2D corner locations of the cuboids depicted in the image. With the output part locations we can subsequently recover information about the camera and 3D shape via camera resectioning.

that detects articulated human body joints [197].

Our cuboid detector is trained across different 3D viewpoints and aspect ratios. This is in contrast to view-based approaches for object detection that train separate models for different viewpoints, e.g. [52]. Moreover, instead of relying on edge detection and grouping to form an initial hypothesis of a cuboid [62, 106, 195, 201], we use a 2D sliding window approach to exhaustively evaluate all possible detection windows. Also, our model does not rely on any preprocessing step, such as computing surface orientations [78]. Instead, we learn the parameters for our model using a structural SVM framework. This allows the detector to adapt to the training data to identify the relative importance of corners, edges and 3D shape constraints by learning the weights for these terms. We introduce an annotated database of images with geometric primitives labeled and validate our model by showing qualitative and quantitative results on our collected database. We also compare to baseline detectors that use 2D constraints alone on the tasks of geometric primitive detection and part localization. We show improved performance on the part localization task.

4.1.1 Model for 3D cuboid localization

We represent the appearance of cuboids by a set of parts located at the corners of the cuboid and a set of internal edges. We enforce spatial consistency among the corners and edges by explicitly reasoning about its 3D shape. Let I be the image and $p_i = (x_i, y_i)$ be the 2D image location of the ith corner on the cuboid. We define an undirected loopy graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ over the corners of the cuboid, with vertices $\mathscr{V}$ and edges $\mathscr{E}$ connecting the corners of the cuboid. We illustrate our loopy graph layout in Figure 4-2(a). We define a scoring function associated with the corner locations in the image:

$$S(I,p) = \sum_{i \in \mathcal{V}} w_i^H \cdot \text{HOG}(I,p_i) + \sum_{ij \in \mathcal{E}} w_{ij}^D \cdot \text{Displacement}^{2\text{D}}(p_i,p_j)$$
$$+ \sum_{ij \in \mathcal{E}} w_{ij}^E \cdot \text{Edge}(I,p_i,p_j) + w^S \cdot \text{Shape}^{3\text{D}}(p) \tag{4.1}$$

where $\text{HOG}(I,p_i)$ is a HOG descriptor [27] computed at image location p_i and

$$\text{Displacement}^{2\text{D}}(p_i,p_j) = -[(x_i - x_j)^2, x_i - x_j, (y_i - y_j)^2, y_i - y_j] \tag{4.2}$$

is a 2D corner displacement term that is used in other pictorial parts-based models [52, 197]. By reasoning about the 3D shape, our model handles different 3D viewpoints and aspect ratios, as illustrated in Figure 4-2. Notice that our model is linear in the weights w. Moreover, the model is flexible as it adapts to the training data by automatically learning weights that measure the relative importance of the appearance and spatial terms. We define the Edge and Shape$^{3\text{D}}$ terms as follows.

$\text{Edge}(I,p_i,p_j)$: The internal edge information on cuboids is informative and provides a salient feature for the locations of the corners. For this, we include a term that models the appearance of the internal edges, which is illustrated in Figure 4-3. For adjacent corners on the cuboid, we identify the edge between the two corners and calculate the image evidence to support the existence of such an edge. Given the corner locations p_i and p_j, we use Chamfer matching to align the straight line between the two corners to edges extracted from the image. We find image edges using Canny edge detection [23] and efficiently compute the distance of each pixel along the line segment to the nearest edge via the truncated distance transform. We use Bresenham's line algorithm [21] to efficiently find the 2D image locations on the line between the two points. The final edge term is the negative mean value of the Chamfer matching score for all pixels on the line. As there are usually 9 visible edges for a cuboid, we have 9 dimensions for the edge term.

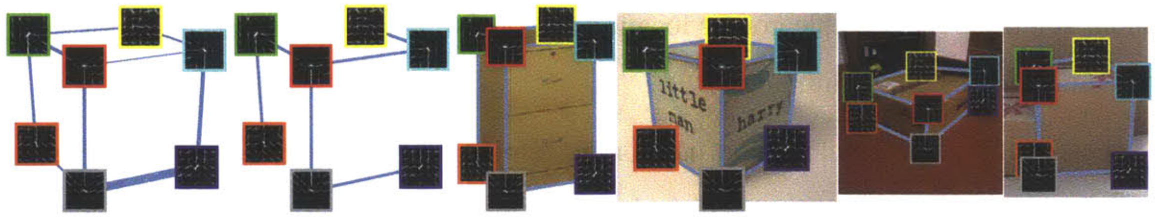

(a) Our Full Model. (b) 2D Tree Model. (c) Example Part Detections.

Figure 4-2: Model visualization. Corresponding model parts are colored consistently in the figure. In (a) and (b) the displayed corner locations are the average 2D locations across all viewpoints and aspect ratios in our database. In (a) the edge thickness corresponds to the learned weight for the edge term. We can see that the bottom edge is significantly thicker, which indicates that it is informative for detection, possibly due to shadows and contact with a supporting plane.

Shape$^{3D}(p)$: The 3D shape of a cuboid constrains the layout of the parts and edges in the image. We propose to define a shape term that measures how well the configuration of corner locations respect the 3D shape. In other words, given the 2D locations p of the corners, we define a term that tells us how likely this configuration of corner locations p can be interpreted as the reprojection of a valid cuboid in 3D. When combined with the weights w_S, we get an overall evaluation of the goodness of the 2D locations with respect to the 3D shape. Let $\mathbf{X}$ (written in homogeneous coordinates) be the 3D points on the unit cube centered at the world origin. Then, $\mathbf{X}$ transforms as $\mathbf{x} = \mathbf{PLX}$, where $\mathbf{L}$ is a matrix that transforms the shape of the unit cube and $\mathbf{P}$ is a 3×4 camera matrix. For each corner, we use the other six 2D corner locations to estimate the product $\mathbf{PL}$ using camera resectioning [64]. The estimated matrix is used to predict the corner location. We use the negative L2 distance to the predicted corner location as a feature for the corner in our model. As we model 7 corners on the cuboid, there are 7 dimensions in the feature vector. When combined with the learned weights w^S through dot-product, this represents a weighted reprojection error score.

Inference

Our goal is to find the 2D corner locations p over the HOG grid of I that maximizes the score given in Equation (4.1). Note that exact inference is hard due to the global shape term. Therefore, we propose a spanning tree approximation to the graph to obtain multiple initial solutions for possible corner locations. Then we adjust the corner locations using randomized simple hill climbing.

For the initialization, it is important for the computation to be efficient since we need to evaluate all possible detection windows in the image. Therefore, for simplicity and speed, we use a spanning tree $\mathscr{T}$ to approximate the full graph $\mathscr{G}$, as shown in Figure 4-2(b). In addition to the HOG feature as a unary term, we use a popular pairwise spring term along the edges of the tree to establish weak spatial constraints on the corners. We use the following scoring function for the initialization:

$$S_{\mathscr{T}}(I,p) = \sum_{i \in \mathscr{V}} w_i^H \cdot \mathrm{HOG}(I,p_i) + \sum_{ij \in \mathscr{T}} w_{ij}^D \cdot \mathrm{Displacement}^{\mathrm{2D}}(p_i,p_j) \tag{4.3}$$

Note that the model used for obtaining initial solutions is similar to [52, 197], which is only able to handle a fixed viewpoint and 2D aspect ratio. Nonetheless, we use it since it meets our speed requirement via dynamic programming and the distance transform [50].

With the tree approximation, we pick the top 1000 possible configurations of corner locations from each image and optimize our scoring function by adjusting the corner locations using randomized simple hill climbing. Given the initial corner locations for a single configuration, we iteratively choose a random corner i with the goal of finding a new pixel location $\hat{p}_i$ that increases the scoring function given in Equation (4.1) while holding the other corner locations fixed. We compute the scores at neighboring pixel locations to the current setting p_i. We also consider the pixel location that the 3D rigid model predicts when estimated from the other corner locations. We randomly choose one of the locations and update p_i if it yields a higher score. Otherwise, we choose another random corner and location. The algorithm terminates when no corner can reach a location that improves the score, which indicates that we have reached a local maxima.

During detection, since the edge and 3D shape terms are non-positive and the weights are constrained to be positive, this allows us to upper-bound the scoring function and quickly reject candidate locations without evaluating the entire function. Also, since only one corner can change locations at each iteration, we can reuse the computed scoring function from previous iterations during hill climbing. Finally, we perform non-maximal sup-

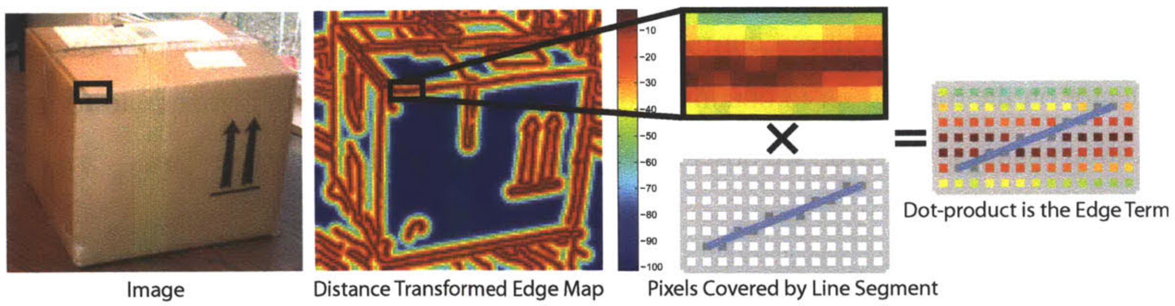


Image
Distance Transformed Edge Map
Pixels Covered by Line Segment
Dot-product is the Edge Term
-10
-20
-30
-40
-50
-60
-70
-80
-90
-100

Figure 4-3: Illustration of the edge term in our model. Given line endpoints, we compute a Chamfer matching score for pixels that lie on the line using the response from a Canny edge detector.

pression among the parts and then perform non-maximal suppression over the entire object to get the final detection result.

Learning

For learning, we first note that our scoring function in Equation (4.1) is linear in the weights w. This allows us to use existing structured prediction procedures for learning. To learn the weights, we adapt the structural SVM framework of [90]. Given positive training images with the 2D corner locations labeled $\{I_n, p_n\}$ and negative training images $\{I_n\}$, we wish to learn weights and bias term $\beta = (w^H, w^D, w^E, w^S, b)$ that minimizes the following structured prediction objective function:

$$\begin{aligned} \min_{\beta,\xi\geq 0} \quad & \frac{1}{2}\beta\cdot\beta + C\sum_n \xi_n \\ \forall n \in \text{pos} \quad & \beta\cdot\Phi(I_n, p_n) \geq 1 - \xi_n \\ \forall n \in \text{neg}, \forall p \in P \quad & \beta\cdot\Phi(I_n, p) \leq -1 + \xi_n \end{aligned} \tag{4.4}$$

where all appearance and spatial feature vectors are concatenated into the vector $\Phi(I_n, p)$ and P is the set of all possible part locations. During training we constrain the weights $w^D, w^E, w^S \geq 0.0001$. We tried mining negatives from the wrong corner locations in the positive examples but found that it did not improve the performance. We also tried latent positive mining and empirically observed that it slightly helps. Since the latent positive mining helped, we also tried an offset compensation as post-processing to obtain the offset of corner locations introduced during latent positive mining. For this, we ran the trained detector on the training set to obtain the offsets and used the mean to compensate for the location changes. However, we observed empirically that it did not help performance.

Discussion

Sliding window object detectors typically use a root filter that covers the entire object [27] or a combination of root filter and part filters [52]. The use of a root filter is sufficient to capture the appearance for many object categories since they have canonical 3D viewpoints and aspect ratios. However, cuboids in general span a large number of object categories and do not have a consistent 3D viewpoint or aspect ratio. The diversity of 3D viewpoints and aspect ratios causes dramatic changes in the root filter response. However, we have observed that the responses for the part filters are less affected.

Moreover, we argue that a purely view-based approach that trains separate models for the different viewpoints and aspect ratios may not capture well this diversity. For example, such a strategy would require dividing the training data to train each model. In contrast, we train our model for all 3D viewpoints and aspect ratios. We illustrate this in Figure 4-2, where detected parts are colored consistently in the figure. As our model handles different viewpoints and aspect ratios, we are able to make use of the entire database during training.

Due to the diversity of cuboid appearance, our model is designed to capture the most salient features, namely the corners and edges. While the corners and edges may be occluded (e.g. by self-occlusion, other objects in front, or cropping), for now we do not handle these cases explicitly in our model. Furthermore, we do not make use of other appearance cues, such as the appearance within the cuboid faces, since they have a larger variation across the object categories (e.g. dice and fire alarm trigger) and may not generalize as well. We also take into account the tractability of our model as adding additional appearance cues will increase the complexity of our model and the detector needs to be evaluated over a large number of possible sliding windows in an image.

Compared with recent approaches that detect cuboids by reasoning about the shape of the entire scene [62, 69, 70, 106, 130, 201], one of the key differences is that we detect cuboids directly without consideration of the global scene geometry. These prior ap-

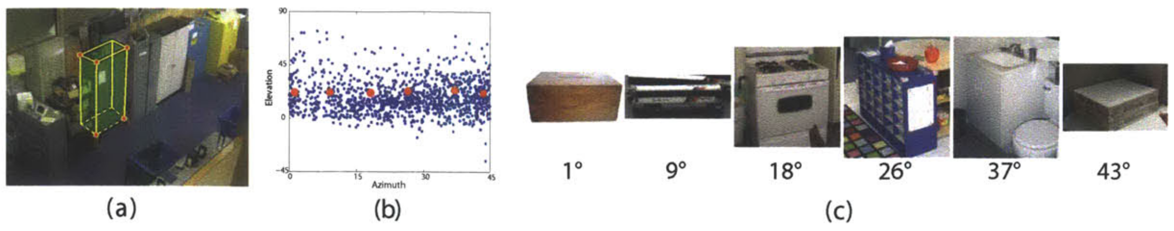


Figure 4-4: Illustration of the labeling tool and 3D viewpoint statistics. (a) A cuboid being labeled through the tool (See Section 4.3 for details). (b) Scatter plot of 3D azimuth and elevation angles for annotated cuboids with zenith angle close zero. We perform an image left/right swap to limit the rotation range. (c) Crops of cuboids at different azimuth angles for a fixed elevation, with the shown examples marked as red points in the scatter plot of (b).

proaches rely heavily on the assumption that the camera is located *inside* a cuboid-like room and held at human height, with the parameters of the room cuboid inferred through vanishing points based on a Manhattan world assumption. Therefore, they cannot handle outdoor scenes or close-up snapshots of an object (e.g. the boxes on a shelf in row 1, column 3 of Figure 4-6). As our detector is agnostic to the scene geometry, we are able to detect cuboids even when these assumptions are violated.

While previous approaches reason over rigid cuboids, our model is flexible in that it can adapt to deformations of the 3D shape. We observe that not all cuboid-like objects are perfect cuboids in practice. Deformations of the shape may arise due to the design of the object (e.g. the printer in Figure 5-19), natural deformation or degradation of the object (e.g. a cardboard box), or a global transformation of the image (e.g. camera radial distortion). We argue that modeling the deformations is important in practice since a violation of the rigid constraints may make a 3D reconstruction-based approach numerically unstable. In our approach, we model the 3D deformation and allow the structural SVM to learn based on the training data how to weight the importance of the 3D shape term. Moreover, a rigid shape requires a perfect 3D reconstruction and it is usually done with non-linear optimization [106], which is expensive to compute and becomes impractical in an exhaustive sliding-window search in order to maintain a high recall rate. With our approach, if a rigid cuboid is needed, we can recover the 3D shape parameters via camera resectioning, as shown in Figure 4-9.

4.1.2 Database of 3D cuboids

To develop and evaluate any models for 3D cuboid detection in real-world environments, it is necessary to have a large database of images depicting everyday scenes with 3D cuboids labeled. In this work, we seek to build a database by manually labeling point correspondences between images and 3D cuboids. We have built a labeling tool that allows a user to select and drag key points on a projected 3D cuboid model to its corresponding location in the image (See Section 4.3 for details). For the database, we have harvested images from four sources: (i) a subset of the SUN database [184], which contains images depicting a large variety of different scene categories, (ii) ImageNet synsets [32] with objects having one or more 3D cuboids depicted, (iii) images returned from an Internet search using keywords for objects that are wholly or partially described by 3D cuboids, and (iv) a set of images that we manually collected from our personal photographs. Given the corner correspondences, the parameters for the 3D cuboids and camera are estimated. The cuboid and camera parameters are estimated up to a similarity transformation via camera resectioning using Levenberg-Marquardt optimization [64].

For our database, we have 785 images with 1269 cuboids annotated. We have also collected a negative set containing 2746 images that do contain any cuboid like objects. We perform an image left/right swap to limit the rotation range. As a result, the min/max azimuth, elevation, and zenith angles are 0/45, -90/90, -180/180 degrees respectively. In Figure 4-4(b) we show a scatter plot of the azimuth and elevation angles for all of the labeled cuboids with zenith angle close to zero. Notice that the cuboids cover a large range of azimuth angles for elevation angles between 0 (frontal view) and 45 degrees. We also show a number of cropped examples for a fixed elevation angle in Figure 4-4(c), with their corresponding azimuth angles indicated by the red points in the scatter plot. Figure 4-8(c) shows the distribution of objects from the SUN database [184] that overlap with our cuboids (there are 326 objects total from 114 unique classes). Compared with [70], our database covers a larger set of object and scene categories, with images focusing on both objects and scenes (all images in [70] are indoor scene images). Moreover, we annotate objects closely resembling a 3D cuboid (in [70] there are many non-cuboids that are annotated

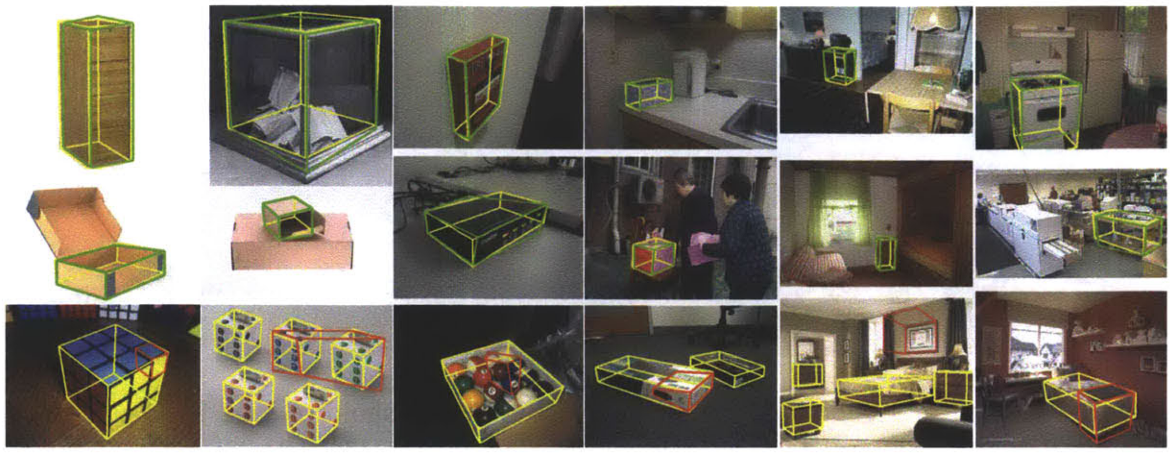


Figure 4-5: Single top 3D cuboid detection in each image. Yellow: ground truth, green: correct detection, red: false alarm. Bottom row - false positives. The false positives tend to occur when a part fires on a "cuboid-like" corner region (e.g. row 3, column 5) or finds a smaller cuboid (e.g. the Rubik's cube depicted in row 3, column 1).

with a bounding cuboid) and overall our cuboids are more accurately labeled.

4.1.3 Evaluation

In this section we show qualitative results of our model on the 3D cuboids database and report quantitative results on two tasks: (i) 3D cuboid detection and (ii) corner localization accuracy. For training and testing, we randomly split equally the positive and negative images. As discussed in Section 4.1.2, there is rotational symmetry in the 3D cuboids. During training, we allow the image to mirror left-right and orient the 3D cuboid to minimize the variation in rotational angle. During testing, we run the detector on left-right mirrors of the image and select the output at each location with the highest detector response. For the parts we extract HOG features [27] in a window centered at each corner with scale of 10% of the object bounding box size. Figure 4-5 shows the single top cuboid detection in each image and Figure 4-6 shows all of the most confident detections in the image. Notice that our model is able to handle partial occlusions (e.g. row 1, column 4 of Figure 4-6), small objects, and a range of 3D viewpoints, aspect ratios, and object classes. Moreover, the depicted scenes have varying amount of clutter. We note that our model fails when a corner fires on a "cuboid-like" corner region (e.g. row 3, column 5 of Figure 4-5).

We compare the various components of our model against two baseline approaches.

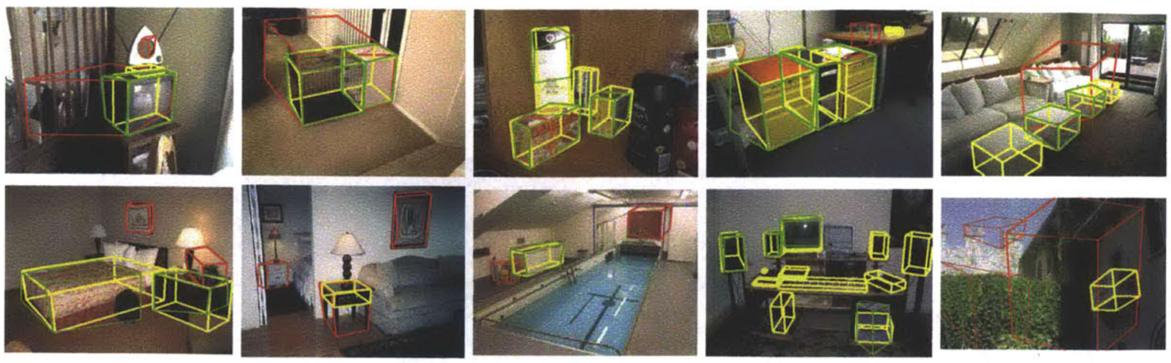


Figure 4-6: All 3D cuboid detections above a fixed threshold in each image. Notice that our model is able to detect the presence of multiple cuboids in an image (e.g. row 1, columns 2-5) and handles partial occlusions (e.g. row 1, column 4), small objects, and a range of 3D viewpoints, aspect ratios, and object classes. Moreover, the depicted scenes have varying amount of clutter. Yellow - ground truth. Green - correct prediction. Red - false positive. Line thickness corresponds to detector confidence.

The first baseline is a root HOG template [27] trained over the appearance within a bounding box covering the entire object. A single model using the root HOG template is trained for all viewpoints and aspect ratios. During detection, output corner locations corresponding to the average training corner locations relative to the bounding boxes are returned. The second baseline is the 2D tree-based approximation of Equation (4.3), which corresponds to existing 2D parts models used in object detection and articulated pose estimation [52, 197]. Figure 4-7 shows a qualitative comparison of our model against the 2D tree-based model. Notice that our model localizes well and often provides a tighter fit to the image data than the baseline model.

We evaluate geometric primitive detection accuracy using the bounding box overlap criteria in the Pascal VOC [45]. We report precision recall in Figure 4-8(a). We have observed that all of the corner-based models achieve almost identical detection accuracy across all recall levels, and out-perform the root HOG template detector [27]. This is expected as we initialize our full model with the output of the 2D tree-based model and it generally does not drift too far from this initialization. This in effect does not allow us to detect additional cuboids but allows for better part localization.

In addition to detection accuracy, we also measure corner localization accuracy for correctly detected examples for a given model. A corner is deemed correct if its predicted

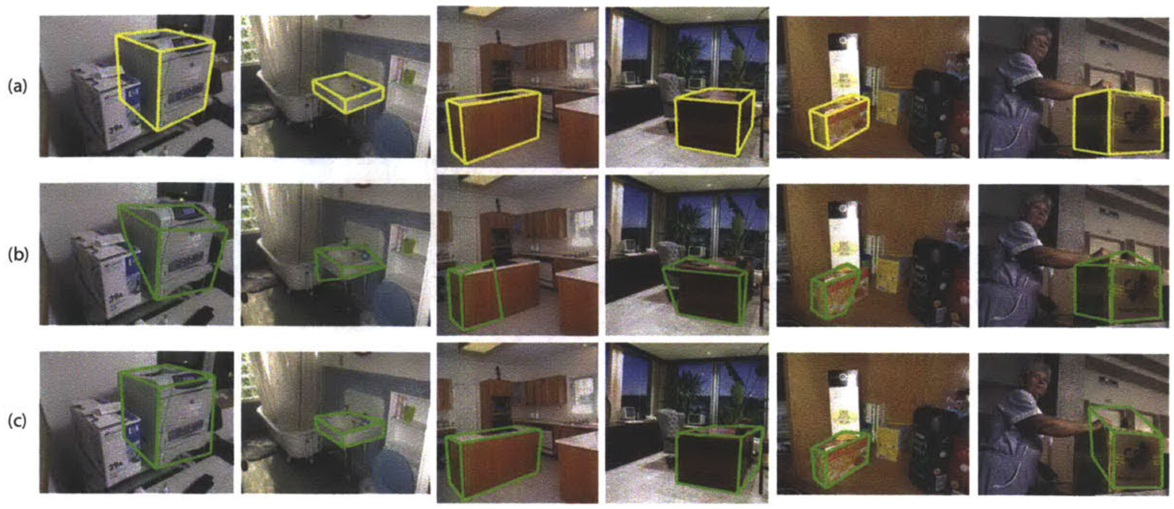
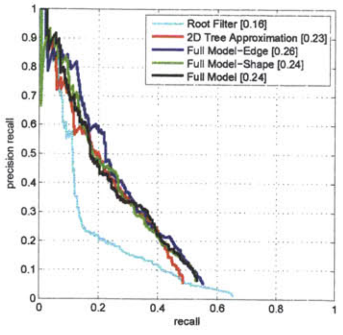


Figure 4-7: Corner localization comparison for detected geometric primitives. (a) Input image and ground truth annotation. (b) 2D tree-based initialization. (c) Our full model. Notice that our model is able to better localize cuboid corners over the baseline 2D tree-based model, which corresponds to 2D parts-based models used in object detection and articulated pose estimation [52, 197]. The last column shows a failure case where a part fires on a "cuboid-like" corner region in the image.

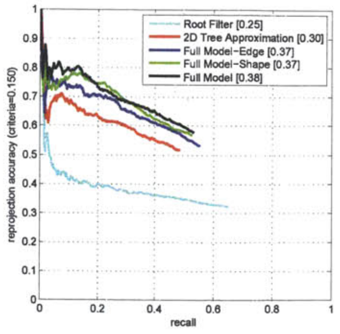
image location is within t pixels of the ground truth corner location. We set t to be 15% of the square root of the area of the ground truth bounding box for the object. The reported trends in the corner localization performance hold for nearby values of t. In Figure 4-8 we plot corner localization accuracy as a function of recall and compare our model against the two baselines. Moreover, we report performance when either the edge term or the 3D shape term is omitted from our model. Notice that our full model out-performs the other baselines. Also, the additional edge and 3D shape terms provide a gain in performance over using the appearance and 2D spatial terms alone. The edge term provides a slightly larger gain in performance over the 3D shape term, but when integrated together consistently provides the best performance on our database.

4.1.4 Conclusion

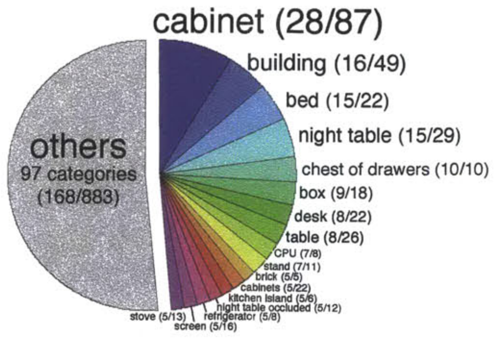
We have introduced a novel model that detects 3D cuboids and localizes their corners in single-view images. Our 3D cuboid detector makes use of both corner and edge information. Moreover, we have constructed a dataset with ground truth cuboid annotations. Our



(a) Cuboid detection



(b) Corner localization



(c) Object distribution

Figure 4-8: Cuboid detection (precision vs. recall) and corner localization accuracy (accuracy vs. recall). The area under the curve is reported in the plot legends. Notice that all of the corner-based models achieve almost identical detection accuracy across all recall levels and out-perform the root HOG template detector [27]. For the task of corner localization, our full model out-performs the two baseline detectors or when either the Edge or Shape^{3D} terms are omitted from our model. (c) Distribution of objects from the SUN database [184] that overlap with our cuboids. There are 326 objects total from 114 unique classes. The first number within the parentheses indicates the number of instances in each object category that overlaps with a labeled cuboid, while the second number is the total number of labeled instances for the object category within our dataset.

detector handles different 3D viewpoints and aspect ratios and, in contrast to recent approaches for 3D cuboid detection, does not make any assumptions about the scene geometry and allows for deformation of the 3D cuboid shape. As HOG is not invariant to viewpoint, we believe that part mixtures would allow the model to be invariant to viewpoint. We believe our approach extends to other shapes, such as cylinders and pyramids. Our work raises a number of (long-standing) issues that would be interesting to address. For instance, which objects can be described by one or more geometric primitives and how to best represent the compositionality of objects in general? By detecting geometric primitives, what applications and systems can be developed to exploit this? Our dataset and source code is publicly available at the project webpage: `http://SUNprimitive.csail.mit.edu`.

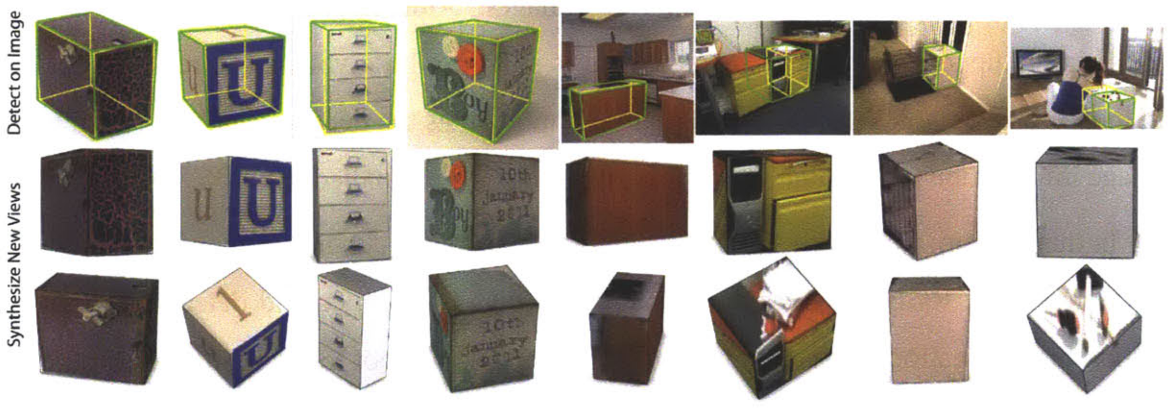


Figure 4-9: Detected cuboids and subsequent synthesized new views via camera resectioning.

4.2 From individual to context

Finding three-dimensional structures and shapes from images is a key task in computer vision. Nowadays, we can obtain reliable depth map using low cost RGBD cameras from the digital consumer market, *e.g.* Microsoft Kinect, Asus Xtion and Primesense. Just like digital cameras that capture raw RGB data, these devices capture raw depth maps along with RGB color images. RGBD images provide a pointwise representation of a 3D space. We would like to extract structures from such data.

Recently, there are a few heroic efforts in extracting structures in RGBD images, *e.g.* [153, 97]. However, most of these approaches group pixels into surface segments, *i.e.* the counterpart of image segmentation for RGB images. Although some noteworthy studies [154] infer support relations in scenes, there is still very little volumetric reasoning used in the 3D space, which should be even more important as the depth is available, than pure image information with which volumetric reasoning is well studied [61, 106, 194, 201, 70].

In this section, we design an efficient algorithm to match cuboid structures in an indoor scene using the RGBD images, as illustrated in Fig. 4-10. A cuboid detector has many important applications. It is a key technique to enable a robot to manipulate box objects [171]. Cuboids also often appear in man-made structures [194]. A cuboid detector thus facilitates finding these structures. Detecting cuboids from RGBD images is challenging due to heavy object occlusion, missing data and strong clutter. We propose an efficient and reliable linear method to make a first step towards solving this problem.

Even though matching planes, spheres, cylinders and cones in point clouds has been intensively studied [149], there have been few methods that are able to match multiple cuboids simultaneously in 3D data. RANSAC has been combined with extra constraints to reconstruct geometrical primitives on industry parts from relatively clean range data [110]; this method assumes perfect geometric primitives. To recover cuboid-like objects in cluttered scenes, we need a different method.

