LIDAR DATA MANIPULATION: OBJECT DETECTION, SEGMENTATION, AND INPAINTING

By

David Doria

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: ELECTRICAL ENGINEERING

Approved by the
Examining Committee:

_____
Richard J. Radke, Thesis Adviser

_____
Kim Boyer, Member

_____
Randolph Franklin, Member

_____
Barbara Cutler, Member

Rensselaer Polytechnic Institute
Troy, New York

March 2012 (For Graduation December 2012)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Light Detection and Ranging (LiDAR) is a data acquisition method that uses time-of-flight calculations on laser pulses to obtain 3D point samples of an environment. These data sets contain information about a scene that is impossible to obtain with standard optical imaging, such as the absolute scale of objects, the position of heavily occluded objects, and surface texture information. In this thesis we investigate several interesting uses of such data sets.

First, we propose a strategy for confirming the proposed positions of objects in a scene, independent of a specific detection or registration method. A dual metric, consistency and confidence, is computed to produce a human-interpretable metric of the certainty that an object has been correctly identified. We show that the proposed method works well for scenes containing partially visible or heavily occluded objects.

Next, we propose a technique to interactively segment objects from the background in LiDAR scans. First, we create a conservative segmentation of the object based on user strokes and LiDAR depths, where they are reliable. Second, this segmentation is refined using a traditional graph-cut method based on colors and depths. The resulting segmentations are sharp and cannot be easily achieved with either color-only or depth-only algorithms.

Third, we study how to fill large holes in LiDAR data, typically due to occlusions caused by diverging lines of sight. Our approach generalizes exemplar-based image inpainting and uses both the color and depth information present in a LiDAR scan. A novel gradient-domain algorithm forms the core of the approach. We show that by combining an inpainting algorithm with the proposed algorithm we can quickly and convincingly remove unwanted objects from LiDAR scans and render the scene from different perspectives.

We also discuss the future work to be completed post-candidacy, which addresses several small but important problems in patch-based image inpainting. The focus of these contributions is preventing incorrect patches from being used by the inpainting algorithm. We extend an existing patch-based inpainting technique to consider the

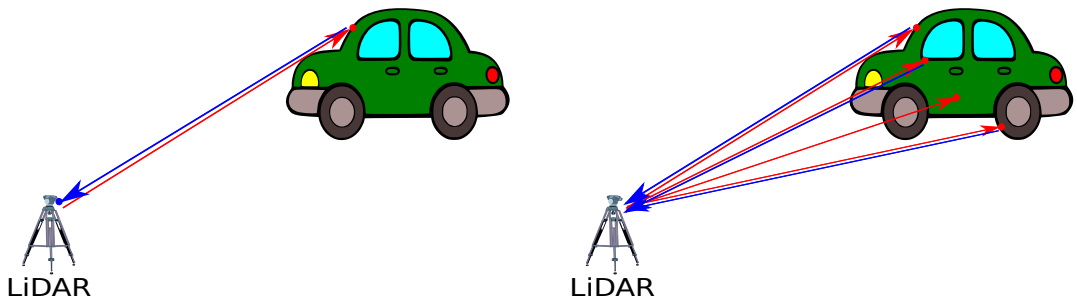
cost of filling multiple candidate patches at each iteration, leading to a procedure that performs better than traditional greedy algorithms.

CHAPTER 1
Introduction

The world around us is very complex. Until recently, the most common method to capture and analyze this world was photography. Specifically, digital photography has become pervasive over the last twenty years, allowing us to take high detail snapshots of the world and begin processing them almost instantaneously. Storing, manipulating, and examining these photographs has become a critical task in a wide range of fields, from art and architecture to archeology to robotics.

Although 2D images contain an enormous amount of useful information, they suffer from a major drawback. By the nature of the acquisition process, there is an ambiguity inherent to each pixel captured in the image — namely, any scene point on a ray in 3D space projects to the same pixel location in the image plane. We explain this problem in detail in Section 2.1.1. To overcome this ambiguity, researchers have investigated several ways of directly acquiring information about the 3D scene (detailed in Section 2.1).

The technique we are concerned with throughout this thesis is Light Detection and Ranging, or LiDAR. LiDAR is a data acquisition method which uses time-of-flight calculations of laser pulses to obtain measurements of a 3D scene. The resulting data sets contain information about a scene that is impossible to obtain with standard optical imaging, such as the absolute scale and measurements of objects, the position of heavily occluded objects, and surface texture information. A sketch of the LiDAR data acquisition process is shown in Figure 1.1.



(a) (b)

Figure 1.1: A sketch of the LiDAR acquisition process. (a) A single point acquired by the LiDAR scanner by pulsing a laser and measuring the time it takes to reflect of the scene and return to the scanner. (b) By measuring several such pulses, an image of the scene is acquired.

Using a time-of-flight calculation, the distance to the first reflecting surface in the scene in the direction of the pulse can be determined. Thousands of such laser pulses are emitted in a grid, forming an image of the scene. We refer to this as a "2.5-D" image; the full 3D structure is not known since the laser pulses cannot penetrate opaque objects to see their reverse side. In conjunction with this laser pulse measurement, modern LiDAR scanners typically include a co-located camera that can be used to assign an RGB color to each detected 3D point.

The size and resolution of data acquired by this method can be incredible. For example, in Figure 1.2 we show a LiDAR scan of a building with 1.6 million points.