Local approaches have been proposed for fitting cuboids to point clouds. In [107], shapes are modeled as superquadratics and detected in clutter-free range data. A gradient descent method has been proposed in [171] to find multiple cuboids. Due to high complex-

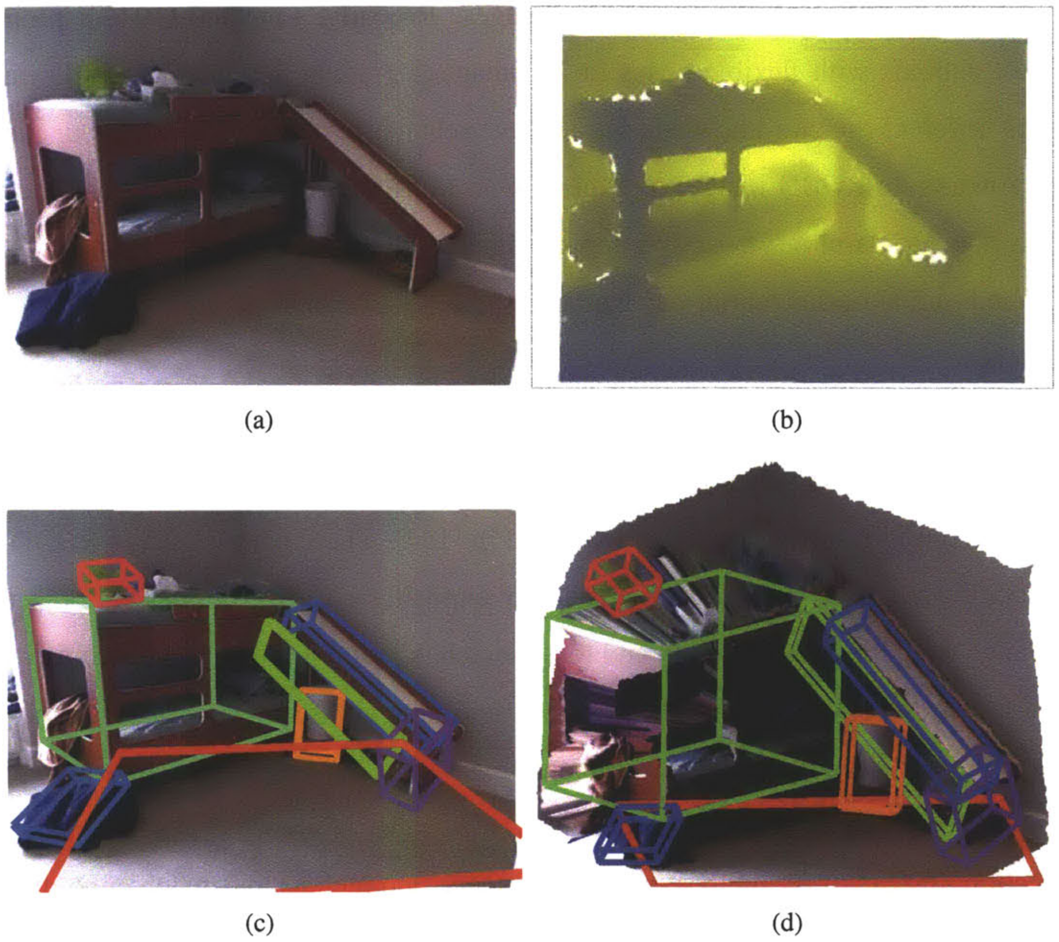


Figure 4-10: Given a color image and depth map we match cuboid-shaped objects in the scene. (a) and (b): The color image and aligned depth map from Kinect. (c): The cuboids detected by the proposed method and projected onto the color image. (d): The cuboids in the scene viewed from another perspective. These cuboids reveal important structures of the scene.

ity, this method has been used to find cuboids in simple scenes with clutter removed. In contrast to these local methods, our proposed method is able to work on cluttered scenes, does not need initialization and guarantees globally optimal result. Our method is also much more efficient.

Cuboid detection has been intensively studied with 2D images as the input. In [106], a method is proposed to reliably extract cuboids in 2D images of indoor scenes. This method assumes that all the cuboids are aligned to three dominant orientations. In [61], spatial reasoning is used to understand outdoor scenes in 2D images; object interactions are modeled with a small set of 3D configurations. Sampling method has been proposed in [30] to extract 3D layout in 2D indoor images. Recently, a method [194] is proposed to detect cuboids with flexible poses in 2D images. Finding 3D cuboids in 2D images requires different domain knowledge to achieve reliable results. In contrast, our method directly works on RGBD images and there is no restriction on the cuboid configuration: cuboids may have arbitrary pose and they can interact in complex ways. By using a branch and bound global optimization, our method is able to give more reliable results than 2D approaches.

The proposed method is also related to 3D point cloud segmentation. In [97] and [153], points in color point clouds are classified into a small number of categories. Supporting relations between patches are further extracted in [154]. These point cloud or RGBD data segmentation and classification methods do not explicitly extract "volumes" of objects. In [34], range data are pre-segmented and local search is proposed to fit shape primitives. In [18], a greedy scheme is proposed for spatial reasoning and segmenting a 3D scene from stereo into object proposals enclosed in bounding boxes. This method is currently applied to simple scenes and the result is dependent on the quality of object level segmentation. In this section, instead of trying to segment a 3D scene into regions, we match cuboids to the scene. Our method constructs reliable cuboid candidates by using pairs of planar patches and globally optimizes the cuboid configuration in a novel linear framework.

Finding cuboids in cluttered RGBD images is still unsolved. No previous methods are able to globally optimize the cuboid configuration when there is no restriction on the poses and interactions among objects. In this section, we propose a linear method that works on generic scenes. The proposed method first partitions the 3D point cloud into groups of piecewise linear patches using the graph method [53]. These patches are then used to generate a set of cuboid candidates, each of which has a cost. We globally optimize the selection of the cuboids so that they have small total cost and satisfy the global constraints. The optimal cuboid configuration has small intersection, and we prefer a large coverage of the cuboids. At the same time, we make sure the cuboids satisfy the occlusion conditions. The optimization is formulated as a mixed integer linear program and efficiently solved by a branch and bound method.

Our contribution is a novel linear approach that efficiently optimizes multiple cuboid matching in RGBD images. The proposed method works on cluttered scenes in unconstrained settings. Our experiments on thousands of images in the NYU Kinect dataset [154] and other images show that the proposed method is efficient and reliable.

4.2.1 Overview

We optimize the matching of multiple cuboids in a RGBD image from Kinect. Our goal is to find a set of cuboids that match the RGBD image and at the same time satisfy the spatial interaction constraint. We construct a set of cuboid candidates and select the optimal subset. The cuboid configuration is denoted by $\mathbf{x}$. We formulate cuboid matching into the following optimization problem,

$$\min_{\mathbf{x}} \{U(\mathbf{x}) + \lambda P(\mathbf{x}) + \mu N(\mathbf{x}) - \gamma A(\mathbf{x}) + \xi O(\mathbf{x})\} \tag{4.5}$$
$$\text{s.t. Cuboid configuration } \mathbf{x} \text{ satisfies global constraints.}$$

Here $U(\mathbf{x})$ is the unary term that quantifies the local matching costs of the cuboids, $P(\mathbf{x})$ is a pairwise term that quantifies the intersection between pairs of cuboids, $N(\mathbf{x})$ is the number of matched cuboids in the scene, $A(\mathbf{x})$ quantifies the covered area of the projected cuboids on the image plane, and O penalizes the occlusions among the cuboids. λ, μ, γ and ξ control the weight among different terms. In this section, $\mu = 0.1$, $\lambda = \xi = 0.02$ and $\gamma = 1$. By minimizing the objective, we prefer to find the multiple cuboid matching that has low local matching cost, small object intersection and occlusion, and covers a large area in the image with a small number of cuboids. Besides the soft constraints specified by the objective function, we further enforce that the optimal cuboid configuration $\mathbf{x}$ satisfies hard constraints on cuboid intersection and occlusion. This optimization is a combinatorial search problem. In the following, we propose an efficient linear solution.

4.2.2 Cuboid candidates

We first construct a set of cuboid candidates using pairs of superpixels in the RGBD image. Finding high quality cuboid candidates is critical; we propose a new method as follows.

Partition 3D points into groups: We first use the graph method in [53] to find superpixels on the RGBD image. With both color and surface normal images, we partition the depth map into roughly piecewise planar patches. As shown in row one of Fig. 5-25, we also use the superpixels from the normal image itself; this helps find textured planar patches.

Constructing cuboids: We use each pair of neighboring 3D surface patches to construct a cuboid candidate. We define that two surface patches are neighbors if their corresponding superpixels in the color or normal image have a distance less than a small threshold, *e.g.* 20 pixels. The distance of two superpixels is defined as the shortest distance between their boundaries. To remove outliers, we fit a plane to each 3D patch by RANSAC. In the following process, we use only the inlier points.

We are ready to construct cuboid candidates from pairs of patches. We select one of the two neighboring patches and rotate the 3D points in them so that the normal vector of the chosen one is aligned with the z axis. We then rotate the 3D points again so that the projected normal vector of the second 3D patch on the xy plane is aligned with y axis. We then find two rectangles parallel to the xy and xz plane to fit the points on the two 3D patches. The sizes of the two rectangles can be obtained by finding the truncated boundaries of the point histograms in each axis direction. For the first plane the z coordinate is the mean z of the points in the 3D patch, and for the second plane its y is the mean y of the points.

The cuboid is the smallest one that encloses both of the rectangles as shown in Fig. 5-25 and Fig. 4-13 (a). Fig. 5-25 rows 2-4 illustrate some of the cuboids reconstructed from neighboring superpixels. We change the red and blue channels of the color image to show the neighboring superpixels. The projection of these cuboids in Fig. 5-25 row 2 shows that the 3D cuboid estimation is accurate. Each candidate cuboid is represented by the lower and upper bounds of x, y and z coordinates in the normalized pose and a matrix T that transforms the cuboid back to the original pose. We also keep the inverse of T as F. Such a representation facilitates the computation of cuboid space occupancy and intersection.

Local matching costs: The quality of the matching of a cuboid to the 3D data is determined by three factors.

The first factor is the coverage area of the points in the two contact cuboid faces. To

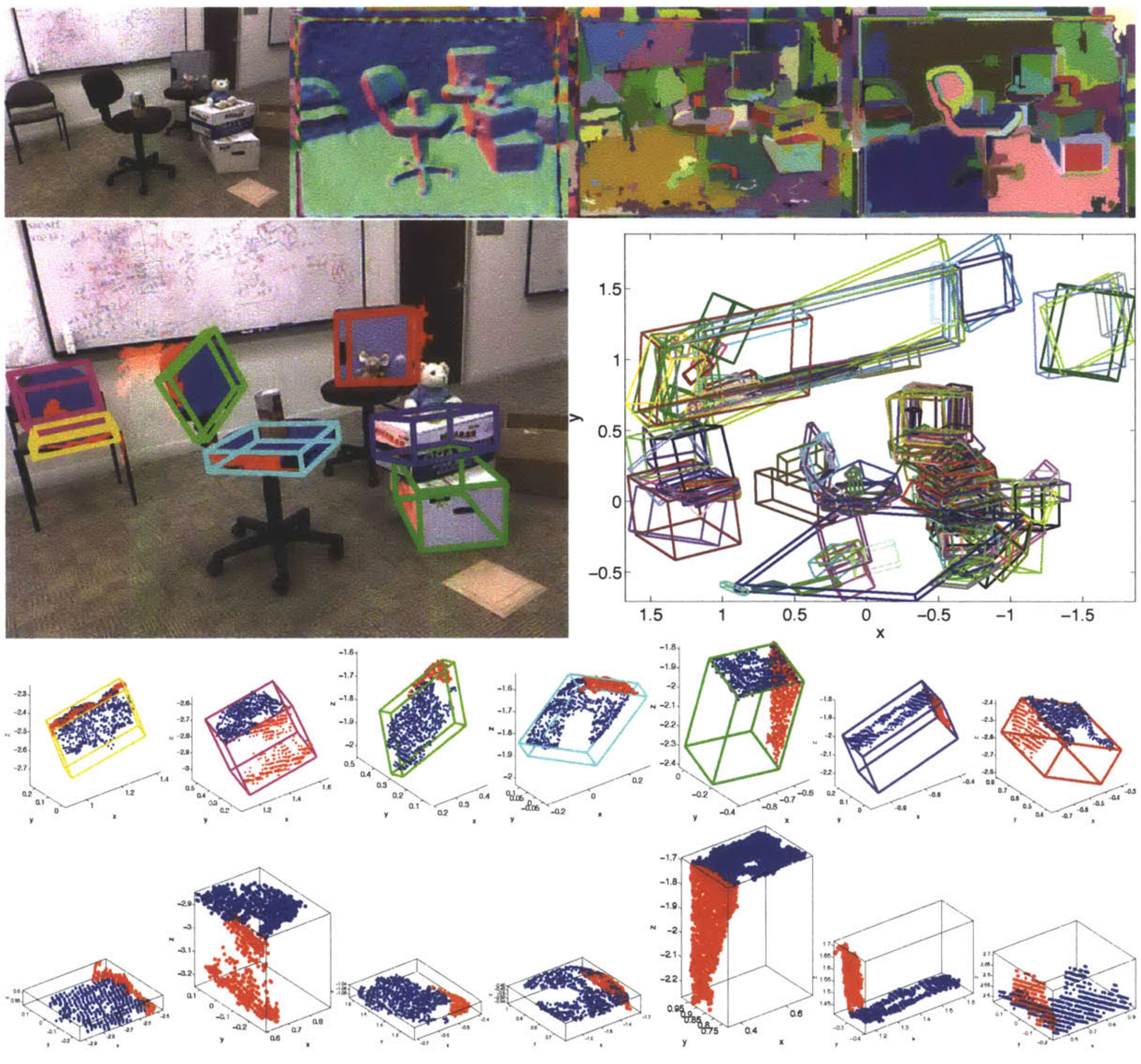
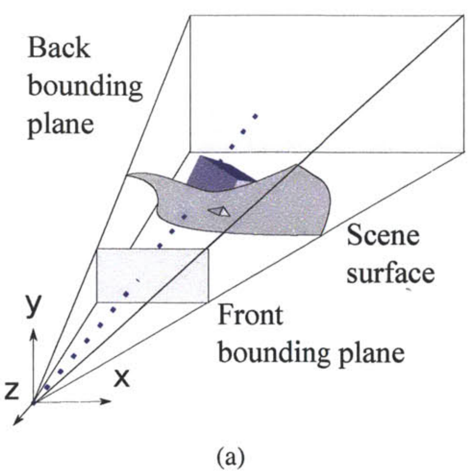


Figure 4-11: Row 1: From left to right are the color image, normal image with the three channels containing the x, y and z components, the superpixels by using both the color and normal images, and the superpixels by using the normal image only. The two superpixel maps are extracted with fixed parameters in this section. Row 2: Left shows the cuboids constructed using neighboring planar patches and projected on the image; right shows the top 200 cuboid candidates. Row 3: 3D view of the cuboids in the color image in row 2. The red and blue dots are the points from the neighboring surface patches. Row 4: The normalized poses of these cuboids with three edges parallel to the xyz axises.



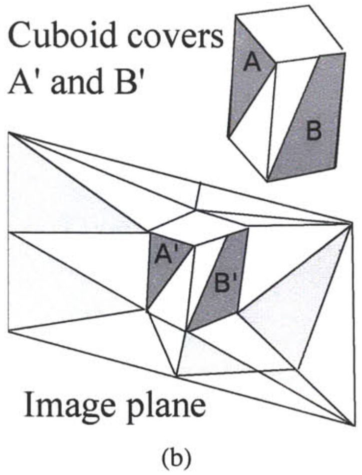


Figure 4-12: (a): We compute the solidness of a cuboid by discretizing the space in front of and behind the scene surface into voxels. (b): We encourage cuboids to cover large area of superpixels on an image.

simplify the computation, the coverage area of the surface points on a cuboid face is determined by the tightest bounding box as shown in Fig. 4-13 (a). We compute the ratio r of the bounding box area to the area of the corresponding cuboid face. The smaller one of the two ratios are used to quantify the cuboid local matching. A perfect cuboid matching has the ratio r of 1.

The second factor is the solidness. We require that cuboids should mostly be behind the 3D scene surface. To measure the solidness of cuboids, as shown in Fig. 4-12 (a), we quantize the space into voxels, whose centers are located on the rays starting from the origin point and passing through each image pixel. The space of interest is bounded in 1 to 10 meters from the camera. In the bounded space, 200 points are uniformly selected along each ray. The points behind the scene surface have z coordinates less than the surface z coordinates. To compute the solidness of cuboid i, we transform the points using the cuboid matrix F_i defined before to bring the cuboid to the normalized position. We do not need to transform all the points but only the points inside the bounding box of the cuboid in the original pose; other points are irrelevant. The solidness is approximated by n_s/n_a, where n_s is the number of solid space points in the cuboid and n_a is the number of all the transformed points falling in the cuboid. We keep only the cuboid candidates whose solidness is greater than 0.5.

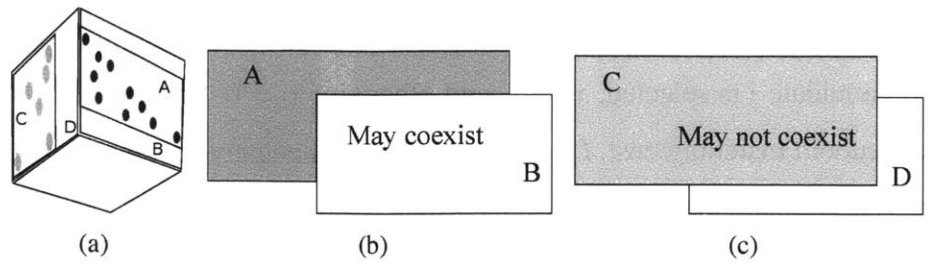


Figure 4-13: (a): The cuboid matching ratio r is $\min(A/B, C/D)$, where A, B, C, D are the areas of the rectangular regions. (b) and (c): The projection of cuboids and their depth order determine the occlusion. Cuboid A and B may coexist, but C and D cannot.

The third factor is the cuboid boundary matching cost. When we project each cuboid candidate to the target image, the candidate's projection silhouette should match the image edges. We find the average distance between the projection silhouette boundary and the edge pixels, which can be computed efficiently using the distance transform of the image edge map. We keep only the cuboid candidates whose silhouettes to edge average distance is less than 10 pixels.

We choose the top M cuboids ranked by the surface matching ratio r with the solidness and average boundary error in specific ranges, *e.g.*, solidness greater than 0.5 and boundary error less than 10 pixels in this section. The costs of cuboids are $c_i = 1 - r_i$, $i = 1..M$. In this section, we keep at most top 200 cuboid candidates.

4.2.3 Formulation

Local matching is not enough to determine the optimal set of cuboids. Since their matching costs are nonnegative, we would obtain a trivial all-zero solution if we simply minimize the unary term. One method is to specify the number of objects to be included in the scene. However, in practice, the number of cuboids is usually unknown. Another method is to train an SVM cuboid classifier based on the features and the classification result would have positive and negative values. Our experiment shows that the cuboid classifier has quite low classification rate and the top 200 candidates are almost always classified into the same category. We need to incorporate more global constraints and estimate the number of the cuboid objects and their poses at the same time. We propose a linear formulation of the optimization in Eq. (4.5).

Unary term We define a binary variable x_i to indicate whether cuboid candidate i is selected. If candidate i is selected, $x_i = 1$, and otherwise $x_i = 0$. The unary term U is the overall local cuboid matching cost, $U = \sum_i c_i x_i$, where nonnegative coefficient c_i is the cost of choosing cuboid candidate i; c_i is defined in section 4.2.2. There is a guarantee that all the cuboid candidates are at least 50% solid and have projection silhouettes with the average distance of less than 10 pixels to the image edges. Directly minimizing U would give trivial all-zero results. We need extra constraints.

Volume exclusion Since each cuboid is solid, they tend to occupy non-overlapping space in the scene. However, completely prohibiting the cuboid intersection is too strong a condition due to the unavoidable errors in candidate pose estimation. We set a tolerance value t for the cuboid intersection and in this section $t = 0.1$, which means cuboids may have up to 10% intersection. Here the intersection ratio of two cuboids is defined as the ratio of the volume intersection to the volume of the smaller cuboid. If one cuboid contains the other, the ratio is 1. The intersection ratio from cuboid i to j is computed by projecting the regularly sampled points in cuboid i in the normalized pose back to the original pose using cuboid matrix T_i and then projecting to cuboid j's normalized pose by using matrix F_j and finally computing the ratio of the inside points to all these projected points. The intersection ratio between cuboid i and j is denoted as $e_{i,j}$, which is the larger one of the two possible intersection ratios.

The volume exclusion term has two parts, the first part is soft and the other is a hard constraint. If two cuboid candidates have intersection less than t, we have the soft term in the objective function, $P = \sum_{\{\{i,j\}:0<e_{i,j}<t\}} e_{i,j} x_i x_j$. When optimizing the objective function, we try to minimize the intersection between cuboids. We linearize the quadratic term $x_i x_j$ by introducing an auxiliary variable $z_{i,j}$ and letting $z_{i,j} \leq x_i$, $z_{i,j} \leq x_j$, $z_{i,j} \geq 0$, and $z_{i,j} \geq x_i + x_j - 1$. It can be verified that $z_{i,j}$ is indeed the product of x_i and x_j.

Apart from the soft term, we prohibit any two cuboids from having intersection ratio that is greater than or equals t. The hard constraint is thus $x_i + x_j \leq 1$ for each pair of cuboid candidates i and j that have volume intersection $e_{i,j} \geq t$. The hard constraint ensures that cuboid candidates whose intersection ratio is at least t do not coexist; the corresponding soft intersection penalty in the objective function is zero.

Surface coverage The volume exclusion constraint ensures the correct space occupancy of the selected cuboids. However, it still does not solve the trivial all-zero solution problem, *i.e.*, if we simply minimize the unary and pairwise terms no cuboids will be selected. To solve the problem, we introduce a surface coverage term to encourage the selected cuboids to cover large surface area. We define that a cuboid covers the surface patches by which we construct the cuboid. And, we say a cuboid covers a superpixel in an image, if it covers the corresponding 3D scene patch, as illustrated in Fig. 4-12 (b). We define a variable y_k, which is 1 if superpixel k is covered by a selected cuboid, and otherwise 0. The surface coverage term is $A = \sum_k a_k y_k$, where a_k is the area of superpixel k. As we minimize the objective function, a large coverage area is preferred.

Superpixel indicator variable y_k and the cuboid indicator variable are correlated by

$$y_k \leq \sum_{\text{cuboid } i \text{ covers superpixel } k} x_i, \;\; 0 \leq y_k \leq 1.$$

If all the cuboids that cover superpixel k are not selected, $y_k = 0$. If at least one cuboid that covers superpixel k is selected, $y_k = 1$ and the term A is maximized. Therefore, when the objective is optimized, A is the covered area of the selected cuboids in the image.

Since cuboids may come from color-normal superpixels or normal superpixels, the superpixels may overlap. We construct fine-grained superpixels, each of which is a sub-region of a superpixel from the two superpixel sets. Variables y_k correspond to these fine-grained superpixels. The surface coverage term thus encourages the selection of a set of cuboids that not only have low local matching cost but also cover a large region in an image.

Smaller number of cuboids We prefer to use a small number of cuboids to explain the scene. The number of cuboids, $N = \sum_i x_i$, is introduced into the objective function. With the surface coverage term, the small number heuristic indicates that we tend to choose few large cuboids instead of many small ones. The cuboid number term and the unary term are merged in the objective function in Eq. (4.6).

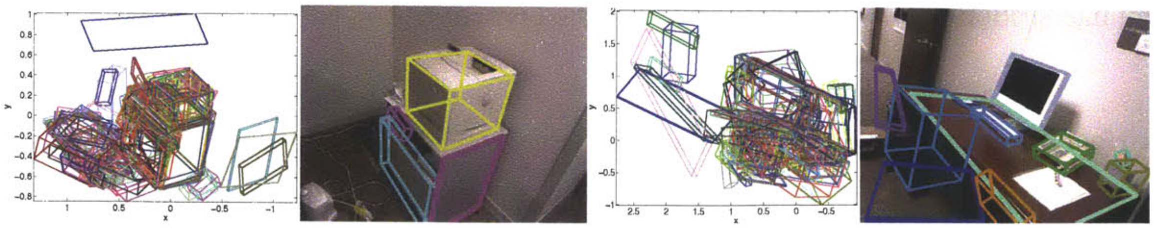


Figure 4-14: Top 200 cuboid candidates and the cuboids matched by the proposed method and projected on the color images.

Occlusion constraints The selected cuboids should have small occlusion among each other from the camera view. If a cuboid is completely occluded by another cuboid from the camera view, one of them has to be a false detection. Apart from complete occlusion, partial occlusion may indeed happen. We therefore introduce a soft term O to penalize the partial occlusion among the chosen cuboids, and we use hard constraints to prohibit the complete occlusion.

Occlusion happens if two cuboids' projection regions on the image plane overlap. The occlusion can thus be defined as $\mathscr{A}(H \cap Q)/\mathscr{A}(Q)$, where H is the projected region of the closer cuboid and Q is the projected region of the farther cuboid, $H \cap Q$ is the intersection of the two projection regions, and $\mathscr{A}(.)$ extracts the area of a region. To test which cuboid is closer, we first find the cuboid surface points corresponding to the common projected region of both cuboids and then we compare their average distances to the camera center. To count for the estimation errors, we define that there is a partial occlusion if the overlap ratio is less than u, *e.g.* 0.75 in this section, and otherwise there is a complete occlusion. The occlusion ratio between cuboid i and j is denoted as $q_{i,j}$. The occlusion reasoning is illustrated in Fig. 4-13 (b) and (c).

We introduce pairwise variable $w_{i,j}$ for cuboid candidates i and j that are partially occluded. We penalize the partial occlusion in the objective function, $O = \sum_{\{\{i,j\}:0<q_{i,j}<u\}} q_{i,j} w_{i,j}$, where $q_{i,j}$ is the occlusion ratio. Similarly to the pairwise cuboid intersection penalty term, we let $w_{i,j} = x_i x_j$. In an equivalent linear format, $w_{i,j} \leq x_i$, $w_{i,j} \leq x_j$, $w_{i,j} \geq 0$, and $w_{i,j} \geq x_i + x_j - 1$. If there is a complete occlusion between cuboid candidates i and j, they cannot coexist. We thus let $x_i + x_j \leq 1$, if $q_{i,j} \geq u$.

4.2.4 Optimization

By combining these terms, we have a complete mixed integer linear optimization:

$$\begin{aligned}
&\min\{\sum_i (c_i+\mu)x_i + \lambda \sum_{\{i,j\}} e_{i,j}z_{i,j} - \gamma \sum_k a_k y_k + \\
&\quad \xi \sum_{\{i,j\}} q_{i,j} w_{i,j}\} \qquad (4.6)\\
\text{s.t. } & z_{i,j} \le x_i, z_{i,j} \le x_j, z_{i,j} \ge x_i + x_j - 1,\\
& \forall\{i,j\} \text{ that } 0 < e_{i,j} < t\\
& y_k \le \sum_{\text{cuboid } i \text{ covers superpixel } k} x_i,\ y_k \le 1,\ \forall \text{ superpixel } k\\
& w_{i,j} \le x_i, w_{i,j} \le x_j, w_{i,j} \ge x_i + x_j - 1,\\
& \forall\{i,j\} \text{ that } 0 < q_{i,j} < u\\
& x_i + x_i \le 1, \forall\{i,j\} \text{ that } e_{i,j} \ge t \text{ or } q_{i,j} \ge u\\
& x_i = 0 \text{ or } 1. \text{ All variables are nonnegative.}
\end{aligned}$$

We solve the mixed integer linear program efficiently by branch and bound. We use the linear program relaxation that discards the integer constraints to obtain the lower bound. The upper bound is initialized by sequentially picking up low cost cuboids that do not have space or occlusion conflict with previous selections. A branch and bound search tree is expanded at the branch with the lowest active lower bound on variable x whose value is the closest to 0.5. In the two new branches, we set the chosen x to 0 and 1 respectively. We update the lowest upper bound and the lowest active lower bound in each step. A branch is pruned if there is no solution or if the lower bound by linear programming is greater than the current upper bound. The lower bound can be efficiently computed using dual simplex method since only one variable changes the value. The procedure converges quickly. With 200 cuboid candidates and a tolerance gap of 0.01, the optimization takes less than 1 second in a 2.8GHz machine. There is in fact no need to set a tight tolerance gap between the upper and lower bounds; a large gap such as 0.5 gives little performance degradation comparing to a small one such as 0.01 and enables even faster convergence.

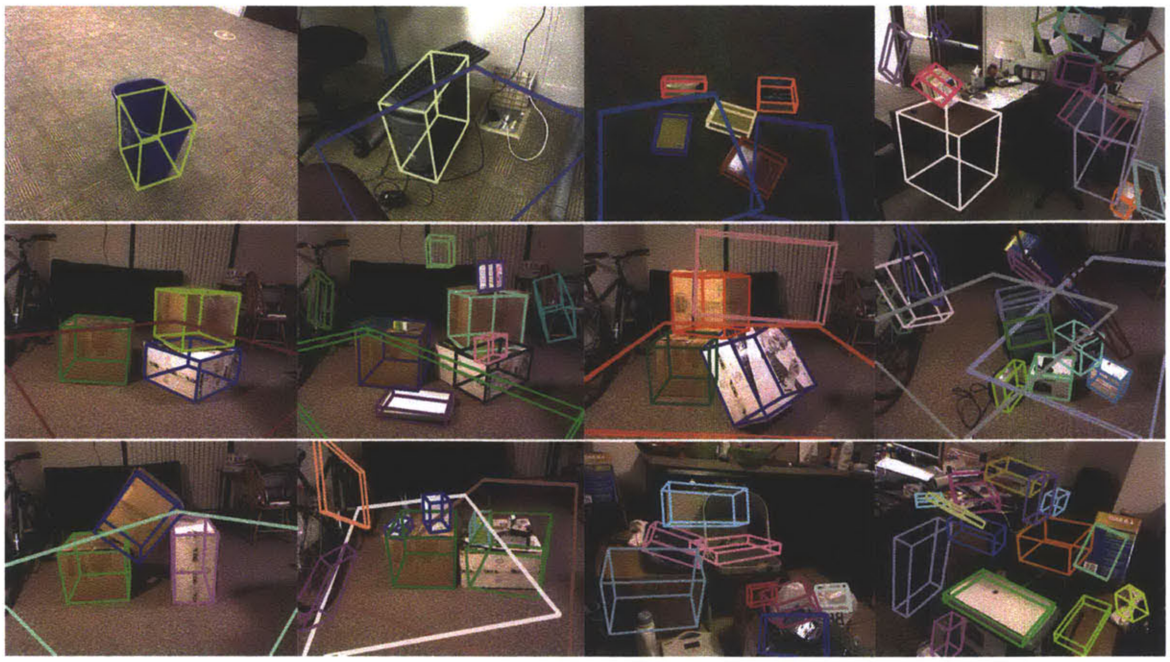


Figure 4-15: Sample cuboid matching results with the proposed method on our captured data.

4.2.5 Experiments

Qualitative evaluation

Fig. 4-14 shows two examples of the cuboid matching results from the proposed method. More results on our captured data are shown in Fig. 4-15. We further apply the proposed method to finding cuboids in the 1449 RGBD images from the NYU Kinect dataset [154]. Sample results of the proposed method on the NYU dataset are shown in Fig. 4-18. Comparing with a method [194] that uses only color images for cuboid detection, as shown in Fig. 4-16, the proposed method gives more reliable results. The proposed method is able to handle concave objects, objects with or without texture, large objects such as tables and small ones such as books. It works well on cluttered scenes.

Quantitative evaluation

We compare the proposed method with its variation and competing methods using the NYU dataset. We labeled 215 images from the 1449 images. These images are from different scene categories. Sample images with the ground truth projection overlaid are shown in

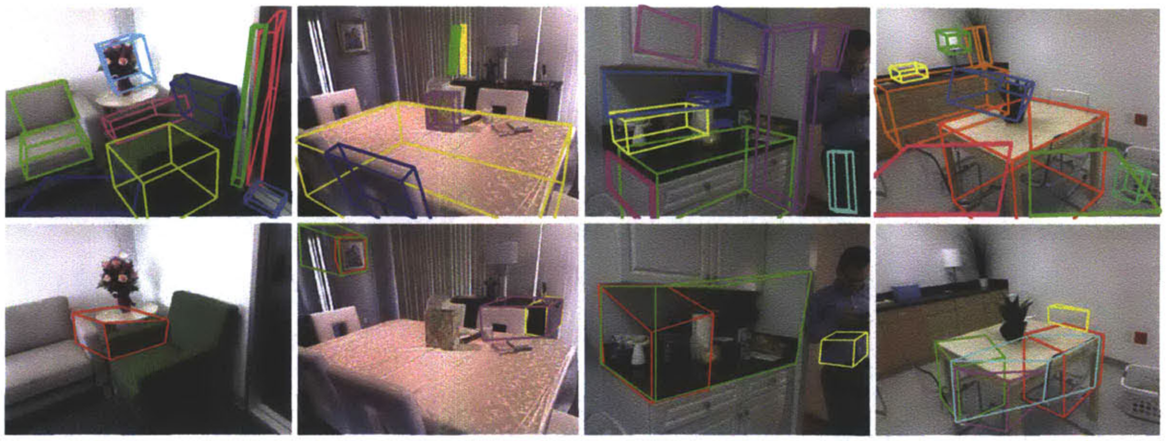


Figure 4-16: Comparison with 2D cuboid detection method [194]. The proposed method's result (Row 1) is more reliable than the result from 2D images only (Row 2).

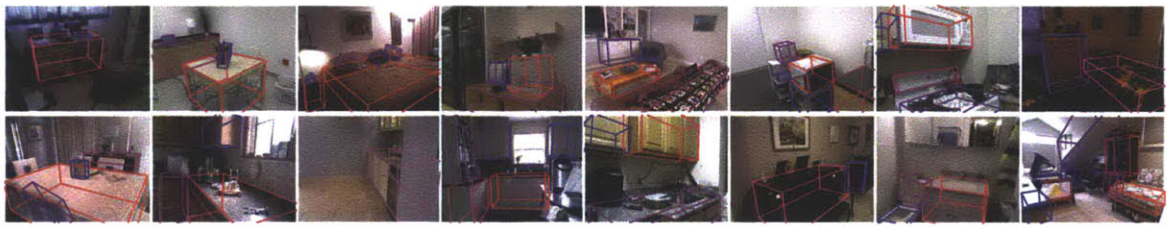


Figure 4-17: Sample images with ground truth cuboid labeling. The saliency of the labeled cuboids are denoted by the color: the warmer the color, the more salient the object is. There are 215 ground truth images and totally 476 labeled 3D cuboids.

Fig. 4-17. We extract the 3D corner coordinates of the cuboids in images using a semi-automatic method. We choose two vertical planes of a cuboid by marking the regions on the color image. The two-plane cuboid reconstruction method is used to extract the 8 corner coordinates of the cuboid. Note that we do not need to mark a complete rectangular region in each cuboid face. Instead, we just need a patch that extends to the face's width and height. The two-plane method reconstructs the cuboid automatically. Such a process sometimes needs to iterate for a few times to obtain a satisfactory 3D labeling. During the labeling, we also specify the sequence of the saliency of labeled cuboids. The object with label one is the most salient and usually it is the biggest cuboid object in the scene. Ties of saliency are broken randomly. Note that even though ground truth 2D segmentations are available for the NYU Kinect dataset, they cannot be used directly for the evaluation because our task is to match cuboids in the images.

Comparison with greedy method: We compare the proposed method with a greedy

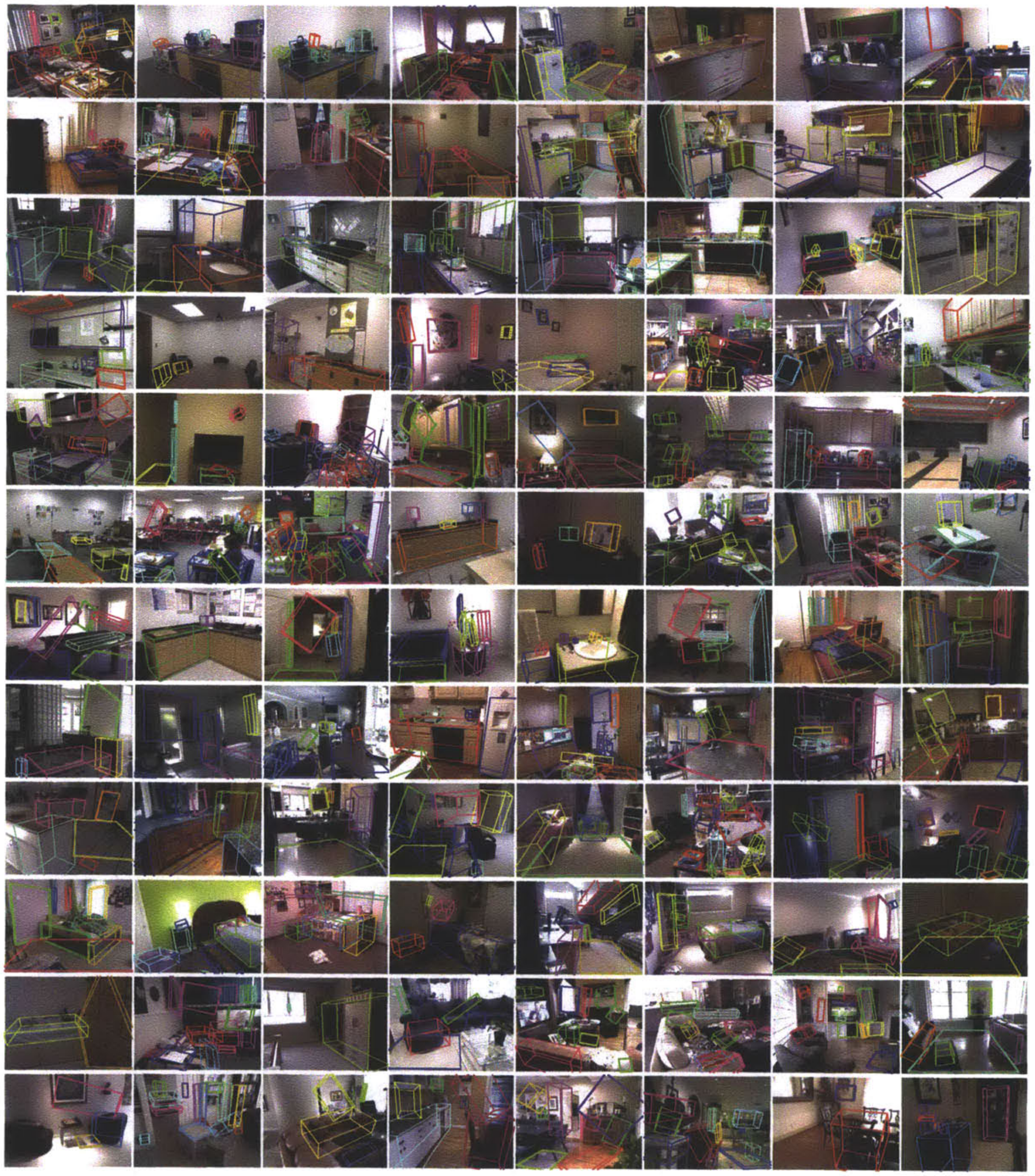


Figure 4-18: Randomly sampled results of the proposed method on the 1449 images from the NYU Kinect dataset [154]. We show the projected cuboids in the images. Our method gives reliable result in cluttered scenes.

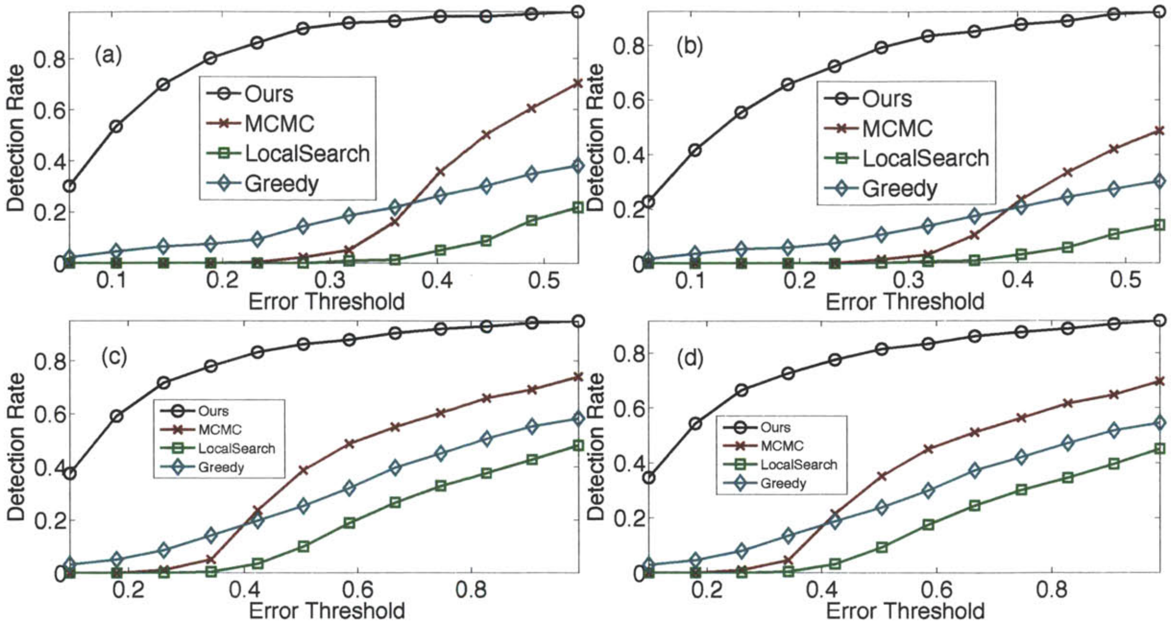


Figure 4-19: Comparing with greedy, local search [171] and MCMC [81] methods. We show the rate of finding objects when the distance error threshold changes for (a) top one, (b) top two, (c) top three and (d) top ten salient cuboids in each image.

method that chooses the top 20 cuboids with the smallest local matching costs. Our method detects less than 20 objects and 6 on average per image in the ground truth test. To evaluate the matching performance, we compute the average corner distance from each ground truth cuboid to all the detections and find the minimum distance. There is no predefined order for the 8 corners; we use bipartite matching to compute the best matching configuration for each pair of cuboids. We further normalize the average distance by the longest diagonal of the ground truth cuboid. We use the detection rate curve to quantify the performance. The curves are shown in Fig. 4-19, which illustrate the detection rate for top saliency, top two, top three and top ten objects in each image. The proposed method has much higher detection rate than the greedy approach. Local matching costs themselves are not reliable. We need to globally optimize all the cuboid matching together to achieve a reliable result.

Comparison with local search method [171] and MCMC [81]: These local search methods need to initialize the cuboid location, size and orientation. We randomly initialize these parameters. For each test image, we detect 50 cuboids with these competing methods. Color is not used by these methods. As shown in Fig. 4-19, the proposed method has much higher detection rate than these local methods. The complexity of the proposed method is

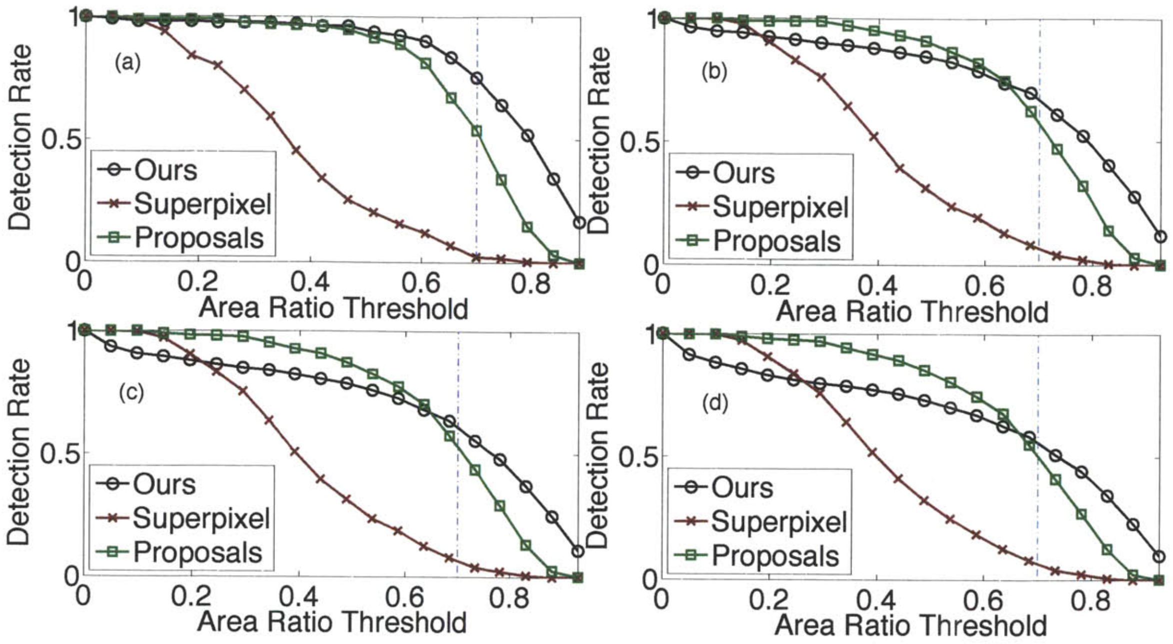


Figure 4-20: Detection rate comparison with bottom-up methods. We detect (a) the most salient object, (b) the top two, (c) the top three, and (d) the top ten objects in each image. The 0.7 threshold, above which the detection is visually correct, is marked with a vertical blue line.

also lower than these competing methods.

Comparing with bottom-up approaches: The above competing methods are top-down. We would like to compare with potential bottom-up competing methods. The segmentation method in [18] also generates object bounding boxes and thus can potentially be used to find cuboids in RGBD images. This method relies on hierarchical superpixels. If the object level segment is not available, the corresponding object cannot be detected. Assuming that we can recover a cuboid perfectly from a superpixel, the chance that the segmentation forms a complete object sets an upper bound that a bottom-up method can achieve when finding cuboids. We compare the proposed method with this upper bound.

Here we use the region overlap ratio to quantify the quality of a detection. To obtain a cuboid region in the target image for the proposed method, we project the cuboid's corners to the image plane and find the convex hull. We use the method in [18] to obtain a large set of superpixels using different parameters followed by merging neighbors progressively based on the color similarity and coplanar degree. We check how the superpixels and our cuboid projection overlap with the ground truth object foreground. The region overlap is quantified by the ratio of the region intersection to the region union. We plot the overlap ratio threshold *vs.* detection rate curve in Fig. 4-20. The proposed method gives better results than the superpixel based approach. This is not a surprise. The hierarchical superpixel method is not always able to merge two faces of a cuboid if they have very different color and texture.

Another method to generate object level regions is by object class independent proposals [42]. The original region proposal method is for color images only. For fair comparison, we include the 3D point coordinates into the superpixel detection. We adjust the weight between the *rgb* and *xyz* vectors to achieve the best performance. The proposal regions are more likely to match the cuboid objects than the superpixels as shown in Fig. 4-20. However, the region proposal method also has a hard time to find many cuboid objects especially large objects with complex textures. Above the region overlap threshold 70%, the detection rate of the proposed method is always higher than that of the region proposal method. The object independent proposal method has higher detection rate if the region overlap threshold is less than 0.6 because it uses many more region detections – several thousand, comparing to less than 20 detections of the proposed method. In fact, the low threshold detection rate is not important because a detection that has less than 60% overlap with the ground truth is most likely wrong and it will be hard to recover the 3D cuboid object using such segments. The bottom-up region based method is also more complex. There are often 5000 region candidates and it takes 5-10 minutes to find them in a 2.8GHz machine. Our proposed method uses many fewer cuboid candidates; it is more efficient. The optimization takes less than one second with the same machine. The whole process that includes extracting candidates, computing intersection and occlusion takes less than one minute per image.

4.2.6 Conclusion

We propose a novel method to match cuboids in RGBD images. The proposed linear method guarantees the global optimization of cuboid matching. It also automatically determines the number of cuboids in the scene. Our experiments on thousands of RGBD images of a variety of scenes show that the proposed method is accurate and it is robust against strong clutter. We believe that this method is a useful component to help 3D scene understanding.

4.3 3D Annotation Tool

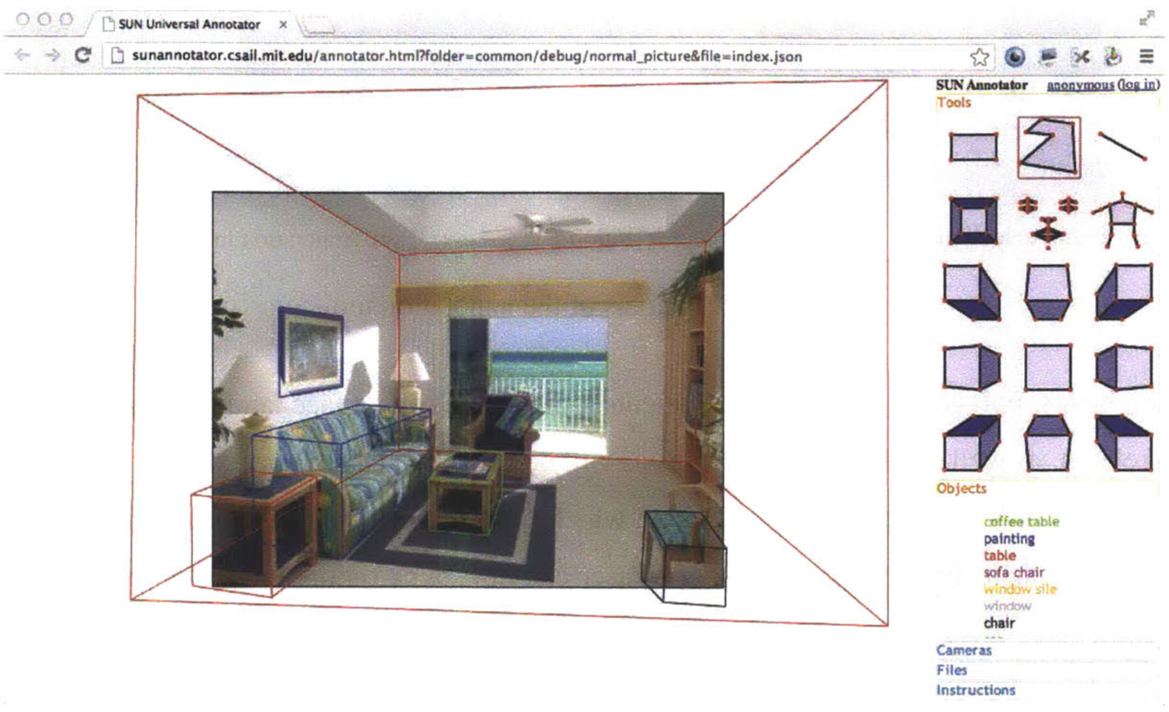
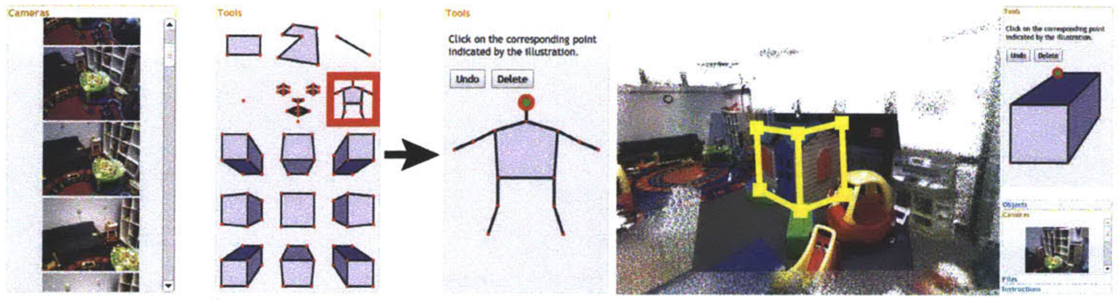


Figure 4-21: User interface of the 3D online annotation tool.

To label different kinds of annotations in a consistent tool and file format, we design an universal online annotation tool for images, panoramas and sequences in 3D, for different kinds of annotations, *e.g.* 2D bounding box, 3D bounding cuboids, human body pose, face landmarks, lines, polygons etc. Our tool is operated through a web browser and is implemented using HTML5 and WebGL in Javascript.

A snapshot of our web-based annotation tool is shown in Figure 4-21. On the left, it shows an image to be annotated. If it is a frame from a video with 3D reconstruction, it will be shown with a 3D point cloud, using the camera pose estimated with structure from motion. The user can choose to switch to another frame of the video by clicking the wanted frame on the right panel (Figure 4-22(a)). The user can also choose to see the point cloud and all the camera poses from a completely new viewpoint without being locked from a frame.

The "Tools" panel on the top right (also Figure 4-22(b)) is the place where the user can choose a tool to start labeling. We have tools for 2D geometric figures such as bound-


Cameras
Tools
Tools
Click on the corresponding point indicated by the illustration.
Undo
Delete
Click on the corresponding point indicated by the illustration.
Undo
Delete
Objects
Cameras
Files
Instructions

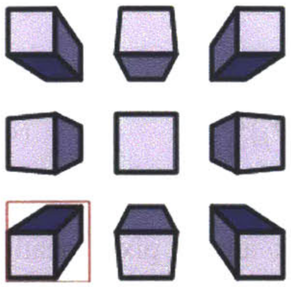
(a) Frame selection (b) Annotation tool selection (c) Annotation in action

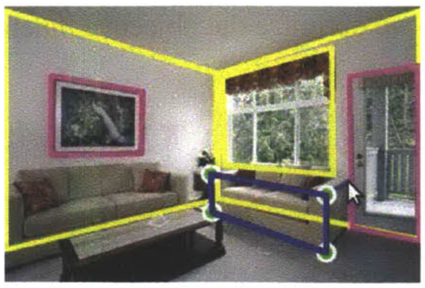
Figure 4-22: Panels of the annotation tool.

ing boxes, polygons, straight line segments, points, and even a face landmark and human skeleton. as well as 3D bounding boxes as will be explained later. When a tool is selected, the panel will become a guide to explain which point to click next to annotate the object, based on their different properties. The user can now click on the image, or outside the image (*e.g.* a corner of a table is outside the available field of view) to finish the annotation. In such a way, the user can freely provide different annotation using the same unified interface. After the annotation, the user can enter the name of an object, and also adjust the keypoint location afterwards. The tool can also render a 360 ull view panoramic images as a texture wrapped on a 3D sphere, and the user can drag the image to rotate and annotate.

Because we have a full 3D reconstruction, it is more natural to annotate 3D bounding boxes around objects rather than the 2D bounding boxes for 2D images [44]. For this purpose, we introduce a tool for labeling the bounding boxes of objects that have a rough cuboid shape. To start annotating an object, the user chooses one of the viewpoints of 3D cuboids (Figure 4-23(a)). Then, the system shows a guide (Figure 4-23(c)) to indicate which corner of the 3D bounding box the user should click on now. The annotation of the object is complete when all the visible corners of the cuboids are annotated, and a popup dialog bubble will appear querying for the object category name. We ask the annotator to annotate the 3D bounding box to be as tight as possible and align with the major orientation of the object. When annotation finishes, the annotator is allowed to adjust the corner locations. With the manual online annotation, the next step is to estimate the 3D geometry from the annotation offline.

Note that the key novelty of our tool is to first ask the user to explicitly choose one of


(a) choose a viewpoint.


(b) labeling interface.

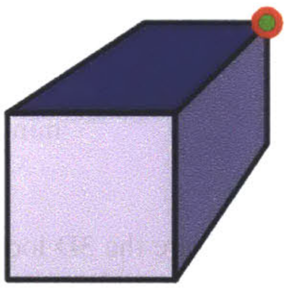

(c) next corner to label.

Figure 4-23: 3D cuboid annotation. To annotate each 3D cuboid, the user pick on view point in (a), and then click on the key points of the 3D bounding boxes on the image in (b). While annotating, the screen display an indication about what is the next corner to click on, as shown in (c).

nine possible viewpoints for the object they are labeling, and subsequently ask the user to click each visible corner, as shown in Figure 4-22 (c). We found this approach to be much easier to use than the keypoint-based approach of [194], where the viewpoint is implicit.

However, the user's clicks are in 2D, and we need to reconstruction a cuboid in 3D by minimizing the sum of reprojection errors for the visible corners. We use a simple linear factorization as initialization, and optimize the following objective function

$$\min \sum_i \|\tilde{\mathbf{x}}_i - \mathbf{K}\left[\mathbf{R}|\mathbf{t}\right]\mathbf{X}_i\|^2, \tag{4.7}$$

where $\mathbf{X}_i$ are the 3D locations of the bounding box corners, and $\tilde{\mathbf{x}}_i$ are the pixel locations annotated by the user. We solve this minimization using the Ceres Solver [3] to obtain the reconstruction of the 3D bounding box, up to a scale.

If there is a depth map associated with the image (or frame of a video) available, we can also make use of the depth to make the reconstruction more reliable. Therefore, we recover the shape of the cuboid from the user's corner annotations by fitting planes to the point cloud, defining an initialization using these planes, and finally optimize jointly using both depth information and user annotation. In more details, first, we obtain the point cloud consisting of points whose 2D projections are on each face of the 3D bounding box. We then use RANSAC to fit a 3D plane for each face's point cloud. After that, we intersect the 3D planes and use the inliers' projections to obtain an initial estimate of the cuboid. Using this solution as initialization, we optimize the objective function

$$\min \sum_i \|\tilde{\mathbf{x}}_i - \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}_i\|^2 + \kappa \sum_j \sum_{k \in 2\mathrm{D}(j)} d(\mathbf{Y}_k, \mathbf{P}_j)^2, \tag{4.8}$$

where $\mathbf{X}_i$ are the 3D locations of the bounding box corners, and $\tilde{\mathbf{x}}_i$ are the pixel locations annotated by the user. For the j-th face $\mathbf{P}_j$, $\mathbf{Y}_k$ is a 3D point cloud associated with the face, and $d(\mathbf{Y}_k, \mathbf{P}_j)$ is the Euclidean distance from the 3D point $\mathbf{Y}_k$ to the 3D plane $\mathbf{P}_j$. Again, we solve this minimization using the Ceres Solver [3] to obtain the final estimate. We use this method to create an RGBD cuboid dataset with 669 images with 1106 3D cuboids[1].

[1] As part of the master thesis by Erika Lee under my supervision.

Chapter 5

Place-centric Scene Understanding

When a human being is navigating in an environment, the visual input is basically a video. But humans are not understanding the scene as many individual disconnected snapshots from a video. We understand the environment as an integrated space. Essentially, to jointly reason about 3D structure and viewpoint in space, we should have a place-centric representation of the 3D space around the observer.

However, the SUN database is view-based, in the sense that it only contains snapshots that capture particular views but not the full 3D extent of a place. We desire a *place-centric* representation, i.e. one that has a comprehensive model of the entire 3D space as its representation rather than a limited set of views. To study this kind of representation, we construct a database that allows us to model the 3D context of objects in space, to reason about mechanics and intuitive physics, and to answer questions such as: "what does this object look like from behind?". A place-centric description of a scene is more complete and largely resembles the real world, providing the strong advantage of being more invariant to viewpoint changes. Therefore, we introduced the SUN3D database of full environments scanned with an RGBD sensor, while walking through the space mimicking human exploration. We designed a robust structure-from-motion pipeline for long RGBD sequences to produce full 3D models of entire houses, offices, etc. However, existing methods of

The materials presented in this chaper are based on [192, 191] in collaboration with Antonio Torralba, Yasutaka Furukawa, and Andrew Owens.

structure from motion that use RGBD cameras often break with trying to reconstruct large places. Therefore, we introduce a bundle adjustment algorithm that incorporates semantic labels introduced by user to obtain an accurate 3D reconstruction of large places scanned with an RGBD camera, and to provide object segmentations and labels for all the objects in the environment. The user only needs to label some frames from the video using a LabelMe [144] style annotation tool. The method will propagate the labels to all the other frames and use the annotations to help constraint the 3D reconstruction.

This place-centric representation also has many applications such as obtaining a scene-level 3D reconstruction of large environments. In the last section of this chapter, we will discuss how to use this as a tool to organize big visual data to provide a photo-realistic map for museum.

5.1 Introduction

The popularity of the Microsoft Kinect and other depth-capturing devices [158] has led to a renewed interest in 3D for recognition. Researchers have extended traditional object and scene recognition datasets (*e.g.* [44, 184, 47, 105]) to incorporate 3D. For example, [101, 163] are an evolution of popular 2D object datasets such as Caltech101 [47] to 3D objects captured by an RGBD camera. The NYU Depth dataset [154] and [87, 57] go beyond objects by capturing RGBD videos of scenes and labeling the objects within. Such datasets are a natural extension of *scene-centered* datasets such as PASCAL VOC [44] and SUN database [184]. However, current 3D datasets still inherit many of the limitations of traditional 2D datasets. That is, they are still **view-based** and consists of a collection of short videos and snapshots that consist views but not the full 3D extent of a place.

We desire a dataset that has a full 3D model of a place (e.g., an entire apartment) rather than a limited set of views (Fig. 5-19). Such a database would allows researchers to develop and evaluate better 3D context models, to infer object affordances, and support the use of quantitative geometry. This will allow us to ask questions like: "what does this object look like from behind?" and "what can the space be expected to look like beyond the available field of view?"

Building a 3D model of a place from RGBD video is not a trivial problem, as the depth data has to be integrated across frames to build a 3D model that is consistent. Despite recent progress on 3D reconstruction, the existing tools to automatically build such a 3D database of large places are not reliable. In this chapter we introduce a method for 3D reconstruction and object labeling that integrates the process of user-driven semantic object labeling with the 3D reconstruction algorithm in order to: (a) produce better 3D reconstructions, while (b) providing an intuitive object annotation tool.

There are several systems that employ RGB-D cameras to produce 3D models of places, *e.g.* [71, 41, 121]. These systems are oriented towards real-time reconstructions. Real-time reconstructions are interesting because they allow the user to interact with the tool to correct errors as they appear. However, scanning a big place remains challenging as reconstruction errors are frequent and the user needs to understand how to correct the error and rescan

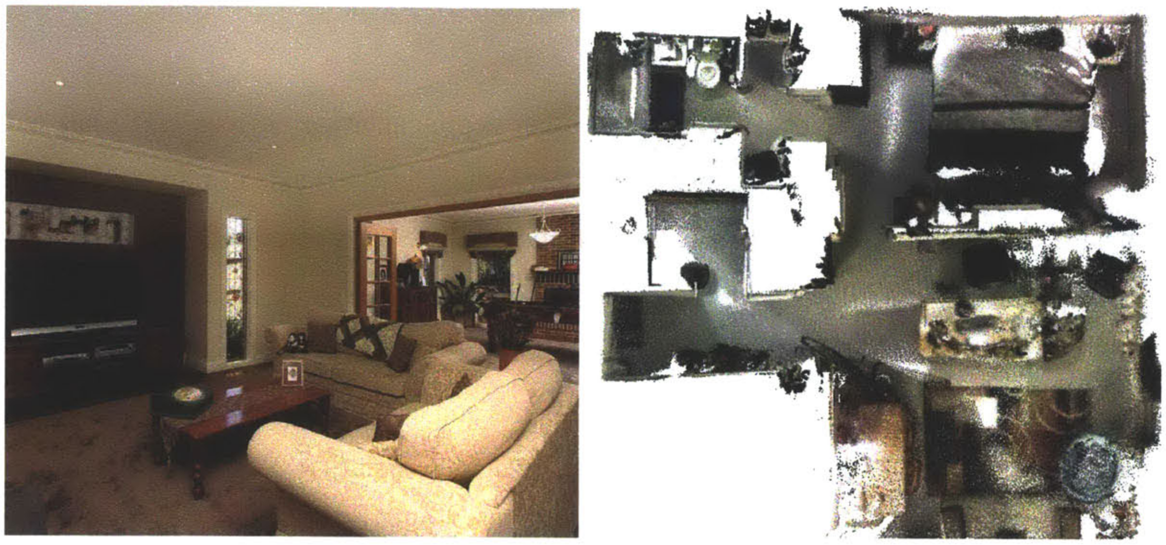

(a) 2D view-based SUN database [184]
(b) Our 3D place-centric database

Figure 5-1: The difference between a view-based scene representation and a place-centric scene representation.

the place. If the error appears due to a loop closure, then the user could have a hard time correcting the error. Furthermore, real-time capture systems put strong constraints on the hardware used for data collection as they are computationally intensive.

Offline reconstruction methods, on the other hand, put less constraints on the data-collection hardware, and thereby facilitate data capture via crowdsourcing. However, once the scanning is done, reconstruction errors will be frequent, as the user was not aware at the time of the scanning, which areas are more difficult and may require more data. Therefore, we introduce here a new method that incorporates user input in order to build correct 3D reconstructions of large spaces offline. Our approach is complementary to recent efforts for real-time 3D reconstruction [71, 41, 121].

As we want to additionally segment and label the objects present in the environment, we design the processes of 3D reconstruction and object labeling to mutually support one another (see Figure 5-2). If reconstruction was perfect, then object labeling would be easy – merely requiring one to label a few frames covering distinct viewpoints of the object. The reconstruction would then be used to propagate these annotations to the rest of the frames. On the other hand, if the objects were annotated in every frame, then reconstruction would improve dramatically since consistencies between frames would be used as constraints in

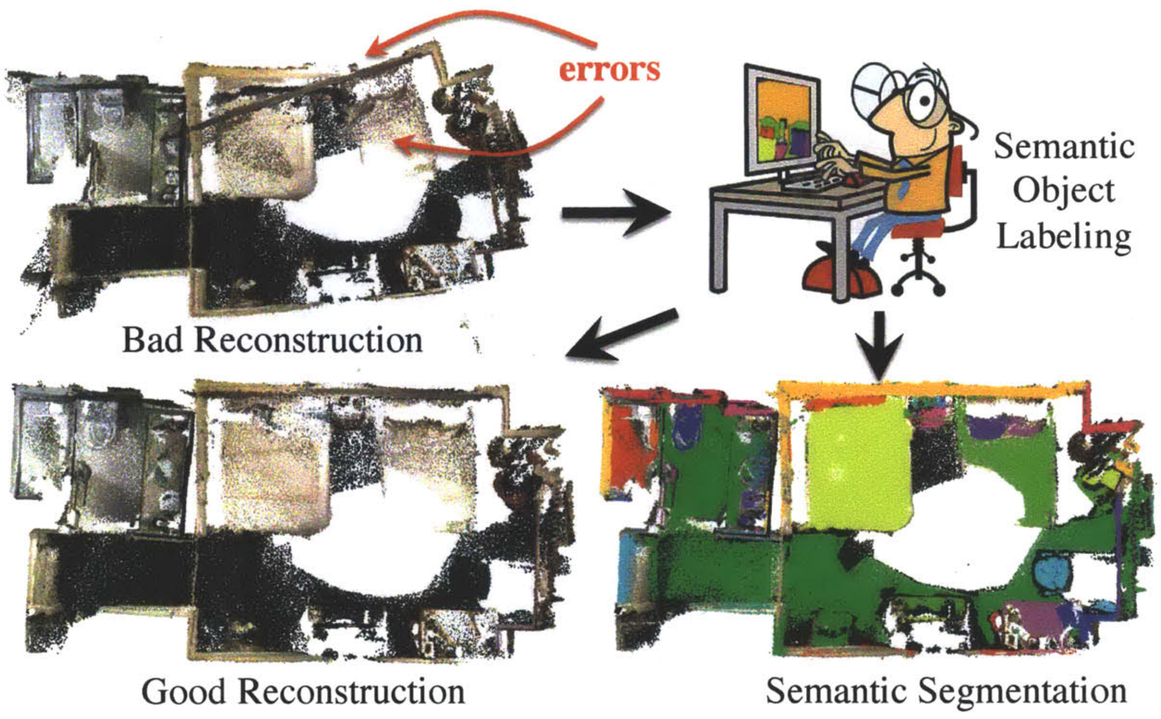


Figure 5-2: Semantic labeling as a way to manually correct reconstruction errors.

optimization. Using object annotations to improve 3D reconstruction, and in turn 3D reconstruction to efficiently propagate object annotations, allows us to obtain both simultaneously. Also, objects are easy to annotate even for non-expert users, as it does not require knowledge of 3D geometry. Furthermore, introducing semantics to help 3D reconstruction opens the door for new exciting types of constraints. For instance, if we know where the mirrors are, we can remove the reflected points from the reconstruction.

Related work There are several approaches that researchers use to introduce 3D information in current databases. For instance, [147] used a 3D laser scanner to capture a database of snapshots with depth information. [57] uses a Velodyne laser scanner to get 3D reconstructions of streets. [101, 163, 87, 154, 98, 11, 94] use Kinect cameras to capture 3D. A less direct way of getting useful 3D information is to rely on user input. For instance, [79, 143, 78, 68, 198, 131] use manual annotations to provide 3D information about images. Most of those datasets provide 3D information for single images.

There are several 3D datasets that capture scenes [154, 98, 11, 57] rather than simple snapshots or videos. Especially noteworthy is the NYU Depth V2 dataset [154], which

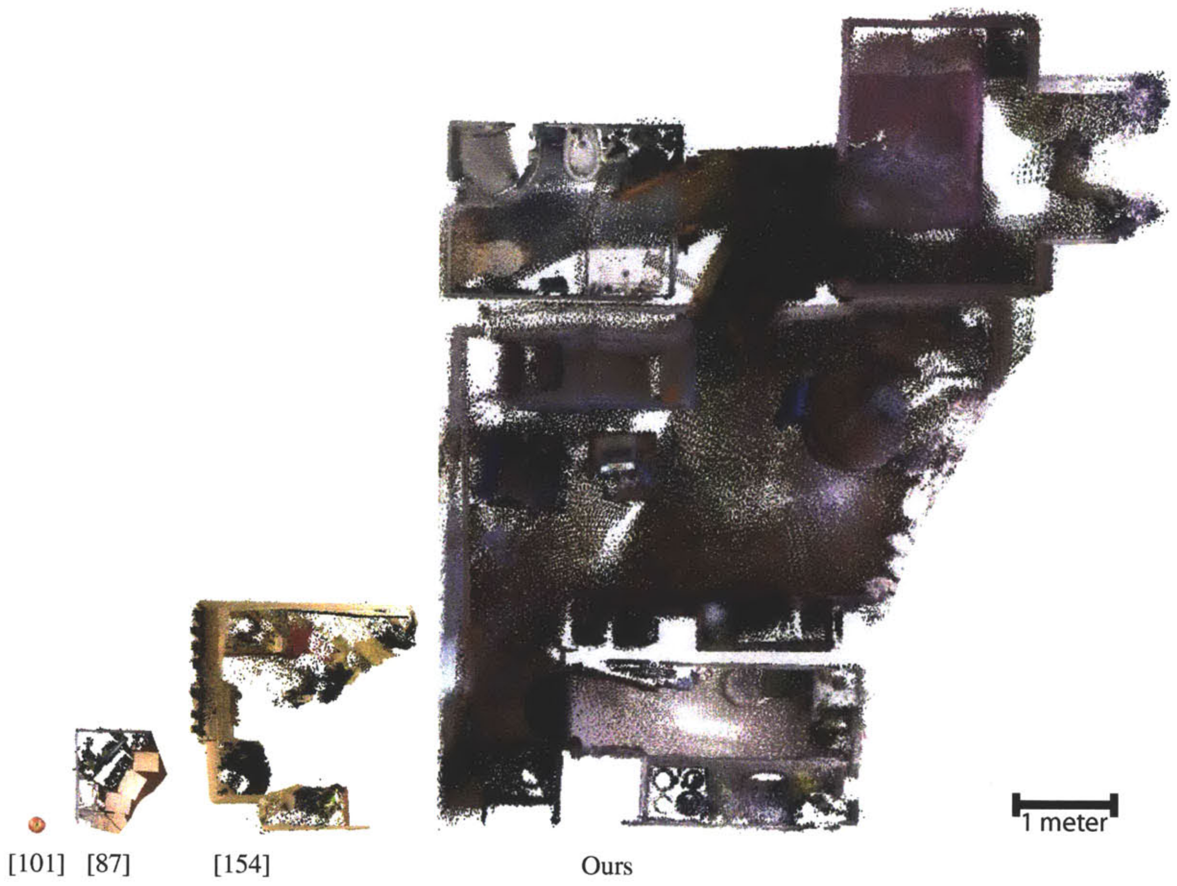


Figure 5-3: Top view of the reconstruction results to illustrate the typical 3D space covered by four different datasets at the same physical scale (meter). (a) RGB-D Object dataset [101]: multiple views of a small object; (b) Berkeley 3-D Object [87]: one view of a scene; (c) NYU Depth V2 [154]: one corner of a room; (d) Ours: a full apartment including bedroom, living room, kitchen, and bathroom.

contains scans of a large number of scenes. Despite the large quantity of scans, it is still very much a view-centric dataset. As an experiment, we reconstructed a large sample of the sequences in the NYU dataset using our structure-from-motion pipeline (Section 5.2) and found the coverage of most spaces to be incomplete. Figure 5-3 compares the size of the spaces scanned in several datasets. We can see that a typical scans in most datasets only cover a small portion of each place, such as the corner of a room.

Kinect@Home [11] uses crowd-sourcing to ask users to record RGBD video online. However, because most online users only have Microsoft Kinect with a short cable, the data collected mainly contains short sequences that cover a small portion of the space. [98] contains some 3D reconstruction results for home and office, mainly aimed at robotics

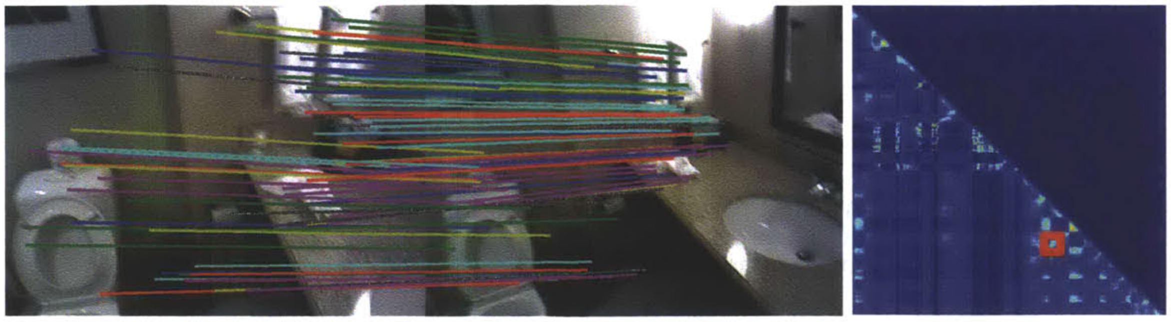


Figure 5-4: Example for loop closing. SIFT matching between two frames on the left, and the score matrix.

applications. The dataset is relatively small, and is still view-based, similar to the NYU dataset [154]. Object annotation is also a tedious process that requires a human to provide ground truth. In the case of the NYU dataset [154] each frame is annotated independently using a LabelMe style interface. In term of methodology, the closest work to us is probably [12] where an automatic system developed using sampling methods.

In this chapter, our goal is to develop tools that will allow building an annotated database of full 3D places. Although there are several great tools available to annotate 2D images and videos [144, 199, 180, 123], they would be less convenient to use for RGB-D sequences because they do not exploit the 3D structure of the problem for label propagation. In this chapter we introduce a structure-from-motion pipeline that makes use of RGB, depth, and semantic annotations to improve the robustness of the 3D reconstructions. The tool uses 3D structure to propagate object labels to unlabeled frames and uses object labels to improve the 3D reconstruction.

5.2 3D reconstruction with human in the loop

We propose a system to let human help the pose estimation for RGB-D video. We start by first running a fully automatic Structure From Motion (SFM) system to reconstruct the initial camera poses for all frames (the rest of this section). Then, the user labels the objects in the video sequence, side-by-side with our 3D label propagation algorithm (Section 5.3). The user starts by choosing a keyframe to label, and our system propagates the labeling from the good key frames to the remaining frames and the user keep refining for other frames. After the annotation is done, our system takes the user semantic labels as constraints in a bundle adjustment procedure (Section 5.4), and produce a better reconstruction. Because of the high requirement of reconstruction quality, our target input is RGB-D video, but the same method can be natural extended to RGB images as well.

5.2.1 Automatic initialization

Frame-to-frame registraction As with many structure-from-motion systems, we match every consecutive pair of frames and compute a relative transformation between them. We begin by matching keypoints using SIFT and remove poor matches using the ratio test [113][1]. Within the set of SIFT keypoint, we choose the ones with valid depth value from the RGBD camera, and use RANSAC to find a 3D rotation and translation transformation to match the two frames, using 3 points in the inner loop of RANSAC. Using the best-estimated transformation from RANSAC, we warp the depth of the first frame to predict the depth of the second frame, and check the consistency between the prediction and the real depth to the other frame. If the depth consistency is low, *i.e.* the overlapping of the two depth maps is small or the difference in the overlapping area is big, then we discard the transformation from RANSAC. Instead, we use Iterative Closest Point (ICP) algorithm [121] to estimate the best alignment between the two point clouds from the two frames, and use this to replace the rotation and translation estimated from RANSAC. With such an estimated transformation, either from RANSAC or ICP, we obtain a list of inlier SIFT keypoint correspondences.

Loop closure To detect loops, we use Bag of Word models to compute a feature vector for each image using td-idf, followed by a geometric verification. For each video, we uses k-means to train a codebook for bag of world model[2]. For a given frame, the SIFT keypoints are detected and their descriptors are used as a histogram of to compute the

[1]To improve matches on textureless surfaces (which are common in indoor scenes), we start with a stringent ratio-test threshold (0.6) and increase it until we obtain a sufficient number of matches.

[2]Our codebook size is fixed to be 4,000 trained using 60,000 SIFT keypoints randomly sample from the same video.

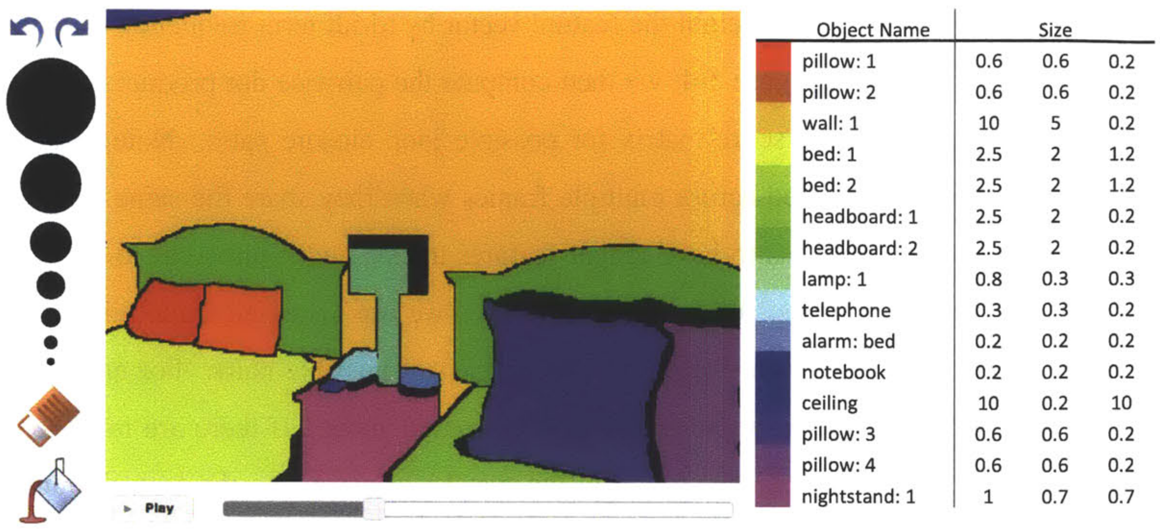
frequency of visual word. We weight the feature vector by td-idf term frequency from the same video. As shown in Figure 5-4, we then compute the pairwise dot product between feature vector to obtain the score matrix for possible loop closure pairs. Note that the SIFT feature tracks are linked across multiple frames when they share the same location at each frame. This allows us to have longer features tracks, and it turns out to be quite important in our experiments. With the score matrix, we use Gaussian smoothing, non-maximal suppression, and then dilation to pick the list of possible pairs. For each pair, we run the pairwise frame-to-frame registration discussed above. If there are more than 25 SIFT keypoint correspondences found in the matching, we merge the feature tracks. By setting very conservative thresholds, our loop closure usually have very high precision with low recall, which is more desirable, as errors in matchings are difficult to deal with in bundle adjustment.

Joint 2D+3D bundle adjustment The estimated transformations between neighboring frames are concatenated together to obtain an initial pose estimation of the sequence. Then, we use the time ordering of frames and their inlier SIFT correspondences to link keypoint tracks for bundle adjustment. We use a joint 2D and 3D objective function for our bundle adjustment as follows:

$$\min \sum_{c} \sum_{p \in \mathrm{V}(c)} \left(\left\| \tilde{\mathbf{x}}_p^c - \mathbf{K} [\mathbf{R}_c | \mathbf{t}_c] \mathbf{X}_p \right\|^2 + \lambda \left\| \tilde{\mathbf{X}}_p^c - [\mathbf{R}_c | \mathbf{t}_c] \mathbf{X}_p \right\|^2 \right)$$

where $\mathbf{K}$ is a fixed intrinsic matrix read from device middleware, $\mathbf{R}_c$ and $\mathbf{t}_c$ are the rotation matrix and translation vector for the c-th camera corresponding to a frame, $\mathbf{X}_p$ is the 3D location of a 3D point visible from the c-th camera (*i.e.* $p \in \mathrm{v}(c)$), and $\tilde{\mathbf{x}}_p^c$ and $\tilde{\mathbf{X}}_p^c$ are the observed 2D pixel location and 3D location in the camera coordinate system respectively. In our experiments, our joint 2D and 3D bundle adjustment seems to be significantly more robust than either 2D or 3D bundle adjustment alone [3].

[3] We use λ (= 1000 in our experiment) to weight the two terms, which are in pixels and meters respectively.



	Object Name	Size		
🟥	pillow: 1	0.6	0.6	0.2
🟥	pillow: 2	0.6	0.6	0.2
🟧	wall: 1	10	5	0.2
🟨	bed: 1	2.5	2	1.2
🟩	bed: 2	2.5	2	1.2
🟩	headboard: 1	2.5	2	0.2
🟩	headboard: 2	2.5	2	0.2
🟩	lamp: 1	0.8	0.3	0.3
🟦	telephone	0.3	0.3	0.2
🟦	alarm: bed	0.2	0.2	0.2
🟦	notebook	0.2	0.2	0.2
🟦	ceiling	10	0.2	10
🟪	pillow: 3	0.6	0.6	0.2
🟪	pillow: 4	0.6	0.6	0.2
🟪	nightstand: 1	1	0.7	0.7

Figure 5-5: Simple user interface: a paint-based tool to quickly segment an image using colors corresponding to object names.

5.3 Multi-view object annotation

We want to ask a user to label the objects and use it to correct reconstruction errors, because it is a very intuitive task that is shown to be doable without much training [144]. The user is shown a video player with the object annotation superimposed on the image frame, and they can play the video using a regular video control bar, as shown in Figure 5-5. And we provide a paintbrush to for the user to color image regions, name the object, and specify its physical size. Whenever a user labels or corrects a frame, the object annotation will be used to propagate automatically to other frames, and the user doesn't need to label them if the propagation is correct. The task is finished when the user is satisfied with the object segmentation on all frames.

5.3.1 Interaction at each frame

When the user scrolls to a frame, if the frame has already been labeled, then the stored object annotation will be shown. Otherwise, if the frame hasn't been touched or confirmed manually, the 3D-based label propagation algorithm will try to propagate labels from other keyframes to this frame. Now, the user can choose to carry out one of the following three operations: **correct** all mistakes, **confirm** that there is no error, or simply **ignore** this frame

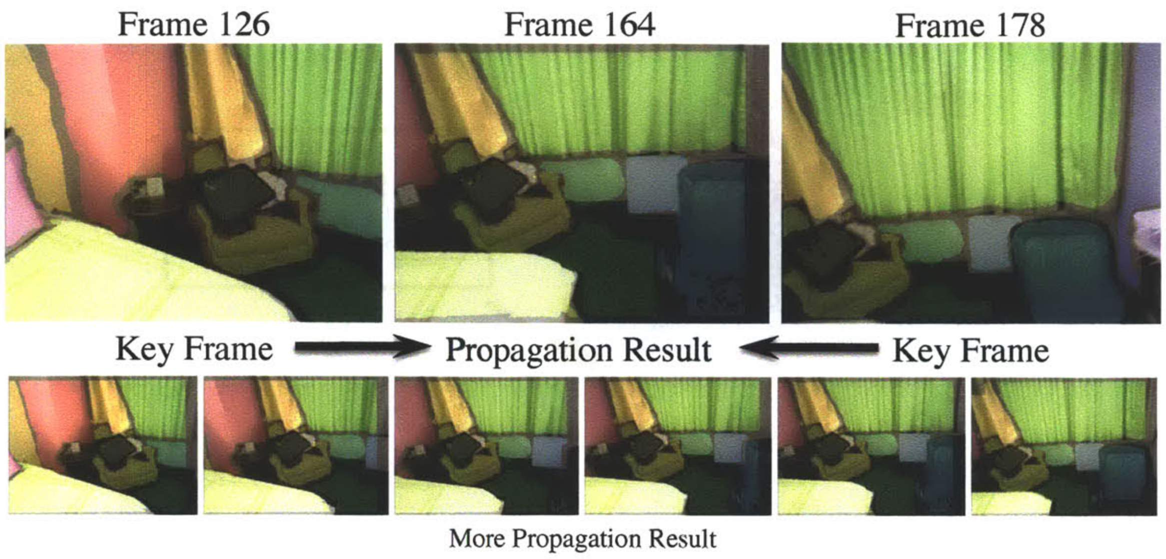


Figure 5-6: 3D label propagation.

and continue to other frames. By **confirming** the correctness, it means that no object is mislabeled as other objects, but it doesn't mean that all objects must be labeled. It is okay for some regions to be labeled as "null" (black color), which is the default value when no reliable propagation is obtained. If the user decides to **correct** the frame, we require that they fully correct all errors, *i.e.* all regions mislabeled as other objects. Once again, it is okay to leave some regions as "null" in order to finish this step. When the user either confirms the correctness or finishes correcting the error, the frame automatically become a key frame, and the object annotation from this frame will be used to propagate to other frames. If the user moves to other frames without correcting or confirming, it means that the user didn't want to provide any feedback, and the algorithm does nothing.

5.3.2 Annotation propagation

We use the very reliable depth map from RGB-D data to have a very simple but powerful label propagation scheme. For an unlabeled frame, we retrieve a small set of nearby keyframes based on the camera distances and view angles. For each of these retrieved keyframes, we take all the pixels with valid depth from RGB-D data, and their labels, and reproject the 3D point cloud to the current frame to accumulate a vote at each pixel, if the reprojected depth is within a small threshold from the depth of the current frame. Then, we

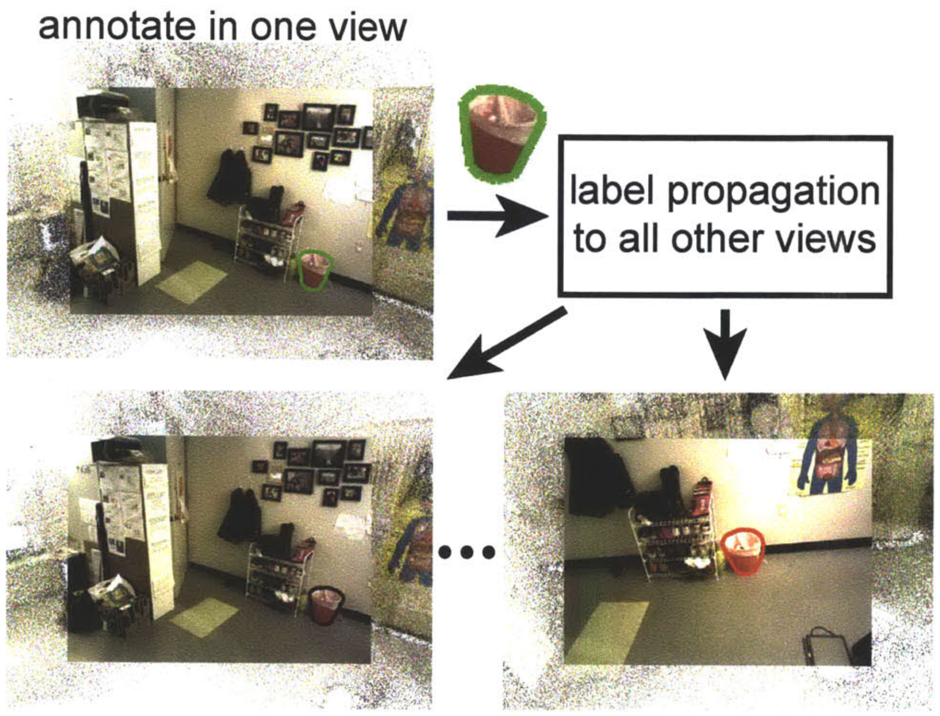


Figure 5-7: Annotation propagation.

just take the maximal vote, and smooth the labeling result using max filtering. With such a very simple propagation scheme, we observe that the results are usually very stable, as we can see in the example shown in Figure 5-6.

5.3.3 Conflict list

The major source of propagation error comes from camera pose errors, which we intended to correct by object annotation. The user can correct the errors produced in one frame, but it is very tedious to correct the errors on every nearby frame. Therefore, we maintain a conflict list between pairs of frames. When a user corrects a major mistake in a frame, the algorithm checks to see which keyframes the wrong label is propagated from, and place them into the conflict list with the current frame. All nearby frames will exclude frames from the conflict list from being used in propagation. Therefore, correcting a major mistake from one frame will significantly improve the quality of propagation for all nearby frames. Using such a simple mechanism, the camera pose errors don't cause much trouble to the user. Furthermore, if an object from one view gets deleted many times, it means that that

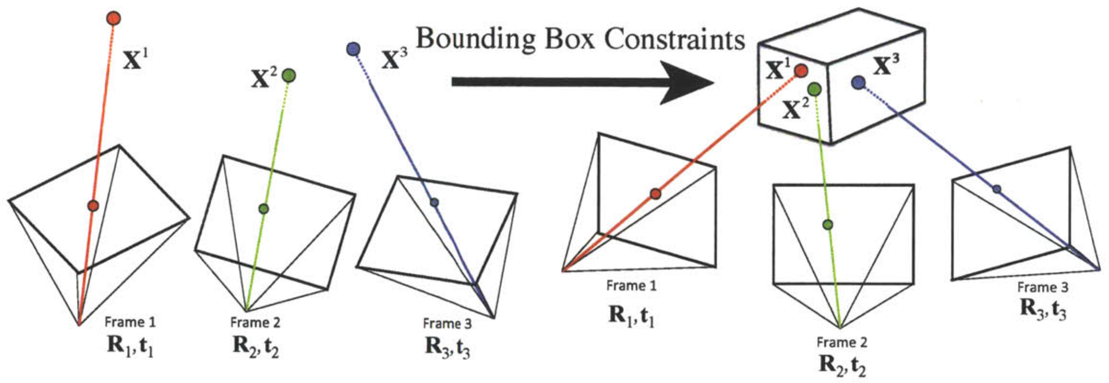

$\mathbf{X}^1$
$\mathbf{X}^2$
$\mathbf{X}^3$
Bounding Box Constraints
Frame 1
$\mathbf{R}_1, \mathbf{t}_1$
Frame 2
$\mathbf{R}_2, \mathbf{t}_2$
Frame 3
$\mathbf{R}_3, \mathbf{t}_3$
$\mathbf{X}^1$
$\mathbf{X}^2$
$\mathbf{X}^3$
Frame 1
$\mathbf{R}_1, \mathbf{t}_1$
Frame 2
$\mathbf{R}_2, \mathbf{t}_2$
Frame 3
$\mathbf{R}_3, \mathbf{t}_3$

Figure 5-8: Before and after the semantic bundle adjustment. Our algorithm basically pulls a list of points beyond to the same object into one 3D bounding box for the object. Together with constrains from other objects, and keypoints, the camera pose can be reliably estimated.

object is very unlikely to be correct. The system offer a suggestion to the user to change the object in the original keyframe.

5.3.4 Size and context

For each object, the system will automatically suggests a 3D physical size for the whole object based on the object category, and the user can change the recommended setting. The user can also provide context constraints between objects, such as that one object is parallel to another object, and this constraint will be used in the semantic bundle adjustment in the next section. For the naming of the object, we ask the user to name an object first with its object category, followed by a short description of where the object is located, so that they can specify if the same object appears twice in another frame (since this is a sign of a possible reconstruction error). For object categories with many instances in the same place, *e.g.* hundreds of chairs in a classroom, we simply avoid using them for correcting the 3D reconstruction errors in the next section.

5.3.5 Discussion

The annotation task is very intuitive because it requires no knowledge of 3D and geometry, and it can be used easily by general users. Most of the labeling time is spent on increasing the labeling coverage of an environment, instead of correcting errors. This is because that the RGB-D depth and local camera pose estimation are usually very reliable, and the conflict list effectively deletes bad source keyframes when the camera poses are inconsistent. We have also considered alternative ways, such as drawing bounding boxes in 3D, but all these approaches require the user to be able to interact with 3D information in a flat-screen environment, which requires significant training and geometrical knowledge. Also, when the camera poses are very wrong, showing the point cloud in 3D is a disaster for user experience.

There are also some objects with highly reflective surface properties, such as mirror, windows, glass bottles *etc.* . For these objects, propagation can still be carried out correctly to a certain degree. For example, the world inside a mirror is a reflective copy of the real world, and it can be reconstructed in the same way, to help propagate the label, *i.e.* "mirror". In the final reconstruction point cloud visualization, we simply delete all 3D points that come from these objects, to avoid creating a reflective copy of the world that will produces a lot of artifact. Also, we use the user annotation to remove the ceiling for better visualization.

5.4 Semantic bundle adjustment

The object annotation not only provides object segmentation, but can be used to effectively correct the camera pose estimation errors as well. For example, if the same object instance appears twice in different places of the environment, it signals that the camera pose estimation is incorrect. We desire a way to allow the user to fix the errors. One simple baseline is to use these related frames with the same object visible and run the frame-to-frame registration and loop closure as we discussed in Section 5.2.1. However, due to huge viewpoint and lighting variation, usually there is no reliable registration that we can find, using SIFT or ICP [4]. Therefore, we desire to have a way to make use of the human annotation to impose strong constraints on the 3D reconstruction. However, these constraints should not be provided at a very low level, akin to clicking on keypoints provided by SIFT matches. This

[4]Otherwise, the automatic loop closing algorithm would have been able to detect this linkage.

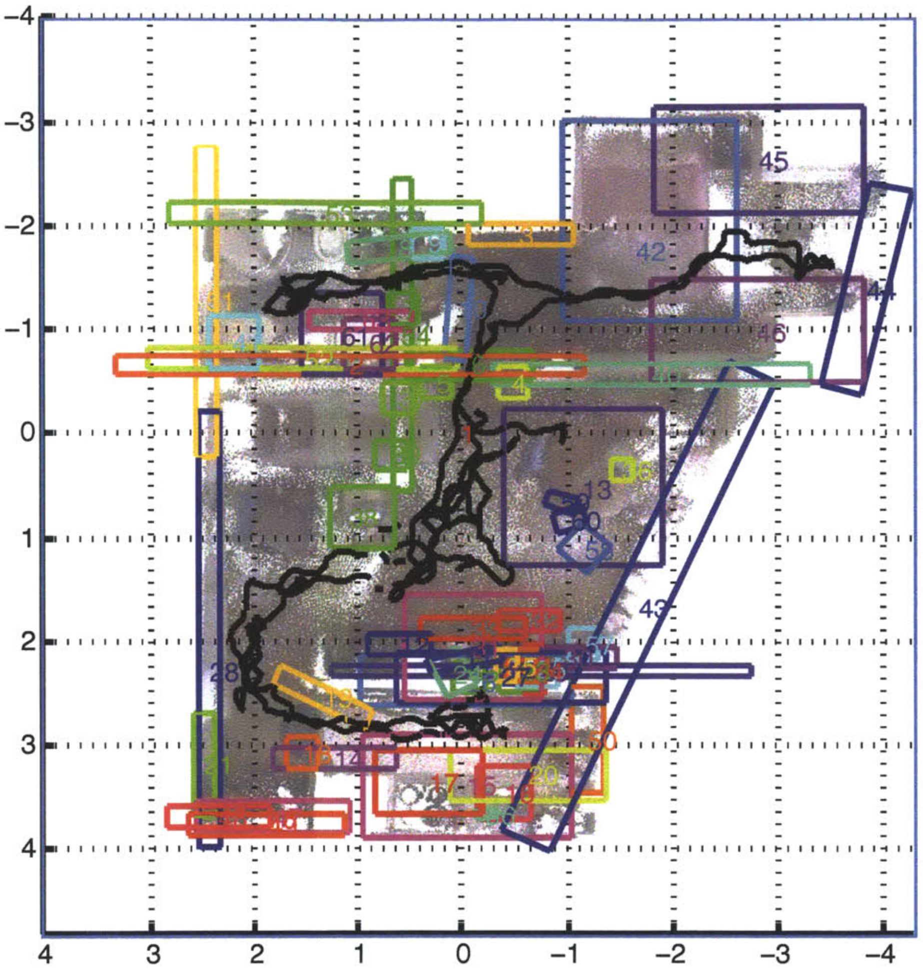


Figure 5-9: Constriained bundle adjustment: each box visualizes a constraint introduced by the user annotation.

would be intractable for a very long sequence. Instead, we desire a way to add constraints at the object level, to make use of the knowledge that an object in one frame is the same object in another frame.

Given that exact point-to-point correspondences are very hard to obtain either automatically and manually, we propose a novel approach to generalize traditional bundle adjustment with point-to-point correspondences into object-to-object correspondences. It starts from a very simple but amazingly powerful idea: we parameterize each object by its 3D location, rotation, and size of a 3D bounding box. As shown in Figure 5-8, the 3D point cloud of an object from a view should lie inside the 3D bounding box for the object. Because there is only one object instance, there is only one 3D bounding box, and all 3D

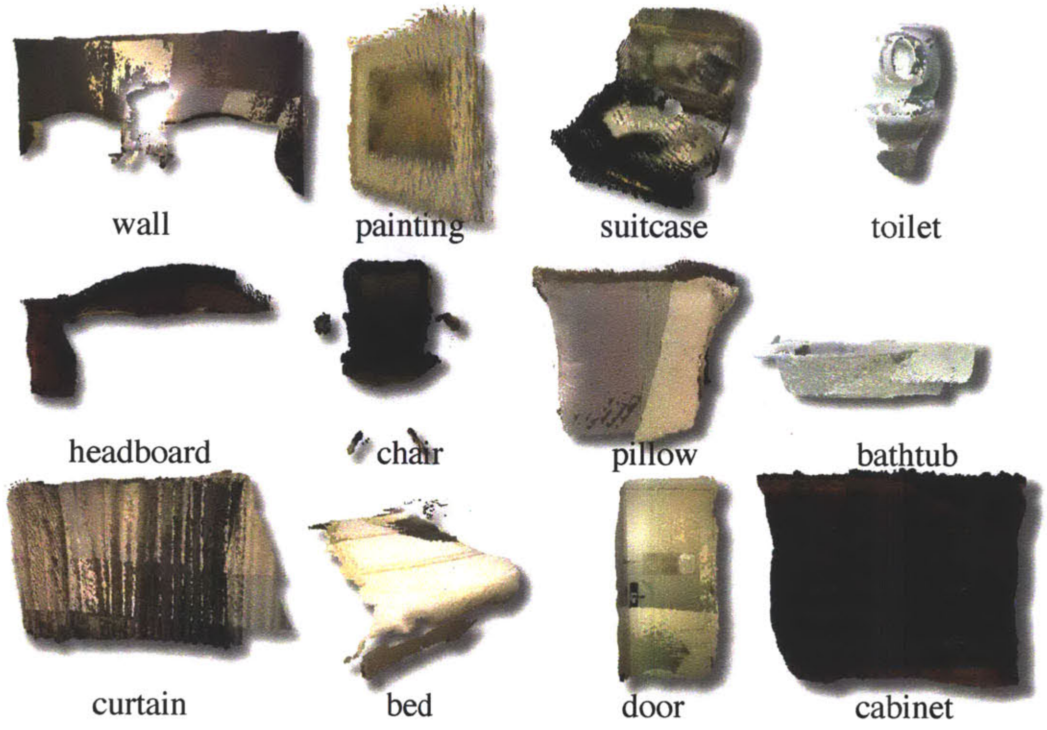


Figure 5-10: Object gallery: segmented object point cloud.

points from all views for the same object should lie inside the same 3D bounding box.

Therefore, when the same object appears at different location in the environment, its 3D bounding box with reasonable physical size cannot contain both. The bundle adjustment optimization will then pull them together close enough to fit in the same 3D bounding box. More specifically, for each object, the 6DOF location $\mathbf{t}_o$ and rotation $\mathbf{R}_o$ are unknown variables to be estimated by the bundle adjustment. The physical size $\mathbf{s}_o$ of the 3D bounding box is provided in the annotation process, either automatically suggested by the system based on the object category, or explicitly specified by the user. Encoding the object-to-object correspondences in 3D, together with the original bundle adjustment constraints based on point-to-point tracks, our new objective function is

$$\begin{aligned}\min \sum_{c} \sum_{p \in \mathrm{V}(c)} \left(\left\| \tilde{\mathbf{x}}_p^c - \mathbf{K} \left[\mathbf{R}_c | \mathbf{t}_c \right] \mathbf{X}_p \right\|^2 + \lambda_0 \left\| \tilde{\mathbf{X}}_p^c - \left[\mathbf{R}_c | \mathbf{t}_c \right] \mathbf{X}_p \right\|^2 \right) \\ + \lambda_1 \sum_{o} \sum_{c} \sum_{p \in \mathrm{L}(o,c)} \Psi\left(\left[\mathbf{R}_o | \mathbf{t}_o \right] \left[\mathbf{R}_c | \mathbf{t}_c \right]^{-1} \tilde{\mathbf{X}}_p^c, \mathbf{s}_o \right)^2,\end{aligned}$$

where

$$\Psi(\mathbf{X},\mathbf{s}) = \left\|\max\left(\mathbf{0},\mathbf{X}-\frac{\mathbf{s}}{2},-\mathbf{X}-\frac{\mathbf{s}}{2}\right)\right\|$$

is a loss function that has zero value inside a 3D cuboid with size $\mathbf{s}$, and goes linearly outside the cuboid. This means that given a 3D point $\tilde{\mathbf{X}}_p^c$ from the c-th camera, we transform it from local camera coordinate system to the world coordinate system using $[\mathbf{R}_c|\mathbf{t}_c]^{-1}$, and transform it from the world coordinate system to the object coordinate system using $[\mathbf{R}_o|\mathbf{t}_o]$, and see how far it is from the canonical 3D cuboid centered at the origin with size $\mathbf{s}$. This Ψ function is basically an extension of the quadratic loss function typically used for point-to-point correspondences.

5.4.1 Special semantics

The same framework can also be used to handle special types of objects. For example, we can make the bounding box with a very small size at one dimension, and it become a planarity constraint. If the sizes of the two dimensions are small, it is a line constraint. If the sizes of all three dimensions are small, it just becomes a point constraint in the traditional bundle adjustment. There are several types of objects with special semantic meanings. For "floor", we parameterize it as a flat plane on the plane $y = 0$. For many objects, we can parameterize it as an axis aligned object with the y axis (*i.e.* the gravity direction in our case). This reduces 2 DOF in the rotation matrix $\mathbf{R}_o$, and makes the optimization easier. This can be used to model both vertical surfaces (*e.g.* walls) and horizontal surfaces (*e.g.* ceiling, table top, counter top) as well. For mirrors and other highly reflective objects, we remove the points falling inside these image area, for both SIFT matching, and add no object constraints. For moving objects, such as doors, persons, and pets, we remove them as well. Of course, all the special operation can be overwritten by the user during the annotation process if certain assumptions don't hold.

5.4.2 Object context

Our generalized bundle adjustment can also model object-to-object relationship as well, given that each object is parameterized to be inside a bounding cuboid. For example, if one

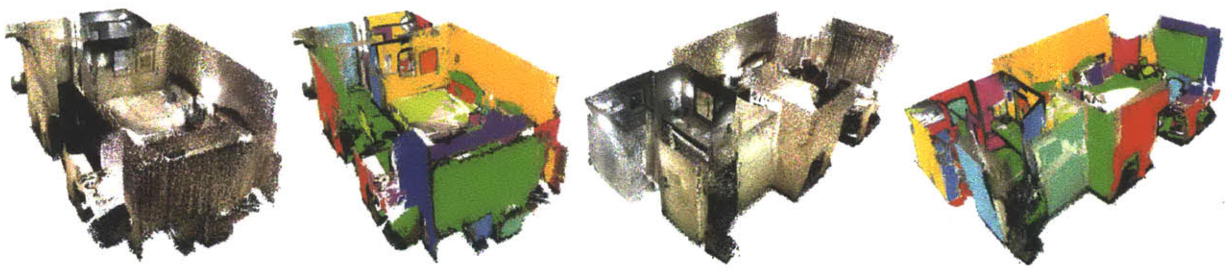
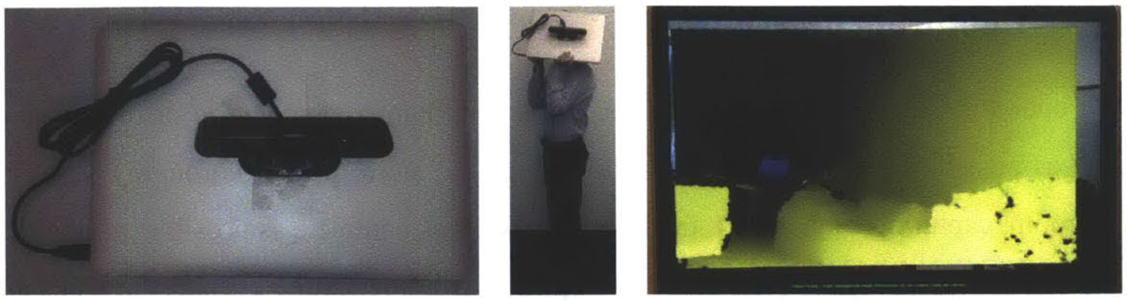


Figure 5-11: Annotation propagation and reconstruction correction result.

object is parallel with the other object, we can add hard constraints to let the two objects share the same rotation variables. If an object is attached on the other object, not only they are parallel, their location also add another constraint to be not too far away from each other, and the projection of the bounding box for the smaller object (*e.g.* a painting) should lie inside the bigger object (*e.g.* a wall). Orthogonality and other angle constraints can be naturally as a relationship between the rotation matrices of two objects. If two objects aligned at certain location, we can also add that as constrains on both the rotation and the translation. More usefully, for walls, we parameterize them as axis aligned object, and the user can also specified the direction of the walls explicitly. This also helps to rectify the point cloud to align in a meaningful direction for other applications such as object recognition and scene understanding to know the camera view angle relative to the gravity direction.

5.4.3 Optimization

We use Levenberg-Marquardt algorithm to optimize our objective function. To make the system flexible for adding various kinds of object and context constraints on the fly, we use automatic differentialization for the optimization using Ceres solver [3]. In our implementation, we use angle axis representation for general 3D rotation, and use the rotation angle for the axis-aligned rotation. The parameters λ_0 and λ_1 are set in the way that the effects are normalized to at similar scale with SIFT keypoint tracks and semantic constraints.



(a) Device (b) Capturing (c) Immediate Visual Feedback

Figure 5-12: Data capturing setup.

5.5 A place-centric 3D database

We introduce a 3D dataset of full environments scanned with a RGBD sensor. In order to build the database, we have exhaustively scanned each environment, walking through it mimicking human exploration, and thoroughly covering natural viewpoints. Our dataset consists of full 3D models of indoor places, as demonstrated in Figure 5-15.

5.5.1 Capturing setup

For the hardware, we mount an ASUS Xtion PRO LIVE sensor to a laptop (Figure 5-12). To make a database that closely matches the human visual experience, we want the RGBD video taken at a height and viewing angle similar to that of a human. Therefore, as shown in Figure 5-12, the operator carries the laptop with the Xtion sensor on his or her shoulder with a view angle roughly corresponding to a horizontal view slightly tilted towards the ground. During recording, the operator sees a fullscreen RGB image with registered depth superimposed in real-time to provide immediate visual feedback. We use OpenNI to record the video for both RGB and depth at 640×480 resolution with 30 frames per second. We use the default factory sensor calibration for the registration between the depth and image. We scan only indoor places, because the RGBD cameras don't work under direct sunlight.

5.5.2 Capturing procedure

Each operator is told to capture the video mimicking human exploration of the space as much as possible. They are told to walk through the entire place, thoroughly scanning each room, capturing each object, the floor, and walls as complete as possible. Guidelines include walking slowly, keeping the sensor upright, avoiding textureless shots of walls and floors, and walking carefully between rooms to avoid reconstructions with disconnected components.

5.5.3 Data-driven brute-force SFM

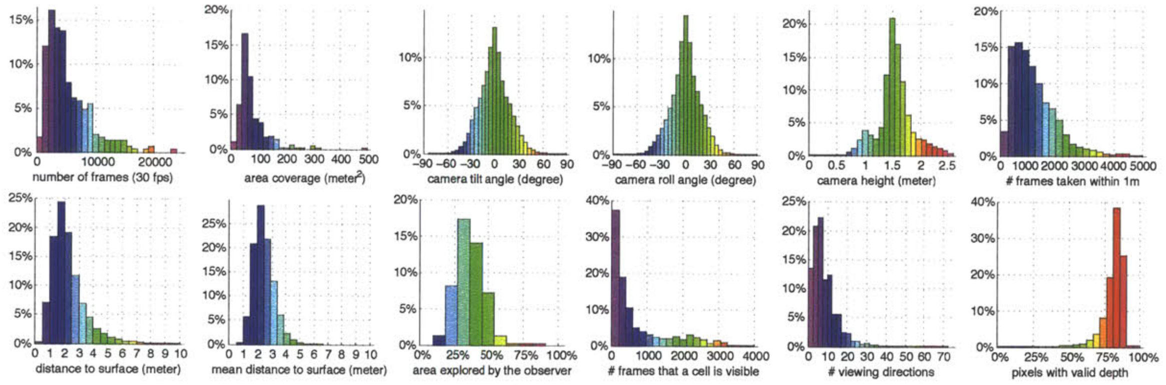
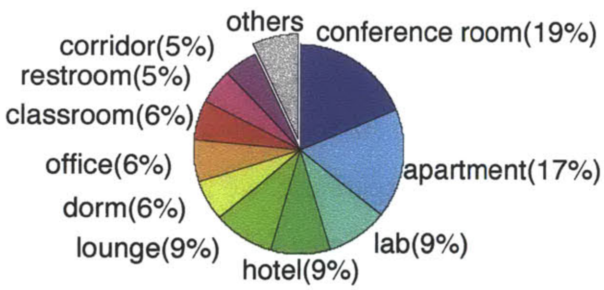
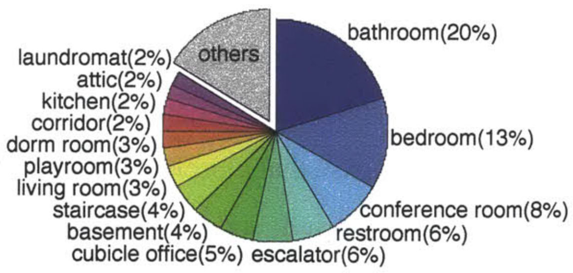


Figure 5-13: Dataset statistics.



Place category distribution.



View category distribution.

Figure 5-14: Semantic statistics of our database.

Drifting is a major source of errors, and loop closure is designed to address this issue. However, due to the great difficulty of view-invariance feature matching, even the state-of-the-art descriptors, are very poor in practical cases. Indoor scenes are also particularly difficult because of the low lighting condition and repetitive structures. As mentioned in

Section 5.2.1, although the automatic loop closing has very high precision, the recall is usually low. Given the fact the feature matching is not good enough, one way is to use a data-driven approach to structure from motion pose estimation. We can scan the same place for many passes, until almost the same viewpoints appear twice, and the distance between image descriptors are so small that the descriptors are working. In such a way, we can establish a lot of loop closing linkages and make the reconstruction much better. This idea is a natural extension of data-driven approach to vision problems, such as scene completion[65]. Therefore, during data capturing of our dataset, we on purposely make many passes inside an environment to make the reconstruction better. In the meantime, this also increase the amount of frames the same object is observed under slightly different viewpoints, which could be useful for object detection.

5.5.4 Dataset analysis

We have 389 sequences captured for 254 different places, in 41 different buildings. Operators capture some places for multiple passes, at different times when possible. We manually group them into 16 place categories. Some statistics is shown in Figure 5-14. Our dataset will be publicly available, offering image frames and registered depth maps, camera poses (and intrinsic camera parameters), segmented 3D point cloud for the whole scene with colors, object segmentation mask for each frame, place category for the whole scene, as well as 3D point clouds with color for each object.

5.6 Integrating category recognition over space

To make the distinction between view-based and place-centric scene representations, and to study their relationship, we want to visualize which views are likely to appear in which places and at what locations.

View-based scene classifier We manually select 54 view-based scene categories from the SUN database [184] corresponding to indoor scenes that we expect to see in our database. We take all the images from the latest version of the SUN database that fall into the chosen

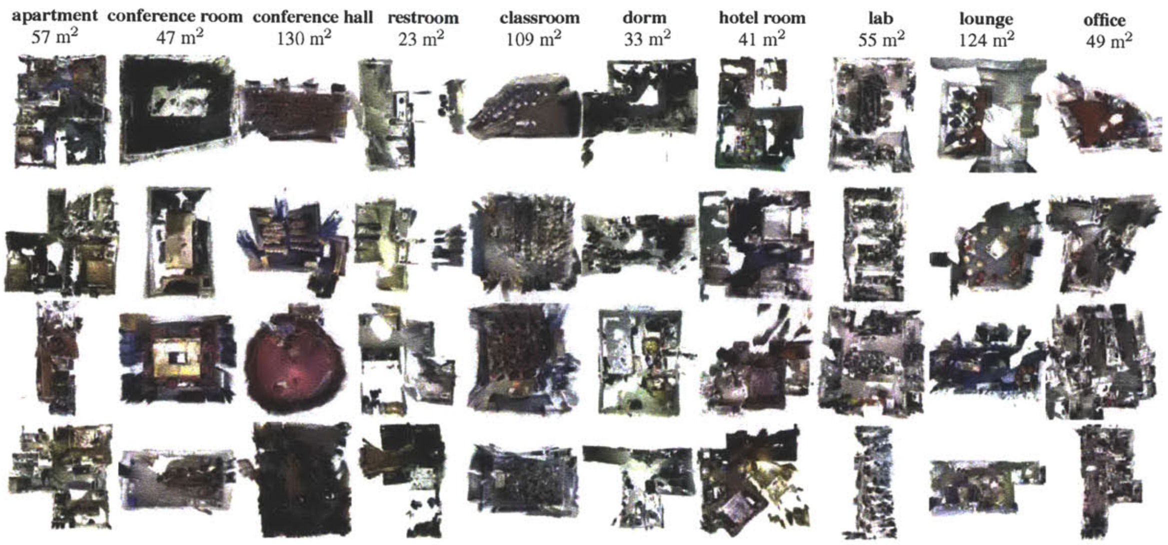


Figure 5-15: Each column contains examples from each place category. The numbers are the median example areas for each place category.

categories and use object bank features [109] to train a one-vs-rest linear SVM classifier. Then, for each frame in a sequence, we run the trained classifier to obtain a score with 54 dimensions for the view-based scene category of the frame.

View-category map To obtain a map of the view category distributed over the space, we define a grid on the floor with cells of size 0.1×0.1 meter^2. We project the 3D points from each view onto the floor grid to cast a vote, and accumulate the 54 dimensional score vector on each grid node to get the sum of the responses for each category voted from different frames. We define a Markov Random Field (MRF) on the grid with neighborhood of size 8, and let the label space to be the view-based scene category, where each node has a unary term to indicate how likely it is to belong to a particular view category. To encourage the spatial smoothness of the view-category map, we use the Potts binary cost as the pairwise potential and optimize the MRF using [20]. Some example results are shown in Figure 5-16. While the result shows some interesting structure, view categorization on our dataset is still very challenging. This is likely due to the fact that the training SUN database images [184] are from Internet photo collections, which do not necessarily resemble real-world navigation experience. Photos found on the Internet are often shot from very restricted canonical viewpoints in brightly-lit rooms with almost zero camera tilt, whereas our scans

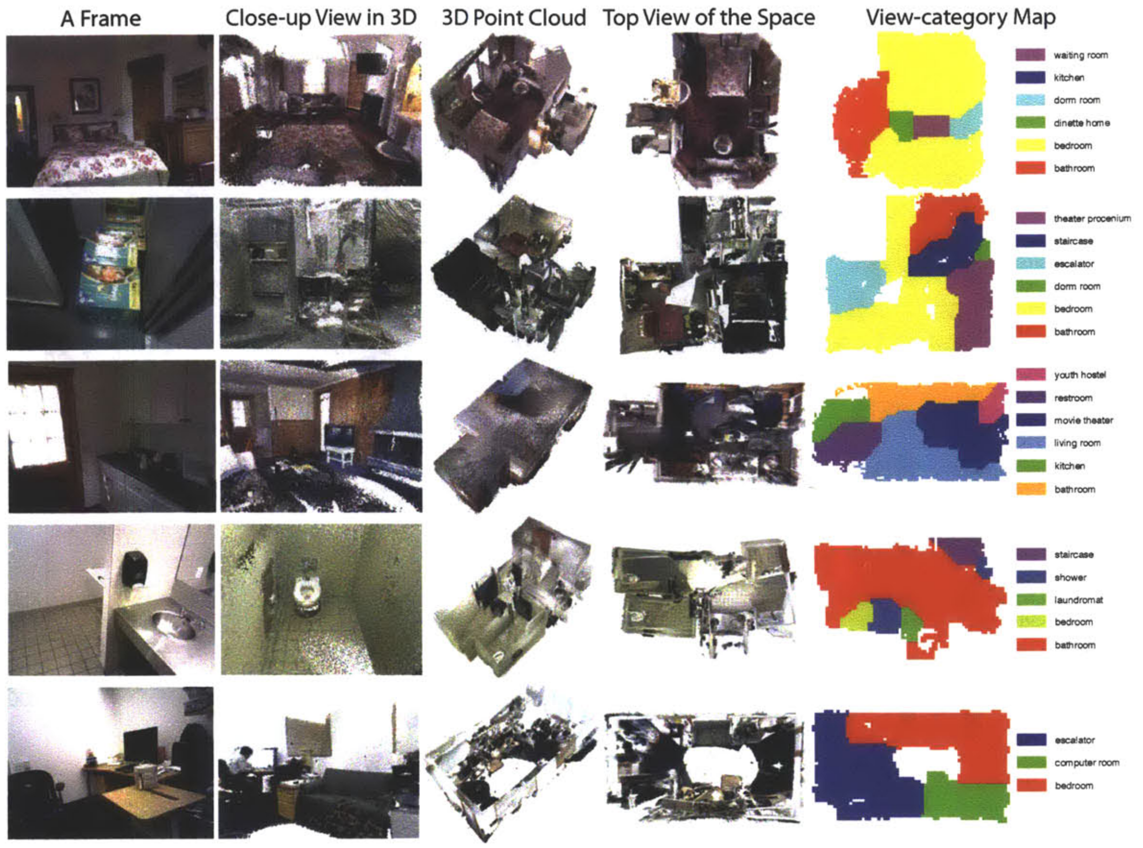


Figure 5-16: Examples of view category maps for the full places.

are much darker with significant camera rotation.

View-view and view-place relation There has been some work studying the relationship between view-based scene and sub-scenes [195], as well as the views inside a 2D panoramic place representation [187]. However, these works can only study the view-view and view-place relationship in 2D image space. Now, with our dataset, we can study the relationship between view and place in space. Figure 5-14 shows the distribution of view categories and place categories in our database, obtained by counting the number of grid cells belonging to each view, and the number sequences belong to each place. Figure 5-17 shows the co-occurrence pattern of neighboring nodes in the space for different views, as well as the co-occurrence pattern of the view categories and place categories, by counting the number of grid cells with view labels inside each place category. Note that the view category is obtained from the prediction of the trained classifier with the MRF, and the place category is labeled manually.

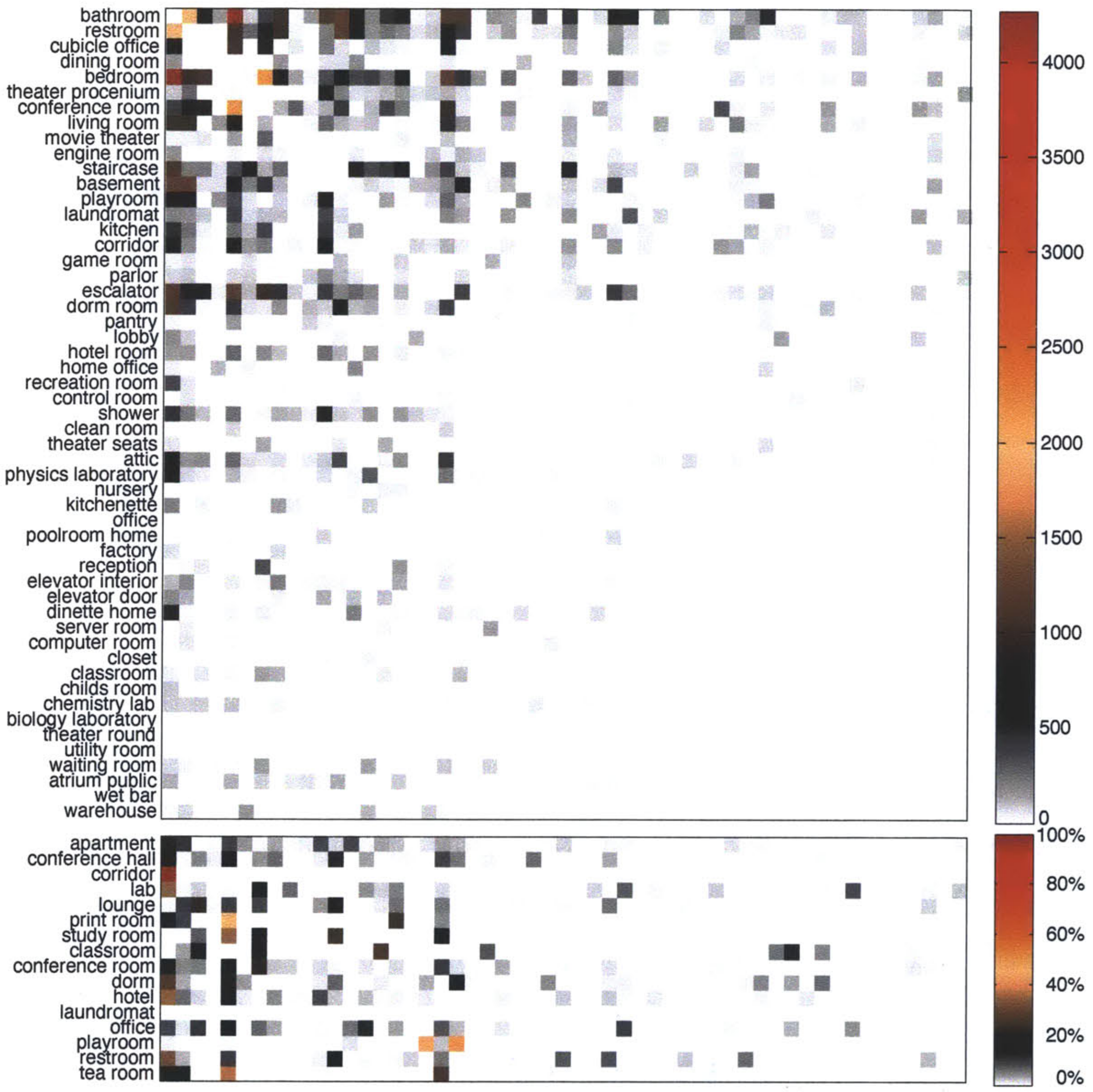


Figure 5-17: Relationship between place category (manually labeled) and view category (automatically predicted). Columns correspond to the view categories.

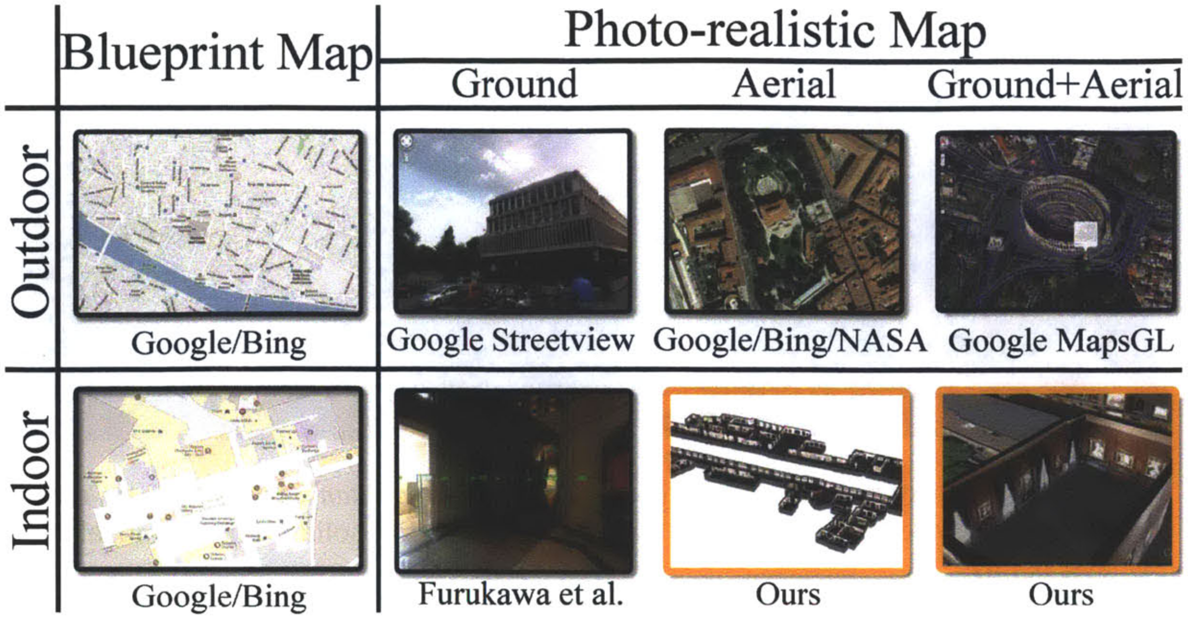


Figure 5-18: Categorization of different map visualization for indoor and outdoor environments. While major efforts have been made in the past for outdoor environments, indoor scenes, in particular, their photo-realistic visualization, is still in its early stage. This section enables the photo-realistic visualization of indoor scenes from aerial viewpoints, which can be nicely integrated with the conventional ground-based indoor navigation experiences.

5.7 Application: photo-realistic indoor map

Digital photography has gone through a revolution in the past decade, where an ever-growing number of photographs are acquired everyday all over the world and shared online. The abundance of photographic data combined with the growing interests in location-aware applications make 3D reconstruction and visualization of architectural scenes an increasingly important research problem with large-scale efforts underway at a global scale. For example, Google MapsGL seamlessly integrates a variety of outdoor photographic content ranging from satellite, aerial and street-side imagery to community photographs. Indoor environments have also been active targets of photographic data capture with increasing business demands. For instance, the Google Art Project allows exploration of museums all over the world as well as close examination of hundreds of artworks photographed at high resolution. Google Maps and Bing Maps serve stunning panorama images of indoor businesses such as grocery stores and restaurants.

However, unlike outdoor environments offering imagery from both aerial and ground-

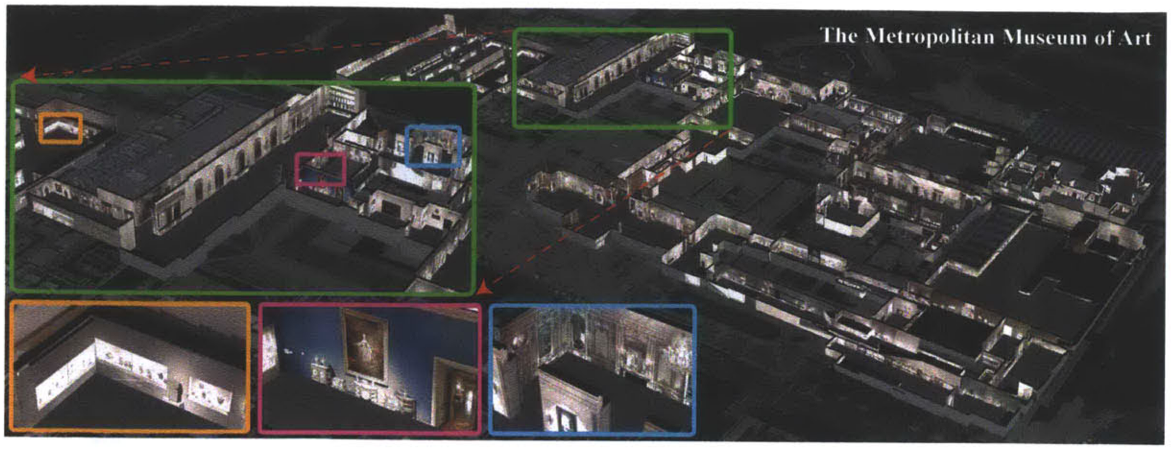


Figure 5-19: A texture-mapped 3D model of *The Metropolitan Museum of Art* reconstructed by our system, which directly recovers a constructive solid geometry model from 3D laser points.

level viewpoints, indoor scene visualization has so far been restricted to ground-level viewpoints, simply because it is impossible to take pictures of indoor environments from aerial viewpoints (See Fig. 5-18). The lack of aerial views hampers effective navigation due to the limited visibility from the ground level, especially for large-scale environments, such as museums and shopping malls. Without a photorealistic overview, users can easily get lost or confused during navigation.

This section presents a 3D reconstruction and visualization system for large indoor environments, in particular museums (See Fig. 5-19). Our system takes registered ground-level imagery and laser-scanned 3D points as input and automatically produces a 3D model with high-quality texture, under the assumption of piecewise planar structure. Our system enables users to easily browse a museum, locate specific pieces of art, fly into a place of interest, view immersive ground-level panorama views, and zoom out again, all with seamless 3D transitions (demos and videos available at [167]). The technical contribution is the automated construction of a Constructive Solid Geometry (CSG) model of indoor environments, consisting of volumetric primitives. The proposed "Inverse CSG" algorithm produces compact and regularized 3D models, enabling photorealistic aerial rendering of indoor scenes. Our system is scalable and can cope with severe registration errors in the input data, which is a common problem for large-scale laser scanning (Fig. 5-20).

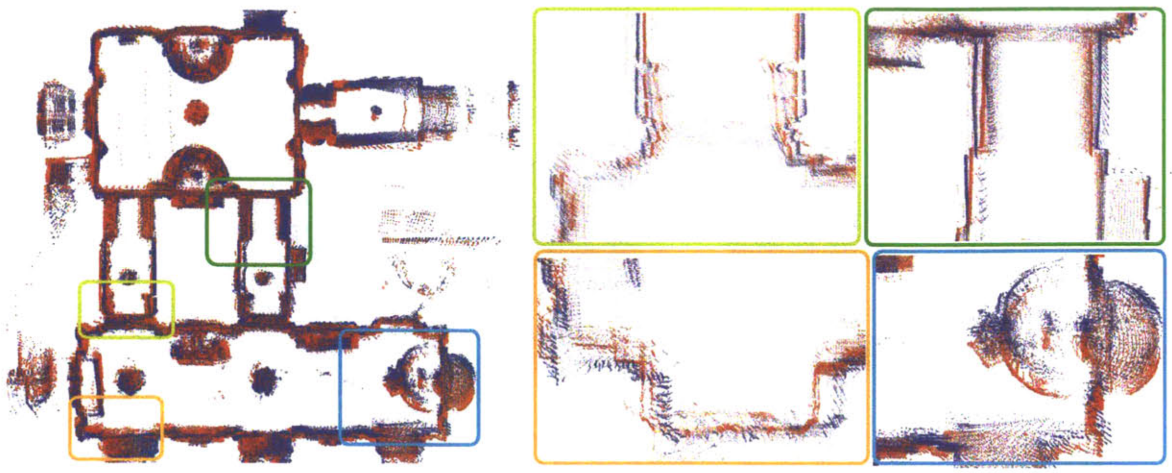


Figure 5-20: A top-down view of input laser points, where two different colors represent two vertical laser range sensors. A challenge is the presence of severe noise in the input laser points. Note that errors are highly structured and "double walls" or even "triple walls" are prevalent.

Related work

Laser range sensors have been popular tools for 3D reconstruction, but the major focus of these approaches has been limited to the reconstruction of small scale indoor environments, objects, and outdoor buildings [26, 110], or the analysis of 3D structure such as the detection of symmetry or moldings [129, 139]. A human-operated backpack system [111, 146] was proposed to produce 3D texture-mapped models automatically, but their results usually have relatively simple structures of the final models. As pointed out in [111], a major challenge for data acquisition is the precise pose estimation for both imagery and 3D sensor data, which is critical for producing clean 3D models with high-quality texture images requiring pixel-level accuracy. Although there exist scalable and precise image-based pose estimation systems based on Structure from Motion algorithms [2, 4], our data collection exposes further challenges. First, indoor scenes are full of textureless walls and require complicated visibility analysis because of narrow doorways. Furthermore, museums often have rooms full of non-diffuse objects (e.g., glass and metallic materials), which violates assumptions of vision-based systems. Active depth sensing usually yields more robust and accurate pose estimation [71, 82]. However, even with high-quality laser scans, pose estimation is still very challenging, as museum floors are often uneven (e.g., floors made of

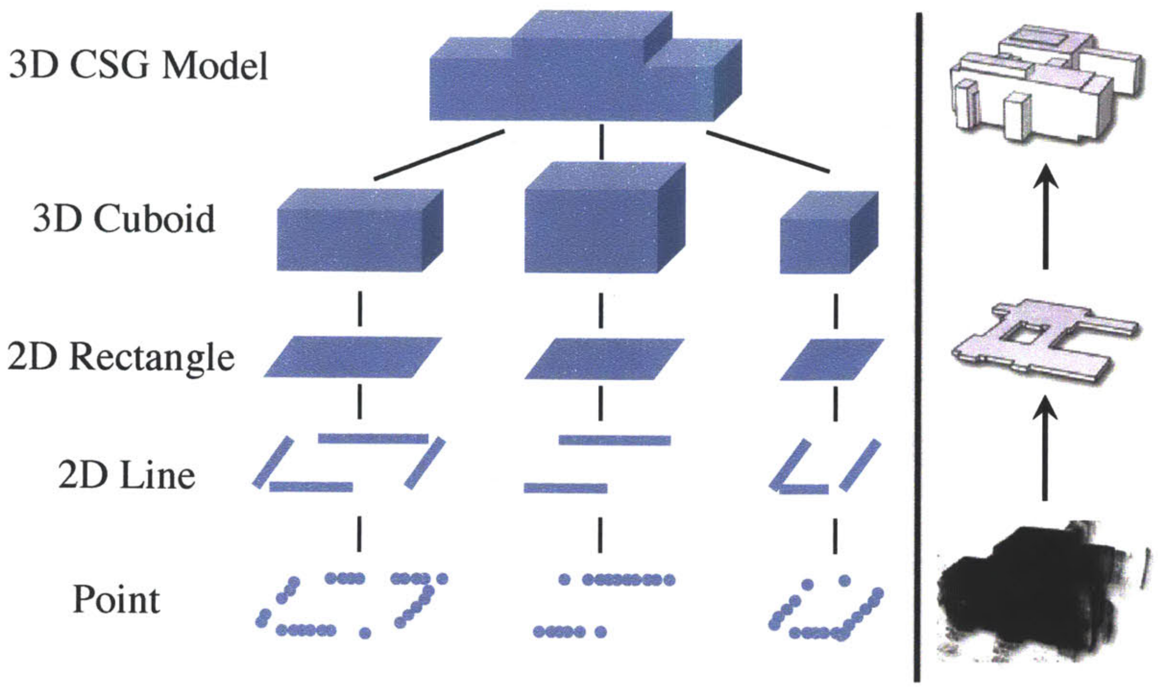


Figure 5-21: 3D CSG construction happens step by step in a bottom-up manner, from laser points, 2D lines, 2D rectangles, 3D cuboids to final 3D CSG models.

stones) causing laser range sensors to vibrate constantly. Furthermore, simply the scale of the problem is unprecedented in our experiments (more than 40,000 images and a few hundred million 3D points for the largest collection), where robustness to registration errors becomes an inevitable challenge to overcome.

In image-based 3D modeling, surface reconstruction is usually formulated as a volumetric segmentation problem via Markov Random Field (MRF) defined over a voxel grid [55, 72]. However, an MRF imposes only local regularization over pairs of neighboring voxels, which are susceptible to highly structured noise in laser points. Also, these approaches typically penalize the surface area which makes it difficult to reconstruct thin structures such as walls, leading to missing structures or reconstruction holes [55]. The requirement of having a voxel grid limits the scalability of these approaches due to memory limitation. Another popular approach is to impose strong regularity [164, 189, 193] by fitting simple geometric primitives to the point cloud data. However, most approaches have focused on modeling outdoor buildings where the structures are much simpler than indoor scenes. Due to these challenges, large-scale indoor scenes are usually modeled by

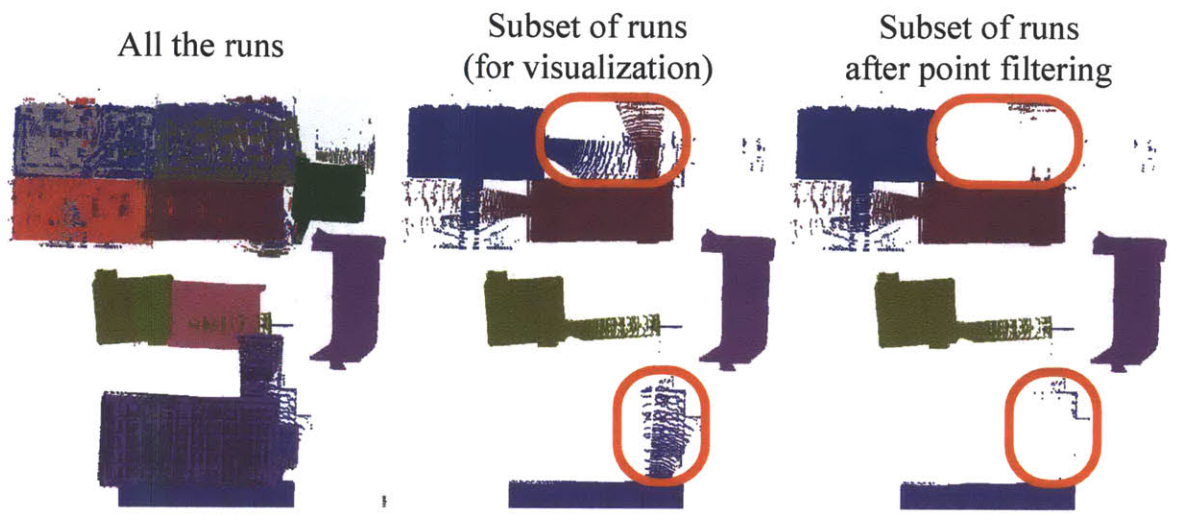

All the runs
Subset of runs (for visualization)
Subset of runs after point filtering

Figure 5-22: Laser range sensors sometimes scan for a long distance, for example through doorways, which unfortunately yields sparse and noisy 3D points. The point filtering step removes such unwanted noisy points (highlighted in red).

hand in CAD software [10]. An alternative option is to extrude walls from floorplans (i.e. extend the 2D floor plan vertically). However, it is not feasible in practice for our targeted museums, as accurate floor plans do not exist for many museums. Even where they exist, floorplans may come in different styles or formats, which makes it difficult to automatically parse such images. Even if walls are extracted, they need to be aligned with images for texture mapping, which involves a challenging image-to-model matching problem at a massive scale. Single-view 3D reconstruction techniques [61, 194] are also popular for architectural scenes, where priors and sophisticated regularization techniques play an important role, which is common in this section. However, their focus is more on the analysis of 3D space, while our primary interest is scene visualization that requires higher quality 3D models.

For indoor scene visualization, view-dependent texture mapping is typically used to provide interactive scene exploration [55, 176]. For example, Google Art Project, Google Maps, and Bing Maps provide immersive panorama-based viewing experiences for museums and indoor businesses. However, unlike in outdoor maps, in all these systems, navigation is restricted to ground-level viewpoints, where photographs can be directly acquired. As a result, there is no effective scene navigation due to the lack of more global overview from aerial viewpoints. Our goal in this work is to produce 3D models that enable aerial

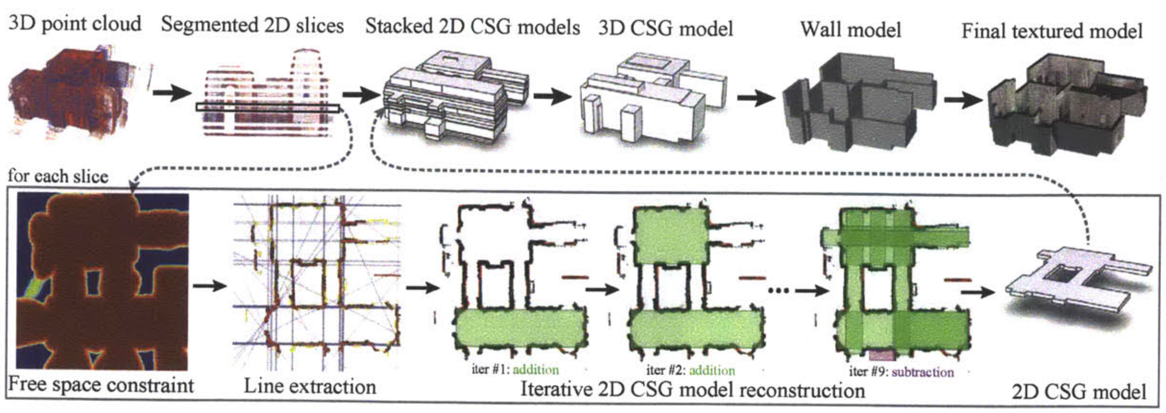


Figure 5-23: System pipeline. The first row shows the entire pipeline of our system, and the second row shows the detailed pipeline of "Inverse CSG" reconstruction of a 2D horizontal slice.

rendering of indoor environments using ground-level imagery.

5.7.1 Data collection and preprocessing

A hand trolley is used to capture input data in our system. It is equipped with a rig of cameras and three linear laser range sensors, two of which scan vertically at the left and the right, while the other sensor scans horizontally. For operational convenience, data collection for a large museum is performed over multiple sessions. We use the term *run* to refer to the data collected from a single session. We use the horizontal laser scans to estimate sensor poses for all runs together using [82], and we handle each floor separately. We use the vertical ones for 3D model reconstruction, since they provide good coverage in general. We also estimate a surface normal at each laser point by fitting a local plane to nearby points.

Laser range sensors on our trolley can scan for long distances, for example, rooms next door through narrow door ways (See Fig. 5-22). This causes a problem, because the same room can be scanned from multiple runs, where laser points from nearby runs are dense and accurate, while those from far runs become sparse and possibly inaccurate. Therefore, we only keep laser points from nearby runs at overlapping regions. More concretely, given laser points from all the runs, we project them onto a horizontal plane, compute their bounding box, and overlay a grid of cells inside the bounding box, where the size of a cell is set to be 0.1 meters. At each cell, we identify $\alpha(=2)$ runs that have the most number of points in the cell, and filter out all the points that belong to the other runs. Note that this filtering procedure is not perfect and may leave unwanted points. However, our reconstruction algorithm is robust, and the main purpose of this step is to remove the majority of the unwanted point clouds.

5.7.2 Inverse CSG for large-scale indoor modeling

While many 3D reconstruction algorithms are designed to recover a surface model, we aim at reconstructing a volumetric representation of a scene from registered laser scan. In particular, we model the visible *free space* (surface exterior space), which is directly measured by laser range sensors, as opposed to the invisible *filled-in space*, which is blocked by walls, floors and ceilings. For simplicity, we assume that the space outside buildings is filled-in, which makes the free-space a well-bounded volume.

We reconstruct the free space volume as a Constructive Solid Geometry (CSG) model, which is an expression of simple geometric primitives with union and difference operations. We choose cuboids as volumetric primitives, as they are the most common shapes in architecture design, and good approximations for others. We restrict cuboids to be aligned with the vertical (gravity) direction but allow arbitrary horizontal orientations, which is more general than the Manhattan-world assumption [54, 55]. The use of volumetric primitives and the CSG representation allows us to impose powerful architectural regularization and recover compact 3D models, which is a key factor for large-scale reconstruction and visualization. Our strategy is to enumerate primitive candidates, then construct a CSG model out of the generated primitives to best describe the input laser information. Instead of directly solving for a CSG model with 3D volumetric primitives, where the number of primitive candidates becomes prohibitively large, we (1) split the 3D space into a set of horizontal slices, each of which shares the same horizontal structure (i.e., a floor plan structure in the slice); (2) extract line segments from the input point clouds in a horizontal slide, which are used to form rectangle primitive candidates; (3) solve a 2D CSG model with rectangle primitives in each slice; (4) generate 3D primitives based on the 2D reconstructions, then

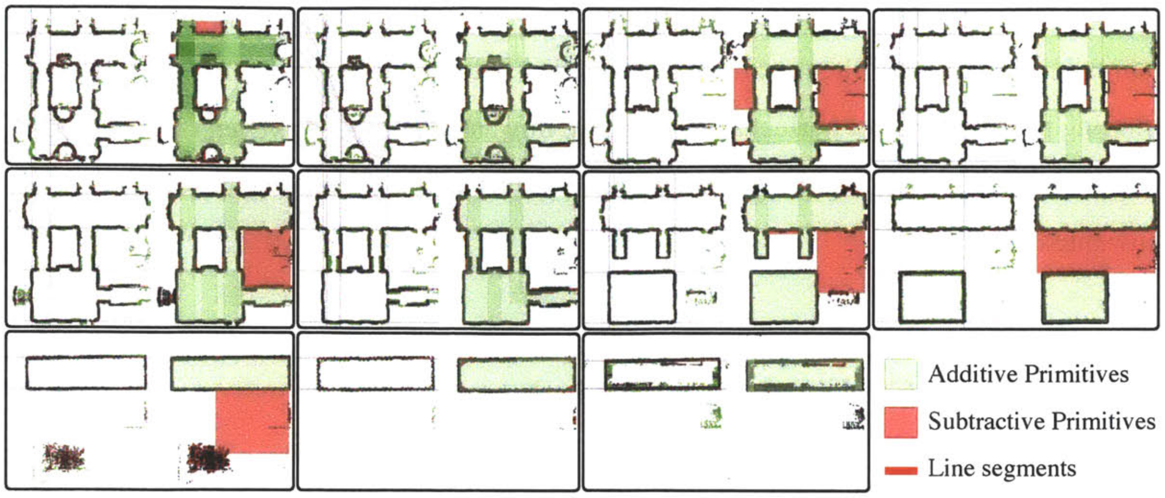


Figure 5-24: 2D CSG reconstructions for the eleven horizontal slices for one run in Met. For each slice, the figure shows the 3D point cloud and extracted line segments (in red) at the left, and reconstructed 2D CSG model at the right, where green (resp. red) rectangles are additive (resp. subtractive) primitives.

solve for a 3D CSG model (See Fig. 5-21). We now detail in these steps.

Slice extraction and 2D primitive generation

A floor plan structure on a horizontal slice changes at the existence of horizontal surfaces. For example, in a room with two different ceiling heights, the structure of the horizontal floor plan changes at the two ceilings. We compute the histogram of the number of 3D points in the gravity direction, and convolve it with a Gaussian smoothing operator with a standard deviation equal to 0.5m. We identify peaks in the histogram to identify dominant horizontal surfaces, which divides the 3D space into 2D slices.

Illustrated in the bottom row of Fig. 5-23, for each slice, we project laser points within the slice onto a 2D horizontal plane, and extract line segments passing through them by Hough transformation [80]. These line segments are used to enumerate rectangle candidates in several ways (Fig. 5-25).

First, a rectangle is generated from every tuple of four line segments forming appropriate angles, where 5° error is allowed (also for the remaining cases). Second, every triple of segments forming a "⊏" shape generates four rectangles, where the position of the missing edge is at one of the four endpoints of the two side line segments. Third, every pair of parallel segments generates $\binom{4}{2} = 6$ rectangles, where the positions of the two missing edges are chosen out of the four endpoints of the two line segments. Lastly, every pair of perpendicular line segments is used to generate $\binom{3}{2} \times \binom{3}{2} = 9$ rectangles, where two horizontal (resp. vertical) positions are determined out of the three possible positions (two endpoints of the parallel line segment and the intersection of the two line segments).

We generate an over-complete set of primitives, because line segment extraction may not be perfect, and we do not want to miss any important rectangles. In order to speed up the following reconstruction step, we prune out primitive candidates that are unlikely to be part of the final model, based on the following three criteria. First, any small rectangle, whose width or height is less than 0.15 meters, is removed. Second, identical rectangles are removed except for one copy, where two rectangles are defined to be identical if the distance between their centers is less than 0.3 meters, and the difference of each dimension (width and height) is less than 0.2 meters. Third, a rectangle without enough *supporting* laser points is removed, that is, if the number of laser points that are within 0.2 meters from the boundary of the rectangle is less than 0.05 times the number of laser points in the slice. For a typical run, the algorithm extracts about 50 to 100 line segments per slice, and generates nearly a million primitives initially, which are reduced to a few hundred thousand after pruning.

Reconstructing 2D CSG models

We aim to construct a CSG model T that best describes the laser information. The solution space is exponential in the number of hypotheses, and we propose a simple algorithm that greedily adds or subtracts a primitive. Let $E(T)$ denote an objective function that measures how well T explains the laser information. We start from an empty model. In each iteration, we try to add or subtract each primitive to or from the existing model, evaluate $E(T)$ for each result, and choose the best one to be the new model. The algorithm iterates until $E(T)$ does not increase by more than a threshold ε, which is set to 0.02. We now give the definition of $E(T)$ in the rest of this section.

The laser scan provides not only a 3D point cloud, but also the information that nothing exists between the laser center and the scanned 3D point. This *free-space score* is calculated

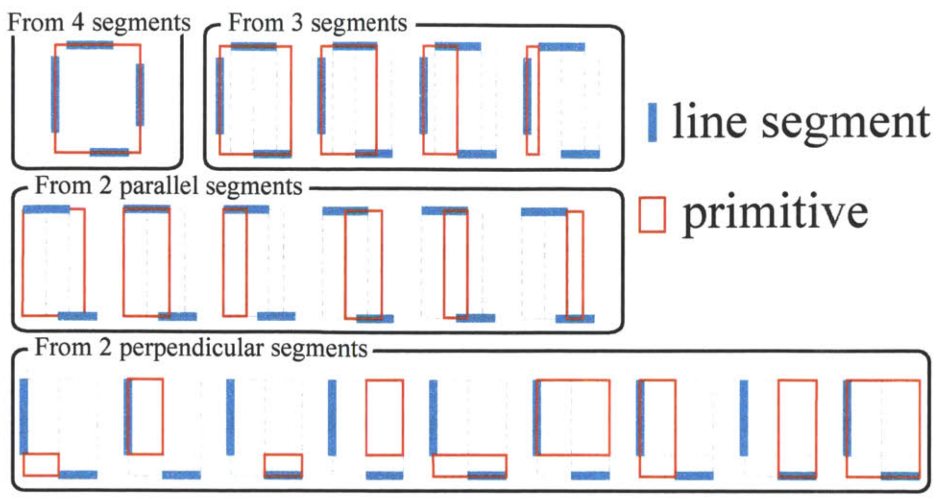


Figure 5-25: Line segments are used to form rectangle primitives. Depending on the number of available line segments, different number of rectangle primitives are generated.

on a grid of voxels in 3D. First, we compute the axis-aligned bounding box of all the laser points, while ignoring the extremal 2% of points in each direction to avoid noise. The voxel size is set to 0.4 meters[5]. For each voxel, we count the number of laser lines passing through it (i.e., a line segment connecting the laser center and the scanned 3D point). A voxel with more counts is more likely to be in the free space, and hence, the free-space score is set to be the number of non-zero count voxels. We truncate the score to be no more than fifteen to avoid bias towards the voxels near the laser centers. Zero-count voxels, in turn, are unlikely to be in the free-space, because laser range sensors usually scan an entire free-space (the trolley operators are instructed to do so). Therefore, we assign a negative free-space score for zero-count voxels. More precisely, for each voxel with zero count, a distance transform algorithm [51] is used to compute the distance to the closest positive-count voxel, and its negated distance (in voxel units) multiplied by 30 is set as the free-space score. The use of distance transform is important as free-space scores tend to become noisy at surface boundaries. To obtain free-space scores for each 2D slice, we simply take the average score value inside the slice vertically.

The objective function $E(T)$ consists of the following three terms. The first term mea-

[5]Different from standard volumetric reconstruction algorithms [55, 72], voxel resolution is not critical for accuracy in our approach, as precise surface positions are determined by primitives.

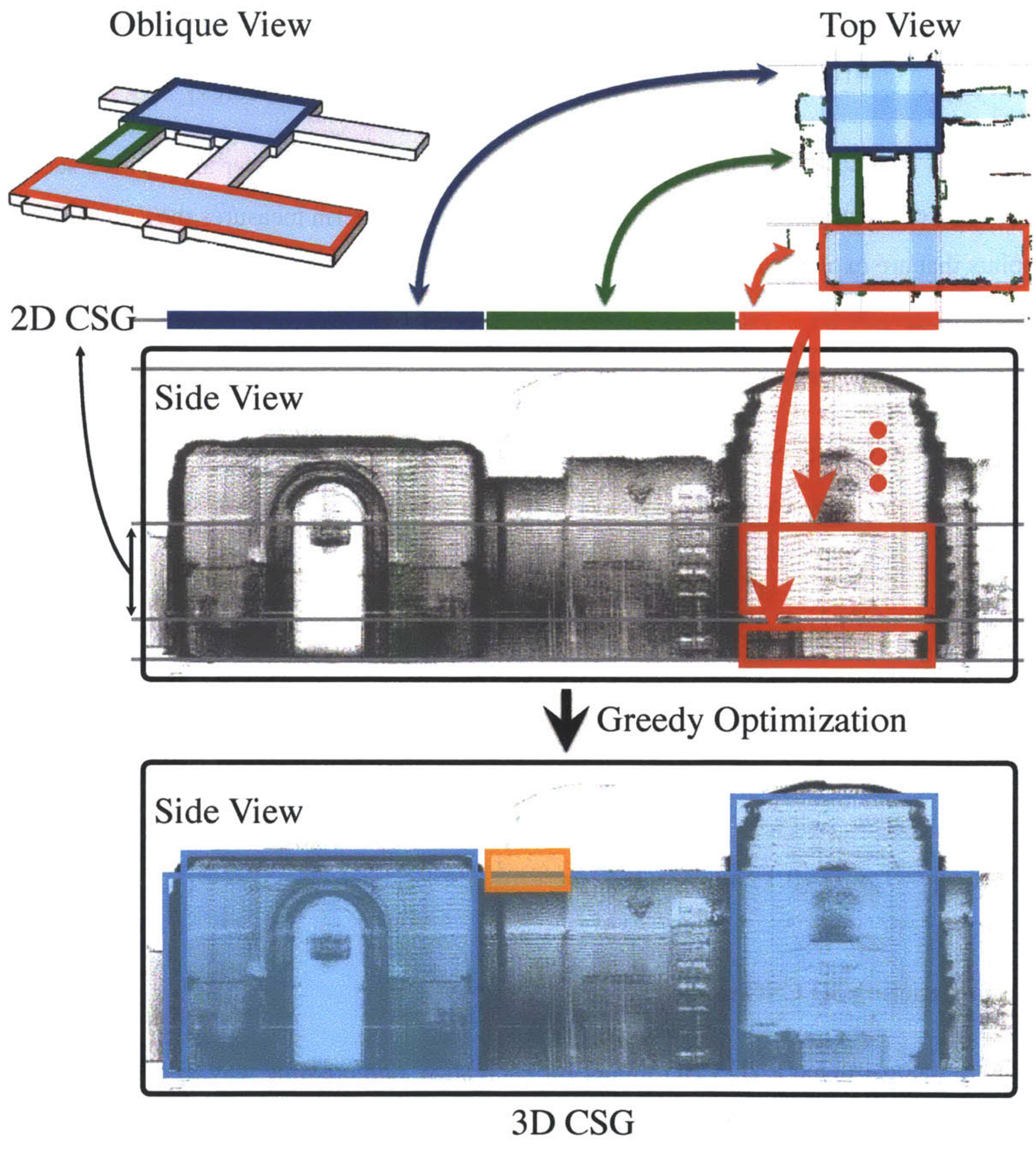


Figure 5-26: From 2D CSG to 3D CSG. The first row contains the oblique and the top views of the 2D CSG model reconstructed in a horizontal slice, which consists of 2D rectangles. In the second row, each 2D rectangle is inflated to generate 3D cuboid primitives, where the top and the bottom faces are determined by using all possible pairs of horizontal slice boundaries. The figure shows two such cuboid primitive examples, which are generated from the red rectangle. In the third row, a pool of cuboid primitives are used to construct 3D CSG model, where we allow subtraction operations as in the 2D case, and a subtractive primitive is illustrated in orange.

sures how much free space information is explained by T:

$$E_1(T) = \frac{\{\text{Sum of free-space scores inside } T\}}{\{\text{Total sum in the domain without negative scores}\}}.$$

Ideally, if the shape T explains the free space perfectly, it would cover all positive scores and none of the negative ones, then $E_1(T) = 1$. The second term measures the ratio of laser points that are explained by T:

$$E_2(T) = \frac{\{\text{\# of points on the surface of } T\}}{\{\text{total \# of points}\}}.$$

However, this term encourages a complex model that explains all the 3D points. Therefore, the third term measures the quality of T from laser points for regularization:

$$E_3(T) = \frac{\{\text{perimeter of } T \text{ near laser points (within 0.2 meters)}\}}{\{\text{total perimeter of } T\}}.$$

The overall objective function $E(T)$ is defined to be a weighted sum of the three terms:

$$E(T) = w_1E_1(T) + w_2E_2(T) + w_3E_3(T), \tag{5.1}$$

where $w_1 = 1$, $w_2 = 0.1$ and $w_3 = 0.4$.

Reconstructing 3D CSG model

Having reconstructed a 2D CSG model for each slice, one possible way to generate a 3D CSG model is to extrude each 2D CSG model as thick as the corresponding horizontal slice, and stack them up. However, this solution tends to create jagged, misaligned models (e.g. the first row of Fig. 5-27), as it does not integrate evidence across multiple slices and thus is sensitive to laser scan errors. To address this issue, we propose a 3D Inverse CSG algorithm to generate a 3D model, as shown in Fig. 5-26. We follow the same two-step InverseCSG algorithm for the 2D slice reconstruction, and extend it to handle 3D reconstruction: first to generate a pool of primitives (3D cuboids), and then use our greedy algorithm to choose a subset of primitives from the pool to represent the final complete shape.

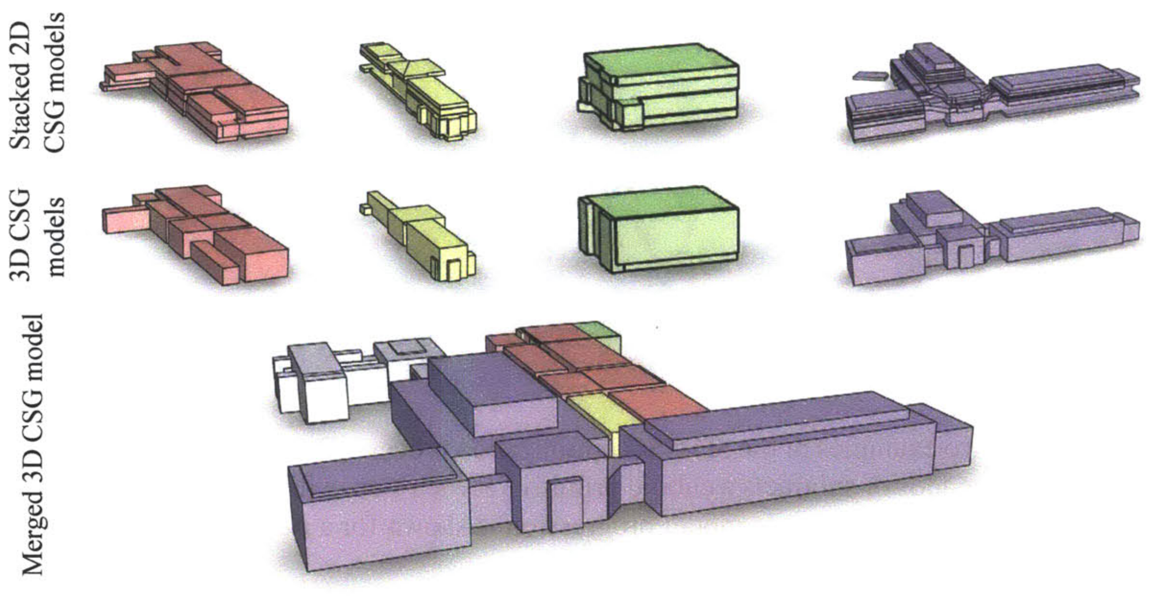


Figure 5-27: Reconstruction results for the Frick Collection. Top: Stacked 2D CSG models for individual runs. Middle: Corresponding 3D CSG models. Bottom: The final merged 3D CSG model. The white model is the example illustrated in Fig. 5-23. Note that there exist three more runs for this museum, which are not visible here because of overlapping or occlusion in rendering.

For the first step, for each 2D primitive in the 2D CSG result, we generate multiple candidate 3D primitives: the vertical positions of their top and bottom faces are chosen from every pair of slice boundary positions (the top and bottom faces have the same shape as the 2D primitive – only their vertical positions vary). Therefore, let N denote the number of the horizontal slice boundaries. Each 2D rectangle generates $\binom{N}{2}$ cuboid primitives. Then, we define the same objective function as before in Eq. 5.1, which can be naturally generalized to 3D, and use the same greedy algorithm as in Sec. 5.7.2 to "redo" everything with 3D cuboids as primitives. After reconstructing a 3D CSG model for each run[6], we simply take the union of all the models in the CSG representation to create the final merged model, and convert it into a mesh model by CGAL [24].

[6]Runs are processed independently for computational and memory efficiency. If needed, a long run can be split into shorter ones, where resulting CSG models can be merged in constant time.

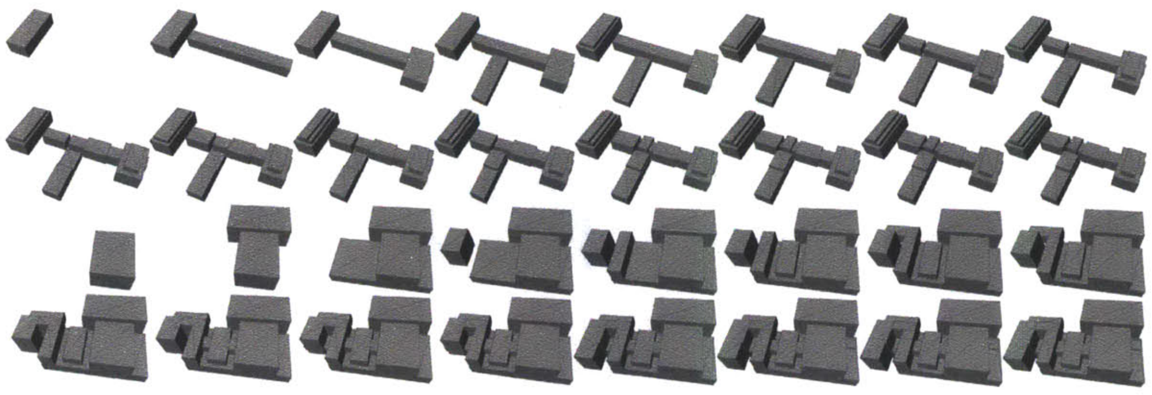


Figure 5-28: Two examples of the 3D CSG optimization process. Our 3D CSG construction algorithm either adds or subtracts a cuboid primitive at each step, while greedily optimizing the objective function. Example model progression is shown for a run in National Gallery and Met, respectively.

5.7.3 Indoor scene visualization

Our goal in visualization is to display an entire museum from aerial viewpoints for effective navigation. However, a reconstructed CSG model is not yet suitable for the task, because ceilings and walls occlude interiors, and the walls have no thickness (i.e., a paper-thin model), which look unrealistic from an aerial view. In this section, we first explain CSG manipulation techniques to remove ceilings and add thickness to walls for the construction of view-independent wall models. Second, we propose a technique to optimize the visibility of wall models for a given viewing direction, in particular, lower the back-facing walls to construct view-dependent wall models optimized for the view. Third, we introduce our scalable texture mapping algorithm, which can handle a very large mesh with tens of thousands of input images. Lastly, we explain our interactive indoor scene navigation systems that make use of the wall models.

View-independent wall model construction

Denote a reconstructed 3D CSG model from previous section as T, which consists of additive and subtractive primitives (cuboids). We expand T to generate an inflated model T^{fat}, then subtract the volume of T from T^{fat}, which carves out interior volume and leaves a thickened wall structure outside (See Fig. 5-30).

In detail, let Δ be a manually specified thickness of the wall ($\Delta = 0.5$m). We inflate

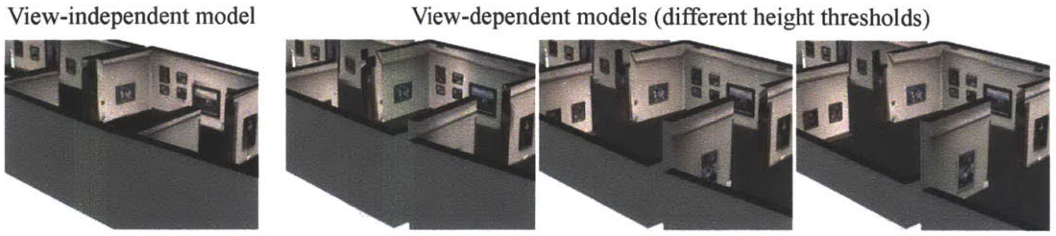


Figure 5-29: View-dependent model construction can lower back-facing walls for increased visibility. Different height thresholds are used to generate models at the right. This example shows the close-ups of the State Tretyakov Gallery.

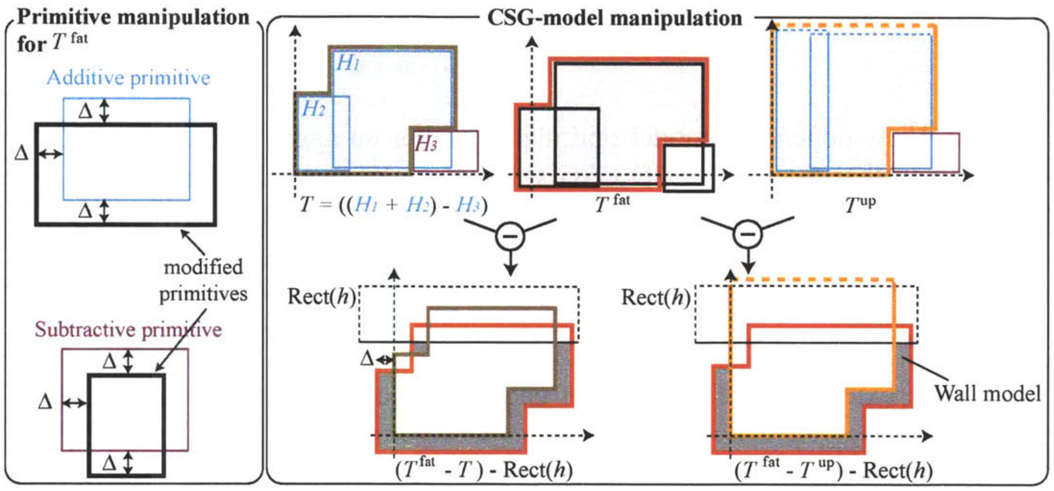
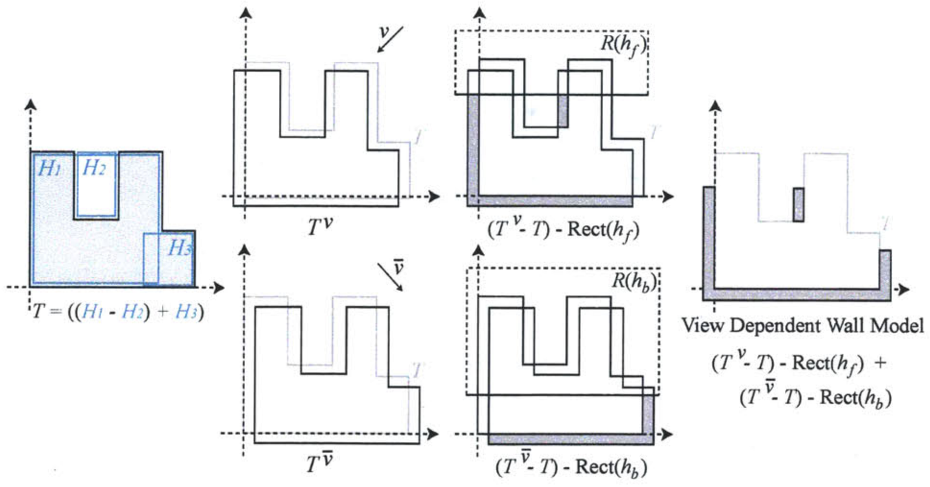


Figure 5-30: Given a CSG model T, we expand each additive primitive and shrink each subtractive primitive (left), to generate an inflated model T^{fat}. Subtracting T from this inflated model T^{fat} gives us a wall model with thickness (right). To remove floating walls due to a complex ceiling structure, we subtract T^{up} that is constructed to have infinite height, instead of T.

(resp. shrink) each additive (resp. subtractive) primitive horizontally, to move outwards (resp. inwards) each vertical face by Δ. The model is then translated downwards by Δ, so that ceilings are removed when subtracting T. This modified CSG model is denoted as T^{fat}, and $(T^{\text{fat}} - T)$ produces a model with thickened walls and without ceilings. We also limit the maximum height h of a structure by further subtracting $\text{Rect}(h)$: $((T^{\text{fat}} - T) - \text{Rect}(h))$, where $\text{Rect}(h)$ denotes a cuboid whose bottom side is at height h and extends to infinity at the other five directions. h is set to 6 meters. Illustrated in Fig. 5-30, this construction produces a solid geometry for every vertical facade in a model. However, there may still be some unwanted fragments in the air near complex ceiling structures. Therefore, we construct $((T^{\text{fat}} - T^{\text{up}}) - \text{Rect}(h))$ instead as our wall model. The difference is to construct



H_1
H_2
H_3
$T = ((H_1 - H_2) + H_3)$
v
T^v
$\bar{v}$
$T^{\bar{v}}$
$R(h_f)$
$(T^v - T) - \mathrm{Rect}(h_f)$
$R(h_b)$
$(T^{\bar{v}} - T) - \mathrm{Rect}(h_b)$
View Dependent Wall Model
$(T^v - T) - \mathrm{Rect}(h_f) +$
$(T^{\bar{v}} - T) - \mathrm{Rect}(h_b)$

Figure 5-31: View dependent model optimization. For an application with a fixed viewpoint, we can further manipulate 3D CSG structure to lower back facing walls. We construct front facing walls (top row in the middle columns) and back facing walls (bottom row in the middle columns) independently, which are merged by simply taking the union operation in the CSG representation to construct a view dependent wall model.

T^{up} from T by extending the top face of each additive primitive to infinite height, and subtract T^{up} instead of T.

View-dependent wall model construction

When a viewing direction is fixed in an application, we can further optimize the visibility of the model for the given direction by lowering back-facing walls. Fig. 5-31 illustrates how CSG model can be further manipulated to lower back-facing wall. Let v be the viewing direction, for which the model is to be optimized, and denote h_f and h_b as the maximum height of the structure to be reconstructed for the front-facing and back-facing walls, respectively. For each primitive cuboid, consider a local coordinate frame whose XYZ axes are aligned with the cuboid. Then, we translate the primitive along each of the XYZ axes along v direction by $\Delta(= 0.5\mathrm{m})$. Let us denote this translated model as T^v, as shown in the second column of Fig. 5-31, then $(T^v - T^{\mathrm{up}}) - \mathrm{Rect}(h_f)$ generates front-facing facades with the height limited at h_f as in the third column of Fig. 5-31. Similarly, let $\bar{v}$ be the inverted reflection vector of v against the ground plane, then the back-facing facades are modeled by $(T^{\bar{v}} - T^{\mathrm{up}}) - \mathrm{Rect}(h_b)$. The finally view-dependent 3D model is obtained by taking their union:

$$((T^{v}-T^{\mathrm{up}})-\mathrm{Rect}(h_f))+((T^{\bar{v}}-T^{\mathrm{up}})-\mathrm{Rect}(h_b)).$$

h_f and h_b are set to 6 and 2 meters, respectively, in our experiments. Effects of the view-dependent construction on real examples are shown in Fig. 5-29.

Texture mapping

The last step of our pipeline is to texture-map a wall model from registered photographs. Modern techniques minimize texture stitching artifacts over multiple faces simultaneously [156, 193]. We take a similar approach but with modifications, as our system needs to be scalable to an entire building with tens of thousands of input images. It needs to be also robust against reconstruction errors in a 3D model as well as large registration errors among cameras and a 3D model, which is a common problem for laser scanning of a large-scale environment. Our approach is to extract piecewise planar structures in a model as *face groups*, which ideally correspond to walls, floors, and ceilings, then solve a stitching problem in each face group independently[7]. Note that stitching artifacts are expected to be present at face group boundaries, but this is not an issue in practice, because face group boundaries typically correspond to room and floor boundaries, where textures are not very important. While our CSG representation inherently contains such face group information, the library we used in our implementation – CGAL [24] – does not keep this information when triangulating a mesh model. Also, a single wall is not reconstructed as planar at times, mainly due to the noise in the input laser scan. Therefore, we design a simple region-growing algorithm to identify face groups from a triangulated mesh model: We first pick a face f with the largest area from the mesh to create a new group, and keep adding a neighboring face f' to the group if f and f' are nearly coplanar, until no more faces can be added. Then, we remove all the faces of the group from the mesh and start again from the beginning. For each face group, we use [156] to stitch a texture image with blending. At times, even a single face group becomes large, yielding expensive graph-cuts

[7]We did consider a large-scale graph-cut algorithm [31], but it is still expensive. Our scheme exploits our compact 3D model, and allows easy implementation that works well in practice.

optimization. Therefore, we solve a stitching problem at a coarse level (roughly one node per $20 \times 20\text{cm}^2$), then up-sample the resulting stitching information to generate higher resolution texture images (one pixel per 1cm^2). Lastly, we observed in our experiments that floor textures suffer from very strong artifacts and become mere distractions, mainly because all the cameras are positioned either parallel to the ground or upwards. We identify floor by taking face groups whose vertices are all within 0.05m from the ground, and fill in with a solid gray color. We also assign a gray color to face groups that are not visible from any input camera.

Ground view and aerial view navigation

A global texture-mapped mesh model enables visualization of large indoor scenes from aerial viewpoints, which is more effective for navigation and exploration than traditional ground-only (e.g., panorama-based) visualization. The model can be rendered from arbitrary viewpoints, but is not effective for close-range ground viewpoints, because our model only captures dominant facade geometry and lacks in small objects or any non-rigid structure such as water from fountain. For a ground viewpoint, a nearby input image is much more immersive, free from any artifacts, and can visualize even non-rigid objects and view-dependent effects such as non-diffuse reflections. However, ground level visualization alone is ineffective for navigation due to the limited visibility and mobility.

To integrate the merits from both modes and overcome their disadvantages, we propose to employ traditional panorama based navigation for ground viewpoints, while allowing users to switch between ground and aerial viewpoints at any time. In our implementation, we geo-register our 3D mesh models and load them to the Google Earth to control the camera path as in Fig. 5-32. Our visualization system enables users to easily browse a large-scale indoor environment from a bird's-eye view, locate specific room interiors, fly into a place of interest, view immersive ground-level panorama views, and zoom out again, all with seamless 3D transitions. See our project website for the videos [167].

Another popular digital mapping implementation is tile-based visualization (See Fig. 5-33), where pre-rendered tile images are displayed to users in multiple resolutions, which allow intuitive panning and zooming operations. Standard top-down views are not effective

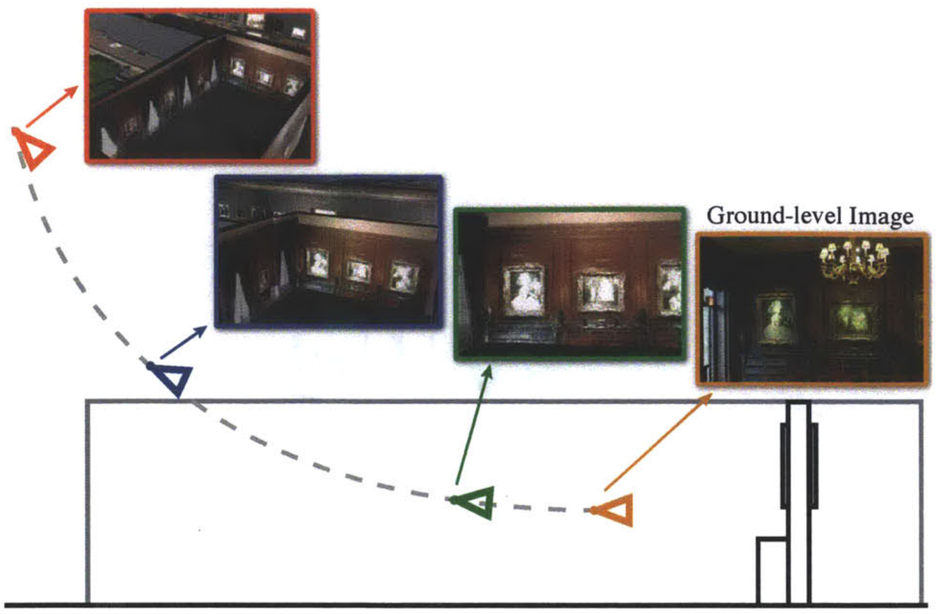


Figure 5-32: Transition from bird's eye view to ground-level view. This enables an intuitive transition between two modes of visualization, and gives the map users a better sense of their current location.

for indoor scenes, where vertical facades contain important information but are not visible. Therefore, we pre-render our texture mapped model from four oblique viewing directions, corresponding to north, east, west, and south-headings, then load the rendered images as a set of tiles to Google Maps through Google Maps API. Note that this is a perfect example to make use of view-dependent wall models, where for each heading, a view-dependent model, optimized for the corresponding viewing direction, is used to render tiles. See our project website to try this demo [167].

5.7.4 Results and discussions

We have evaluated our system on a variety of museums in the world. Our smallest data set is *National Gallery* in London consisting of a single run and 2,670 images, while the largest one is *The Metropolitan Museum of Art* (Met) in New York City with 32 runs, 42,474 images, and more than two hundred million laser points (See Table 5.1). The major computation time is on the 3D reconstruction, where the running times for the 2D CSG and the 3D CSG modeling are listed in Table 5.1. Since computation over multiple runs can be executed in parallel, we list running time for both the serial and the parallel executions.

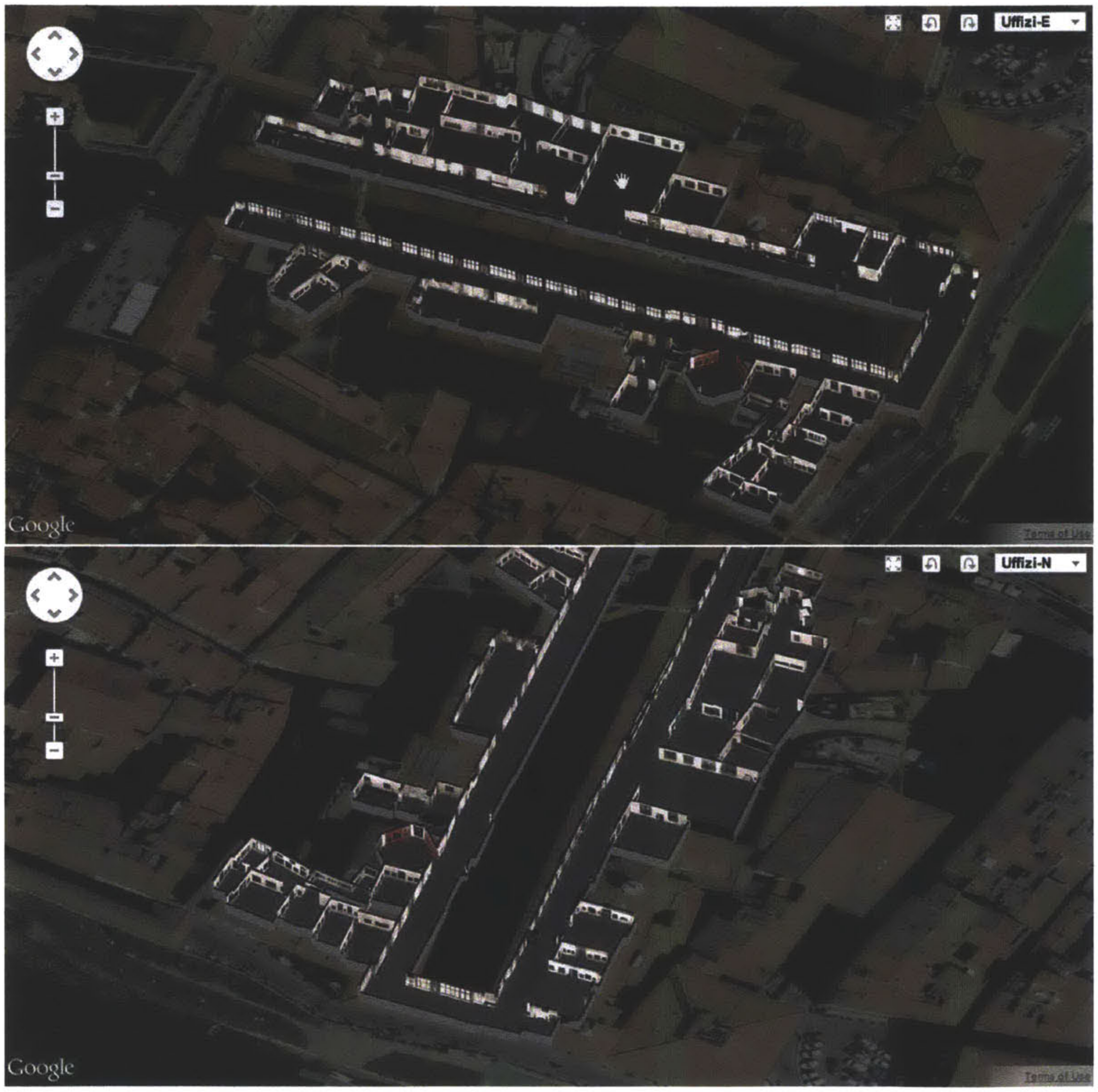


Figure 5-33: A global texture-mapped mesh model allows indoor scene visualization based on pre-rendered tiles, which has been a popular technique for online digital mapping products. We generate tiles for four different headings (i.e., north, south, east and west) in multiple resolution levels by using the view-dependent wall models, where back-facing walls are lowered to improve visibility for each direction. This figure shows the Uffizi Gallery from the two different headings.

Table 5.1: Statistics on input and output data, as well as running time.

			National Gallery	Frick	Tretyakov	Hermitage	Uffizi	Met
input	# of runs		1	8	19	20	48	32
	# of images		2,670	8,466	3,274	21,174	11,622	42,474
	# of laser points [million]		12.3	39.0	15.2	75.7	43.8	201.8
output	# of layers (run-average)		9	7.9	5	8.8	7.3	9.8
	# of primitives	additive	14	66	73	137	302	330
		subtractive	2	9	6	23	60	38
	# of triangles		1,050	4,962	3,830	6,902	18,882	22,610
	# of face groups		108	342	282	485	1,123	2,123
run time	sequential [hour]	2D CSG	3.2	4.8	1.9	6.1	19.2	20.1
		3D CSG	4.4	3.2	0.4	13.8	24.4	25.2
	parallel [hour]	2D CSG	3.2	1.0	0.2	0.7	3.9	3.9
		3D CSG	4.4	2.5	0.07	3.0	4.2	2.9

The parallel execution is simulated with a single process, with the maximum running time over all the processes recorded.

Intermediate reconstruction results are shown in Fig. 5-24 and Fig. 5-28. Fig. 5-24 shows 2D CSG reconstruction results for one run of Met, which consists of eleven horizontal slices. Note that 2D CSG models are fairly consistent across different slices but are not exactly, and hence, a 3D CSG construction step is necessary to produce a clean and compact 3D model. Fig. 5-28 shows how the 3D CSG construction algorithm iteratively improves the model for runs in National Gallery and Met. Notice that a very small number of simple cuboids can represent fairly complex building structure.

Fig. 5-27 shows reconstruction results of *The Frick Collection*, illustrating the stacked 2D CSG models at the top row, which are further simplified by the 3D CSG model construction. As listed in Table 5.1, the entire museum is represented by only 75 volumetric primitives (66 additive and 9 subtractive), where the merged mesh model only contains 4,962 triangles. The figure shows that there is a fair amount of overlap between runs, which has been successfully merged by simply taking the union of their CSG models. A door is usually reconstructed by a box covering the pathway between two rooms (e.g. the second row of Fig. 5-23).

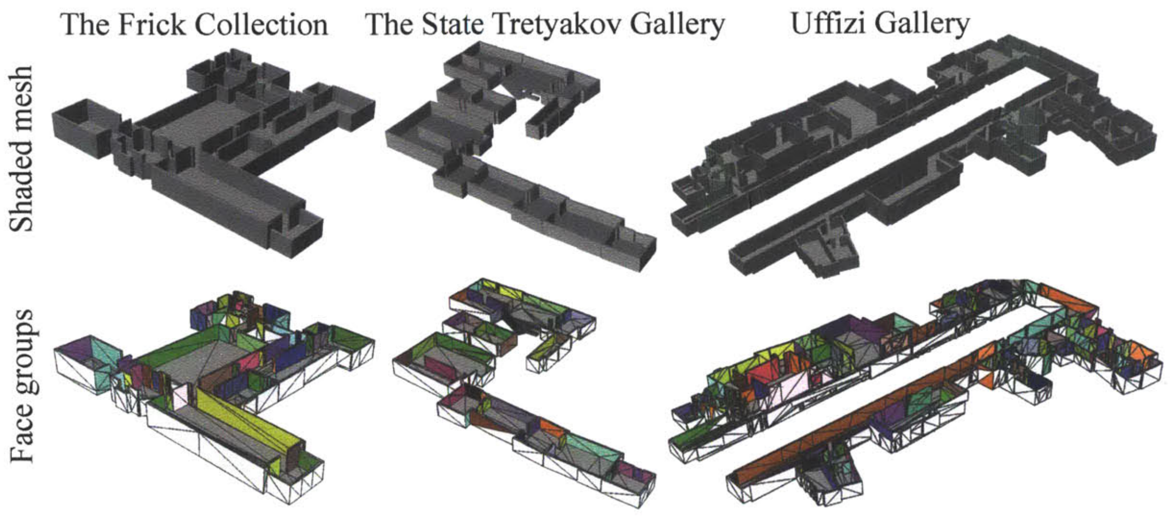


Figure 5-34: Shaded renderings of the view-dependent models and the corresponding face-groups. A random color is assigned to each group except for the white faces that are not visible from any input camera and do not require texture stitching, and gray faces that are identified as floors.

Figure 5-34 shows shaded renderings of the view-independent wall models and the extracted face groups, which illustrates that the algorithm successfully identifies major floor and wall structures as face groups. Figure 5-38 shows final texture-mapped models with some close-ups. The models in these two figures are obtained by the view-dependent CSG construction for better visibility. Figure 5-39 also shows final texture-mapped mesh model of Met with close-ups, which is the largest reconstruction in our experiments. The model is constructed by the view-independent method. These examples illustrate that the stitched textures are of high quality with very few artifacts. Note that our model focuses on dominant facades and does not capture small-scale details or objects. Nonetheless, we can often see and even recognize such missing structure and objects in texture-mapped imagery. Thanks to our regularized mesh model, an image usually goes through a simple affine transformation (assuming weak perspective) during texture mapping without much stitching. Our eyes are surprisingly good at correcting such low frequency distortions and understanding image contents. On the other hand, texture stitching, which is more necessary to fill-in texture for complicated meshes, causes annoying artifacts to our eyes. Our approach pushes stitching artifacts to facade boundaries and succeeds in producing high

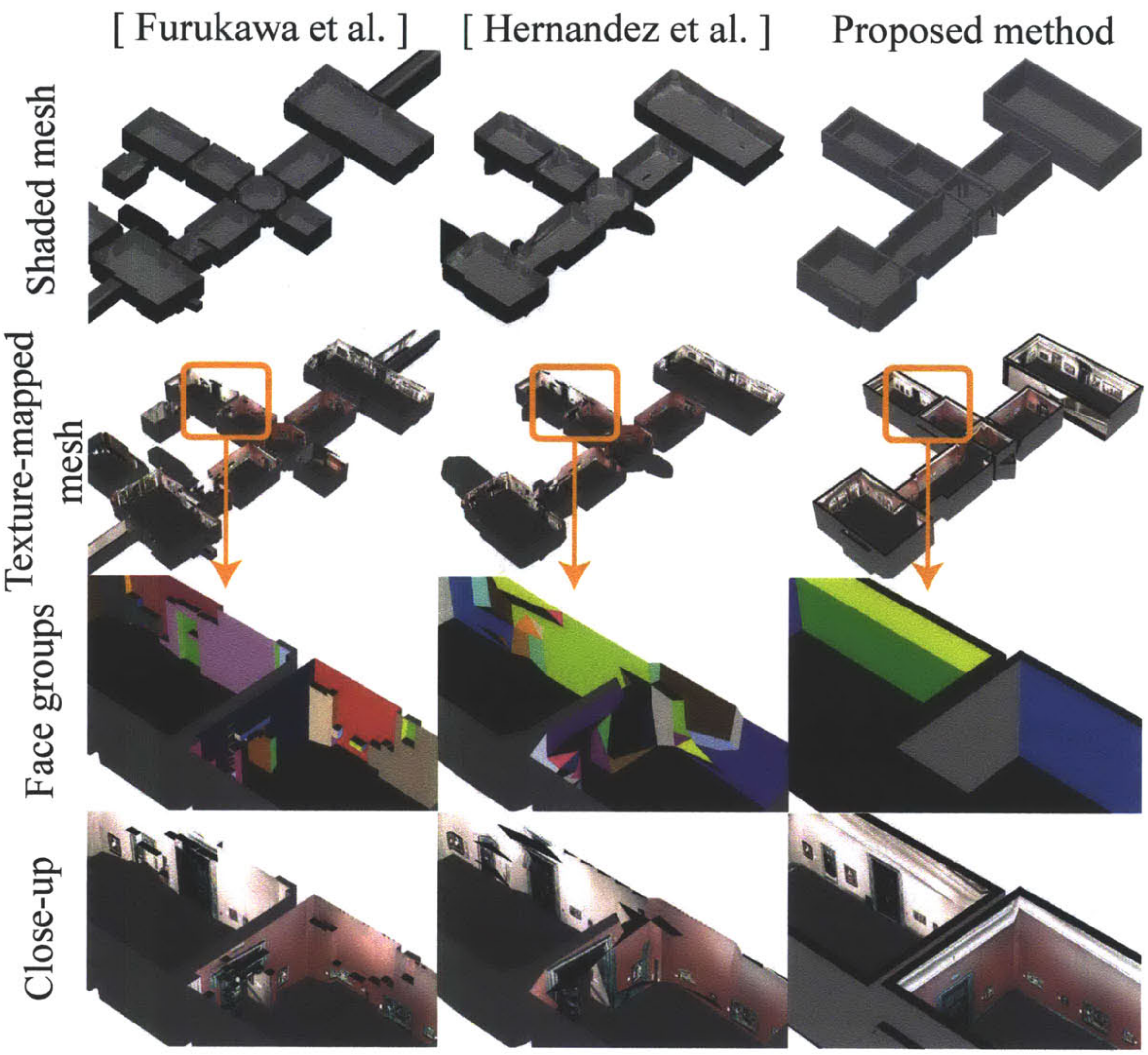


Figure 5-35: Comparison with two state-of-the-art 3D reconstruction algorithms [55, 72].

quality textures in the middle of each facade even where geometry is inaccurate. In an extreme case, at the bottom right in Fig. 5-39, our model misses an entire room, but the room can be recognized through texture-mapped imagery on a planar wall without major artifacts.

The effects of the view-dependent construction are illustrated in Fig. 5-29, where different thresholds are used to control the height of back-facing walls. Paintings are completely occluded with the view-independent model, but become visible when back facing walls are lowered.

Unlike traditional surface reconstruction from laser scan, it is not our goal to make

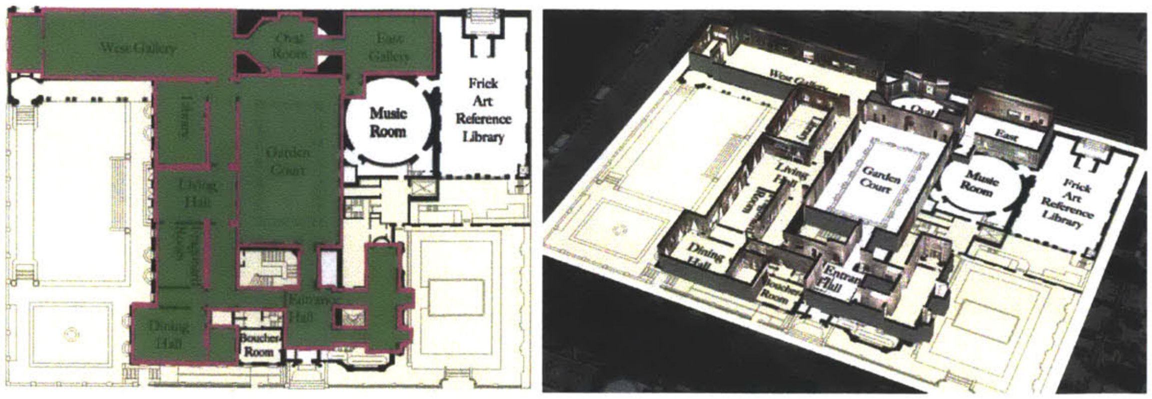


Figure 5-36: Left: Manually overlaying our model with a floor plan image of The Frick Collection. For our model, floor is colored in green and walls are in pink. Right: This overlaying enables hybrid visualization where the texture-mapped model is rendered over the floor plan image and a (darkened) satellite imagery.

models as physically accurate as possible, but to reconstruct a global, compact, and well-regularized mesh for high-quality aerial view visualization. It is a totally different challenge, where it makes more sense to compare with vision algorithms designed to handle noise using strong regularization. In Fig. 5-35, we compare our algorithm with two state-of-the-art techniques – Hernández et al. [72] and Furukawa et al. [55], both of which can usually tolerate noise better than a technique used by the Computer Graphics community [26]. However, because neither approach is scalable due to the need of a high resolution voxel grid, the smallest dataset *National gallery* is used for comparison. Since both algorithms merge depth maps into a mesh model, each laser point is treated as a single depth map pixel. The output of Hernández's algorithm [72] is a very dense model, where face-grouping and texture mapping fail completely due to pose errors. Therefore, we use a mesh simplification software QSlim [56] to reduce the number of triangles to 2000, which is still twice as many as our 3D model. After reconstruction, we clip their models above a certain height to remove ceilings, then apply our face-grouping and the texture-mapping algorithm. Noisy surfaces and severe stitching artifacts are noticeable for both approaches in Fig. 5-35, where significant noise in the laser points cannot be regulated well by their Markov Random Field (MRF) formulation. At macro scale, several rooms suffer from space distortions due to the pose error, which breaks the Manhattan-world assumption for [55]. Our algorithm, in turn, succeeds in producing clean texture-mapped 3D models, illus-

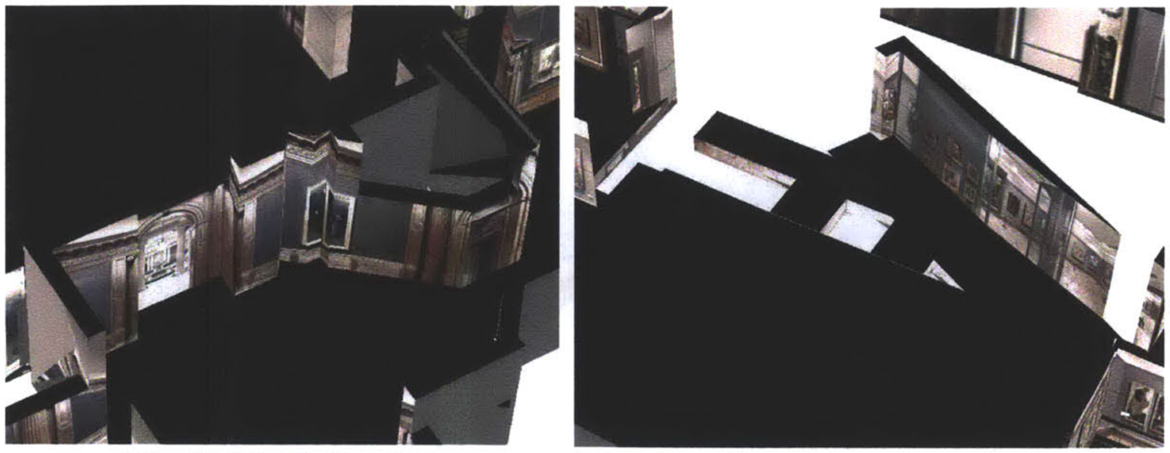


Figure 5-37: Limitations of current approach. Left: Our structure assumption is more general than Manhattan-world, but cannot handle curved surfaces such as an oval room. Right: When input pose errors are significant, our geometry reconstruction completely fails.

trating the effectiveness of the texture mapping algorithm leveraging our highly regularized 3D models. Although Furukawa et al. produces more rooms, a careful look at the textured models would reveal severe artifacts in texture for the extra rooms. This is simply because the data capturing device did not enter those rooms, and they are partially scanned by cameras and lasers seeing through windows/doorways. Existing methods only enforce local weak regularization and cannot suppress reconstructions of such rooms. We enforce strong regularization to avoid these rooms where the laser scanner does not reach well.

The lack of ground-truth data prevents us from conducting quantitative evaluations. To qualitatively assess reconstruction accuracy, in Fig. 5-36, we manual overlay our wall model onto a floor plan image of The Frick Collection. Our model aligns fairly well, which enables interesting hybrid visualization at the right of Fig. 5-36: The texture-mapped model is rendered on top of the floor plan image, without face-groups identified as floors.

Lastly, our system is not free from errors as in any other large-scale systems. Small structures such as doorways tend to be missed (e.g., National Gallery in Fig. 5-38), mostly due to line detection errors. Although our system handles non-Manhattan structures, it is still limited to rectangular structures, and cannot properly model the oval room in the Frick Collection (Fig. 5-36), or the hexagonal shaped room in National Gallery (Fig. 5-

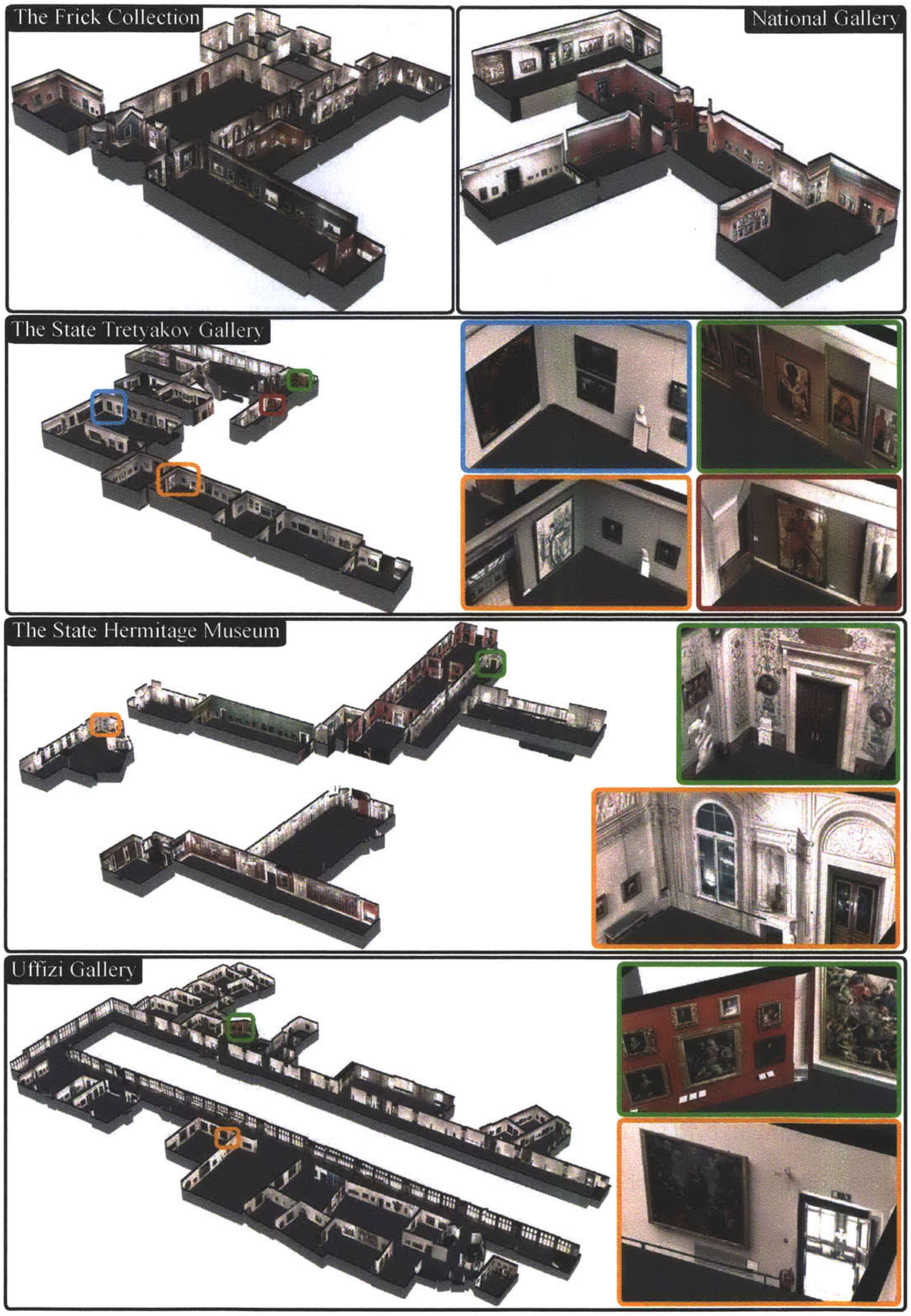


Figure 5-38: Final view-dependent texture-mapped 3D models with some close-ups. More results are available in the project website [167].

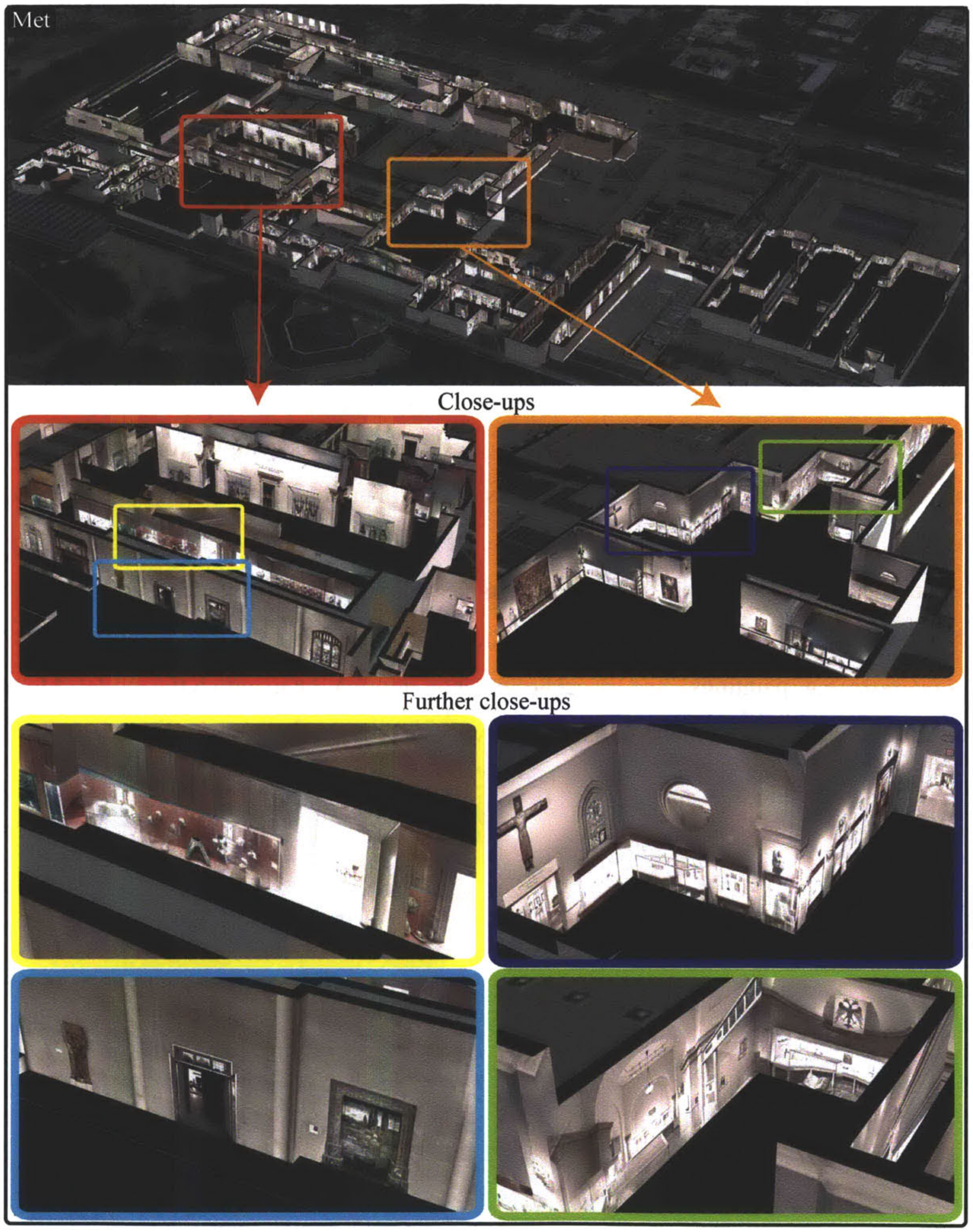


Figure 5-39: The texture-mapped 3D model for The Metropolitan Museum of Art. In the close-up at the bottom right, the texture of an entire room is mapped to a plane due to the lack of precise geometry. The stitched texture is free from major artifacts, except for an inconsistent perspective.

38). Texture mapping suffers from severe artifacts at such places as shown in the left of Figure 5-37. Input pose errors in cameras and laser points also yield severe artifacts as in the right of Fig. 5-37, where the geometry reconstruction completely fails.

5.7.5 Summary

We have presented a novel indoor modeling and visualization system that directly solves for a CSG model from laser points. Our system has produced high-quality texture-mapped 3D models that are effective for rendering from aerial viewpoints, which we believe opens up new possibilities for indoor scene navigation and mapping. Our future work includes extension of geometric primitive types to more shapes, such as cylinders or spheres. Our major failure modes are due to severe errors in the input pose information, and we need to explorer better pose estimation algorithm. It is also interesting to model objects in a scene, such as sculptures, glass cabinets and tables that are currently missing in our models. Another challenge would be to go beyond the notion of face-groups, and identify *objects of interests* (e.g. paintings), and associate semantic information to enrich the model. More results and demo can be found in our project website [167].

5.8 Conclusion

We argue for the great importance of place-centric representation of the 3D space for scene understanding. To construct a large place-centric dataset in 3D, we propose a human-in-the-loop system that incorporates semantic labels introduced by user to obtain an accurate 3D reconstruction of large places scanned with an RGBD camera, and to provide object segmentations and labels for all the objects in the environment. This novel place-centric representation with rich data opens up many exciting new opportunities, such as, integrating scene recognition over space. It also has many applications such as obtaining a scene-level 3D reconstruction of large environments.

Chapter 6

Conclusion

In this dissertation, I focus on leveraging rich data to build computer systems to understand visual scenes, both inferring the semantics and extracting 3D structure for a large variety of environments. We hope that this thesis will inspire researchers to get one step closer to eventually pass the "basic level scene understanding" Turing test.

We conclude with an overall statement of the philosophy that has driven my research: Scene understanding is a complicated process, because it depends on both the *rich 3D structure* of the physical world and the observer's *viewpoint*. To build computer systems to understand visual scenes, it is important to consider the role of computer as an active explorer in a 3D world and use rich data that is close to the natural input that human have. The underlying principle is that exploring and learning from these rich descriptions of data will be crucial for finding the right representation to eventually solve computer vision.

Bibliography

[1] `http://www.360cities.net/`.

[2] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. Building Rome in a day. *Communications of the ACM*, 2011.

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

[4] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *Proceedings of IEEE International Conference on Computer Vision*, 2009.

[5] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Trans. Graph. (Proc. of SIGGRAPH)*, 23(3):294–302, 2004.

[6] G.J. Agin and T.O. Binford. Computer description of curved objects. *IEEE Transactions on Computers*, 100(4):439–449, 1976.

[7] Timo Ahonen, Jiří Matas, Chu He, and Matti Pietikäinen. Rotation invariant image description with local binary pattern histogram fourier features. In *SCIA*, 2009.

[8] Pablo Arbelaez, Charless Fowlkes, and David Martin. The berkeley segmentation dataset and benchmark. *see http://www. eecs. berkeley. edu/Research/Projects/CS/vision/bsds*, 2007.

[9] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011.

[10] Autodesk. Revit architecture, 2012.

[11] Alper Aydemir, Rasmus Göransson, and Patric Jensfelt. *Kinect@Home*, 2012.

[12] Sid Yingze Bao and Silvio Savarese. Semantic structure from motion. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[13] M. Bar. Visual objects in context. *Nature Reviews Neuroscience*, 5(8):617–629, 2004.

[14] K. Barnard, P. Duygulu, N. de Freitas, D. Forsyth, D. Blei, and M. Jordan. Matching words and pictures. *Journal of Machine Learning Research*, 3:1107–1135, 2003.

[15] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. Simultaneous structure and texture image inpainting. *IEEE Transactions of Image Processing*, 12(8):882–889, 2003.

[16] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proc. SIGGRAPH*, pages 417–424, 2000.

[17] Irving Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94:115–147, 1987.

[18] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Extracting 3d scene-consistent object proposals and depth from stereo images. In *Proceedings of European Conference on Computer Vision*, pages 467–481. Springer, 2012.

[19] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

[20] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[21] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[22] J. Bunge and M. Fitzpatrick. Estimating the number of species. *Journal of the American Statistical Association*, 88(421):364–373, 1993.

[23] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[24] CGAL. Computational Geometry Algorithms Library, 2012. http://www.cgal.org.

[25] Antonio Criminisi, Patrick Perez, and Kentaro Toyama. Object removal by exemplar-based inpainting. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[26] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312, 1996.

[27] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[28] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B Goldman, and Pradeep Sen. Image melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics (TOG)*, 31(4), 2012.

[29] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, MarcAurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, 2012.

[30] Luca Del Pero, Jinyan Guan, Ernesto Brau, Joseph Schlecht, and Kobus Barnard. Sampling bedrooms. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2009–2016. IEEE, 2011.

[31] Andrew Delong and Yuri Boykov. A scalable graph-cut algorithm for n-d grids. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[33] P. Dev. Segmentation processes in visual perception: A cooperative neural model. In *Proceedings of the 1974 Conference on Biologically Motivated Automata Theory, McLean, Va., June*, 1974.

[34] Sven J Dickinson, Dimitri Metaxas, and Alex Pentland. The role of model-based segmentation in the recovery of volumetric parts from range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(3):259–267, 1997.

[35] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. Fragment-based image completion. 22(3):303–312, 2003.

[36] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proc. SIGGRAPH*, 2001.

[37] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of IEEE International Conference on Computer Vision*, 1999.

[38] Krista A Ehinger and Aude Oliva. Canonical views of scenes depend on the shape of the space. 2011.

[39] Krista A. Ehinger, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Estimating scene typicality from human ratings and image features. In L. Carlson, C. Hlscher, and T. Shipley, editors, *Proceedings of the 33rd Annual Conference of the Cognitive Science Society*, pages 2114–2119. Cognitive Science Society, 2011.

[40] Krista Anne Ehinger. Visual features for scene recognition and reorientation. In *PhD. Thesis*, 2013.

[41] Felix Endres, Juergen Hess, Nikolas Engelhard, Juergen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2012.

[42] Ian Endres and Derek Hoiem. Category independent object proposals. In *Proceedings of European Conference on Computer Vision*, pages 575–588. Springer, 2010.

[43] R. Epstein and N. Kanwisher. A cortical representation of the local visual environment. *Nature*, 392:598–601, 1998.

[44] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge. *International Journal of Computer Vision*, 2010.

[45] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[46] M. Eysenck. *Cognitive Psychology: A Student's Handbook: A Student's Handbook 5th Edition.* Psychology Press, 2005.

[47] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples. In *Workshop on Generative-Model Based Vision, IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[48] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 524–531, 2005.

[49] C. Fellbaum. *Wordnet: An Electronic Lexical Database.* Bradford Books, 1998.

[50] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1), 2005.

[51] Pedro Felzenszwalb and Daniel Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell, 2004. Computing and Information Science TR2004-1963.

[52] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[53] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[54] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Manhattan-world stereo. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[55] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Reconstructing building interiors from images. In *Proceedings of IEEE International Conference on Computer Vision*, 2009.

[56] Michael Garland. Qslim: Quadric-based simplification algorithm, 1998.

[57] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[58] James Jerome Gibson. *The ecological approach to visual perception.* Routledge, 1986.

[59] M.A. Goodale and A.D. Milner. Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1):20–25, 1992.

[60] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[61] Abhinav Gupta, Alexei A. Efros, and Martial Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *Proceedings of European Conference on Computer Vision*, 2010.

[62] Abhinav Gupta, Scott Satkin, Alexei A. Efros, and Martial Hebert. From 3D scene geometry to human workspace. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[63] P.M. Hans. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th international joint conference on Artificial intelligence*, pages 584–584, 1977.

[64] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[65] James Hays and Alexei A Efros. Scene completion using millions of photographs. *SIGGRAPH*, 2007.

[66] James Hays and Alexei A. Efros. im2gps: estimating geographic information from a single image. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[67] Kaiming He and Jian Sun. Statistics of patch offsets for image completion. In *Proceedings of European Conference on Computer Vision*, 2012.

[68] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering the spatial layout of cluttered rooms. In *Proceedings of IEEE International Conference on Computer Vision*, 2009.

[69] Varsha Hedau, Derek Hoiem, and David Forsyth. Thinking inside the box: Using appearance models and context based on room geometry. In *Proceedings of European Conference on Computer Vision*, 2010.

[70] Varsha Hedau, Derek Hoiem, and David Forsyth. Recovering free space of indoor scenes from a single image. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[71] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments, 2010.

[72] Carlos Hernández Esteban, George Vogiatzis, and Roberto Cipolla. Probabilistic visibility for multi-view stereo. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[73] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proc. SIGGRAPH*, 2001.

[74] J. Hochberg. Perception (2nd edn), 1978.

[75] D. Hoiem, A.A. Efros, and M. Hebert. Putting objects in perspective. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[76] D. Hoiem, A.A. Efros, and M. Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 75(1), 2007.

[77] Derek Hoiem. *Seeing the World Behind the Image: Spatial Layout for 3D Scene Understanding*. PhD thesis, Carnegie Mellon University, 2007.

[78] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Geometric context from a single image. In *Proceedings of IEEE International Conference on Computer Vision*, 2005.

[79] Y. Horry, K.-I. Anjyo, and K. Arai. Tour into the picture: using a spidery mesh interface to make animation from a single image. *SIGGRAPH*, pages 225–232, 1997.

[80] Paul V.C. Hough. Machine analysis of bubble chamber pictures. In *Proceedings of International Conference on High Energy Accelerators and Instrumentation*, 1959.

[81] Hai Huang, Claus Brenner, and Monika Sester. 3D building roof reconstruction from point clouds via generative models. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 16–24, 2011.

[82] Qi-Xing Huang and Dragomir Anguelov. High quality pose estimation by aligning multiple scans to a latent map. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2010.

[83] K. Ikeuchi and T. Suehiro. Toward an assembly plan from observation: Task recognition with polyhedral objects. In *Robotics and Automation*, 1994.

[84] H. Intraub and M. Richardson. Wide-angle memories of close-up scenes. *Journal of experimental psychology. Learning, memory, and cognition*, 15(2), March 1989.

[85] Helene Intraub. Rethinking scene perception: A multisource model. *Psychology of learning and motivation*, 52:231–264, 2010.

[86] Phillip Isola, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. What makes an image memorable. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[87] Allison Janoch, Sergey Karayev, Yangqing Jia, Jonathan T. Barron, Mario Fritz, Kate Saenko, and Trevor Darrell. A category-level 3-d object dataset: Putting the kinect to work. In *ICCV Workshop on Consumer Depth Cameras for Computer Vision*, 2011.

[88] Roland Fleming JErum Arif Khan, Erik Reinhard and Heinrich Buelthoff. Image-based material editing. *ACM Trans. Graph. (Proc. of SIGGRAPH)*, August 2006.

[89] Hao Jiang and Jianxiong Xiao. A linear approach to matching cuboids in rgbd images. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[90] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1), 2009.

[91] P. Jolicoeur, M.A Gluck, and S.M. Kosslyn. Pictures and names: Making the connection. *Cognitive Psychology*, 16:243–275, 1984.

[92] Biliana Kaneva, Josef Sivic, Antonio Torralba, Shai Avidan, and William Freeman. Matching and predicting street level images. In *Workshop for Vision on Cognitive Tasks, ECCV*, 2010.

[93] Biliana Kaneva, Josef Sivic, Antonio Torralba, Shai Avidan, and William T. Freeman. Infinite images: Creating and exploring a large photorealistic virtual space. In *Proceedings of the IEEE*, 2010.

[94] Kevin Karsch, Ce Liu, and Sing Bing Kang. Depth extraction from video using non-parametric sampling. In *Proceedings of European Conference on Computer Vision*, 2012.

[95] Aditya Khosla, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Memorability of image regions. In *Advances in Neural Information Processing Systems*, 2012.

[96] Johannes Kopf, Wolf Kienzle, Steven Drucker, and Sing Bing Kang. Quality prediction for image completion. *ACM Trans. Graph. (Proc. of SIGGRAPH Asia)*, 31(6), 2012.

[97] Hema S Koppula, Abhishek Anand, Thorsten Joachims, and Ashutosh Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *Advances in Neural Information Processing Systems*, pages 244–252, 2011.

[98] H.S. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *Advances in Neural Information Processing Systems*, 2011.

[99] Jana Kosecka and Wei Zhang. Video compass. In *Proceedings of European Conference on Computer Vision*, 2002.

[100] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Trans. Graph. (Proc. of SIGGRAPH)*, 24(3):795–802, 2005.

[101] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2011.

[102] Jean-François Lalonde, Alexei A. Efros, and Srinivasa G. Narasimhan. Webcam clip art: Appearance and illuminant transfer from time-lapse sequences. *ACM Trans. Graph. (Proc. of SIGGRAPH Asia)*, 28(5), December 2009.

[103] Jean-François Lalonde, Derek Hoiem, Alexei A. Efros, Carsten Rother, John Winn, and Antonio Criminisi. Photo clip art. *ACM Transactions on Graphics (SIGGRAPH)*, 26(3), 2007.

[104] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2169–2178, 2006.

[105] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[106] David C Lee, Abhinav Gupta, Martial Hebert, and Takeo Kanade. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. In *Advances in Neural Information Processing Systems*, volume 1, page 3. Citeseer, 2010.

[107] Ales Leonardis, Ales Jaklic, and Franc Solina. Superquadrics for segmenting and modeling range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1289–1295, 1997.

[108] Anat Levin, Assaf Zomet, and Yair Weiss. Learning how to inpaint from global image statistics. In *Proceedings of IEEE International Conference on Computer Vision*, 2003.

[109] Li-Jia Li, Hao Su, Eric P. Xing, and Li Fei-Fei. Object bank: A high-level image representation for scene classification & semantic feature sparsification. In *Advances in Neural Information Processing Systems*, 2010.

[110] Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. Globfit: consistently fitting primitives by discovering global relations. In *ACM Transactions on Graphics (TOG)*, volume 30, page 52. ACM, 2011.

[111] Timothy Liu, Matthew Carlberg, George Chen, Jacky Chen, John Kua, and Avideh Zakhor. Indoor localization and visualization using a human-operated backpack system. In *Proceedings of International Conference on Indoor Positioning and Indoor Navigation*, 2010.

[112] HC Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms, MA Fischler and O. Firschein, eds*, pages 61–62, 1987.

[113] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[114] K. Lyle and M. Johnson. Importing perceived features into false memories. *Memory*, 14(2):197–213, 2006.

[115] D. Marr and T. Poggio. Cooperative computation of stereo disparity. Technical report, DTIC Document, 1976.

[116] D. Marr and T. Poggio. A computational theory of human stereo vision. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 204(1156):301–328, 1979.

[117] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of IEEE International Conference on Computer Vision*, 2001.

[118] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761 – 767, 2004.

[119] J. L. Mundy. Object recognition in the geometric era: A retrospective. In *Toward Category-Level Object Recognition, volume 4170 of Lecture Notes in Computer Science*, pages 3–29. Springer, 2006.

[120] V. Nedovic, A. W. M. Smeulders, A. Redert, and J. M. Geusebroek. Depth information by stage classification. In *Proceedings of IEEE International Conference on Computer Vision*, 2007.

[121] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proceedings of 10th IEEE international symposium on Mixed and augmented reality (ISMAR)*, 2011.

[122] Kai Ni, Anitha Kannan, Antonio Criminisi, and John Winn. Epitomic location recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[123] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, C.-C. Chen, Jong Taek Lee, Saurajit Mukherjee, J. K. Aggarwal, Hyungtae Lee, Larry Davis, Eran Swears, Xioyang Wang, Qiang Ji, Kishore Reddy, Mubarak Shah, Carl Vondrick, Hamed Pirsiavash, Deva Ramanan, Jenny Yuen, Antonio Torralba, Bi Song, Anesco Fong, Amit Roy-Chowdhury, and Mita Desai. A large-scale benchmark dataset for event recognition in surveillance video. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[124] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

[125] A Oliva, S Park, and T Konkle. Representing, perceiving, and remembering the shape of visual space. 2011.

[126] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42:145–175, 2001.

[127] Stephen Palmer, Eleanor Rosch, and Paul Chase. Canonical perspective and the perception of objects. *Attention and performance IX*, 1:4, 1981.

[128] Genevieve Patterson and James Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[129] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. Discovering structural regularity in 3d geometry. *SIGGRAPH*, 2011.

[130] Luca Del Pero, Joshua C. Bowdish, Daniel Fried, Bonnie D. Kermgard, Emily L. Hartley, and Kobus Barnard. Bayesian geometric modelling of indoor scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[131] Luca Del Pero, Jinyan Guan, Ernesto Brau, Joseph Schlecht, and Kobus Barnard. Sampling bedrooms. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[132] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[133] Y. Poleg and S. Peleg. Alignment and mosaicing of non-overlapping images. In *Proceedings of IEEE International Conference on Computational Photography*, 2012.

[134] Jean Ponce, Tamara L Berg, Mark Everingham, David A Forsyth, Martial Hebert, Svetlana Lazebnik, Marcin Marszalek, Cordelia Schmid, Bryan C Russell, Antonio Torralba, et al. Dataset issues in object recognition. In *Toward category-level object recognition*, pages 29–48. Springer, 2006.

[135] Y. Pritch, E. Kav-Venaki, and S. Peleg. Shift-map image editing. In *Proceedings of IEEE International Conference on Computer Vision*, pages 151–158, 2009.

[136] Edward S Reed. *James J. Gibson and the psychology of perception*. Yale University Press New Haven, CT, 1988.

[137] L.W. Renninger and J. Malik. When is scene recognition just texture recognition? *Vision Research*, 44:2301–2311, 2004.

[138] Lawrence G Roberts. Machine perception of three-dimensional solids. Technical report, DTIC Document, 1963.

[139] Enrique Valero Rodriguez, Antonio Adan Oliver, Daniel Huber, and Carlos Cerrada. Detection, modeling, and classification of moldings for automated reverse engineering of buildings from 3d data. In *International Symposium on Automation and Robotics in Construction*, 2011.

[140] E. Rosch. Natural categories. *Cognitive Psychology*, 4:328–350, 1973.

[141] E. Rosch, C.B. Mervis, W.D. Gray, D.M. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8:382–439, 1976.

[142] Eleanor Rosch. Principles of categorization. *Concepts: core readings*, pages 189–206, 1999.

[143] B. C. Russell and A. Torralba. Building a database of 3d scenes from user annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[144] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008.

[145] Mohammad Amin Sadeghi and Ali Farhadi. Recognition using visual phrases. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[146] Victor Sanchez and Avideh Zakhor. Planar 3d modeling of building interiors from point cloud data. In *Proceedings of International Conference on Image Processing*, 2012.

[147] A. Saxena, M. Sun, and A.Y. Ng. Make3D: Learning 3d scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[148] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42, 2002.

[149] Ruwen Schnabel, Raoul Wessel, Roland Wahl, and Reinhard Klein. Shape recognition in 3d point-clouds. In *Proc. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*, volume 2. Citeseer, 2008.

[150] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[151] Eli Shechtman and Michal Irani. Matching local self-similarities across images and videos. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[152] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proceedings of European Conference on Computer Vision*. 2006.

[153] Nathan Silberman and Rob Fergus. Indoor scene segmentation using a structured light sensor. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 601–608. IEEE, 2011.

[154] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *Proceedings of European Conference on Computer Vision*, 2012.

[155] Saurabh Singh, Abhinav Gupta, and Alexei A. Efros. Unsupervised discovery of mid-level discriminative patches. In *Proceedings of European Conference on Computer Vision*, 2012.

[156] Sudipta N. Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. Interactive 3D architectural modeling from unordered photo collections. In *SIGGRAPH Asia*, 2008.

[157] J. Sivic and A. Zisserman. Video data mining using configurations of viewpoint invariant regions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[158] Jan Smisek, Michal Jancosek, and Tomas Pajdla. 3d with kinect. In *Consumer Depth Cameras for Computer Vision*. Springer, 2013.

[159] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[160] N. Snavely, S.M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.

[161] John P. Snyder. *Flattening the earth : two thousand years of map projections*. University of Chicago Press, Chicago, 1993.

[162] Merrielle Spain and Pietro Perona. Some objects are more equal than others: Measuring and predicting importance. In *Proceedings of European Conference on Computer Vision*, 2008.

[163] M. Sun, G. Bradski, B.X. Xu, and S. Savarese. Depth-encoded hough voting for joint object detection and shape recovery. *Proceedings of European Conference on Computer Vision*, 2010.

[164] Ildiko Suveg and George Vosselman. Reconstruction of 3d building models from aerial images and maps. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2004.

[165] Richard Szeliski. *Computer Vision: Algorithms and Applications.* Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[166] Ping Tan, Tian Fang, Jianxiong Xiao, Peng Zhao, and Long Quan. Single image tree modeling. *ACM Trans. Graph.*, 27(5):108:1–108:7, December 2008.

[167] Reconstructing the world's museums. `http://mit.edu/jxiao/museum`, 2012.

[168] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large database for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, November 2008.

[169] A. Torralba, K. Murphy, W. Freeman, and M. Rubin. Context-based vision system for place and object recognition. In *Proceedings of IEEE International Conference on Computer Vision*, 2003.

[170] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1521–1528, 2011.

[171] Robert Truax, Robert Platt, and John Leonard. Using prioritized relaxations to locate objects in points clouds for manipulation. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2011.

[172] Tinne Tuytelaars, Christoph H. Lampert, Matthew B. Blaschko, and Wray Buntine. Unsupervised object discovery: A comparison. *International Journal of Computer Vision*, 88(2), 2010.

[173] B. Tversky and K. Hemenway. Categories of environmental scenes. *Cognitive Psychology*, 15:121–149, 1983.

[174] Ullman. *The Interpretation of Visual Motion.* Dissertation, MIT, 1977.

[175] S. Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979.

[176] Matthew Uyttendaele, Antonio Criminisi, Sing Bing Kang, Simon Winder, Richard Szeliski, and Richard Hartley. Image-based interactive exploration of real-world environments. *IEEE Computer Graphics and Applications*, 2004.

[177] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[178] J. Vogel and B. Schiele. A semantic typicality measure for natural scene categorization. In *German Symposium on Pattern Recognition DAGM*, 2004.

[179] J. Vogel and B. Schiele. Semantic model of natural scenes for content-based image retrieval. *International Journal Computer Vision*, 72:133–157, 2007.

[180] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowd-sourced video annotation. *International Journal of Computer Vision*, 2012.

[181] Huayan Wang, Stephen Gould, and Daphne Koller. Discriminative learning with latent variables for cluttered indoor scene understanding. In *Proceedings of European Conference on Computer Vision*, 2010.

[182] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):463–476, 2007.

[183] Oliver Whyte, Josef Sivic, and Andrew Zisserman. Get out of my picture! internet-based inpainting. In *Proceedings of British Machine Vision Conference*, 2009.

[184] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[185] Jianxiong Xiao. Segmentation for image-based modeling. Final Year Thesis for Bachelor of Engineering in Computer Science, The Hong Kong University of Science and Technology, 2007. Undergraduate Thesis.

[186] Jianxiong Xiao, Jingni Chen, Dit-Yan Yeung, and Long Quan. Learning two-view stereo matching. In *Proceedings of European Conference on Computer Vision*, Berlin, Heidelberg, 2008.

[187] Jianxiong Xiao, K.A. Ehinger, A. Oliva, and A. Torralba. Recognizing scene view-point using panoramic place representation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2695–2702, 2012.

[188] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. 2013.

[189] Jianxiong Xiao, Tian Fang, Ping Tan, Peng Zhao, Eyal Ofek, and Long Quan. Image-based façade modeling. *ACM Trans. Graph.*, 27(5):161:1–161:10, December 2008.

[190] Jianxiong Xiao, Tian Fang, Peng Zhao, Maxime Lhuillier, and Long Quan. Image-based street-side city modeling. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 114:1–114:12, New York, NY, USA, 2009. ACM.

[191] Jianxiong Xiao and Yasutaka Furukawa. In *Proceedings of European Conference on Computer Vision*, Berlin, Heidelberg, 2012.

[192] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. Semantic rgb-d bundle adjustment with human in the loop. In *Under review*, 2013.

[193] Jianxiong Xiao and Long Quan. Multiple view semantic segmentation for street view images. In *Proceedings of IEEE International Conference on Computer Vision*, 2009.

[194] Jianxiong Xiao, Bryan Russell, and Antonio Torralba. Localizing 3d cuboids in single-view images. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 755–763. 2012.

[195] Jianxiong Xiao, Bryan C. Russell, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Basic level scene understanding: from labels to structure and beyond. In *SIGGRAPH Asia 2012 Technical Briefs*, SA '12, pages 36:1–36:4, New York, NY, USA, 2012. ACM.

[196] Jianxiong Xiao, Jingdong Wang, Ping Tan, and Long Quan. Joint affinity propagation for multiple view segmentation. In *Proceedings of IEEE International Conference on Computer Vision*, 2007.

[197] Y. Yang and D. Ramanan. Articulated pose estimation using flexible mixtures of parts. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[198] Stella Yu, Hao Zhang, and Jitendra Malik. Inferring spatial layout from a single image via depth-ordered grouping. In *IEEE Workshop on Perceptual Organization in Computer Vision*, 2008.

[199] Jenny Yuen, Bryan C. Russell, Ce Liu, and Antonio Torralba. Labelme video: Building a video database with human annotations. In *Proceedings of IEEE International Conference on Computer Vision*, 2009.

[200] Yinda Zhang, Jianxiong Xiao, James Hays, and Ping Tan. Framebreak: Dramatic image extrapolation by guided shift-maps. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[201] Yibiao Zhao and S.-C. Zhu. Image parsing with stochastic scene grammar. In *Advances in Neural Information Processing Systems*. 2011.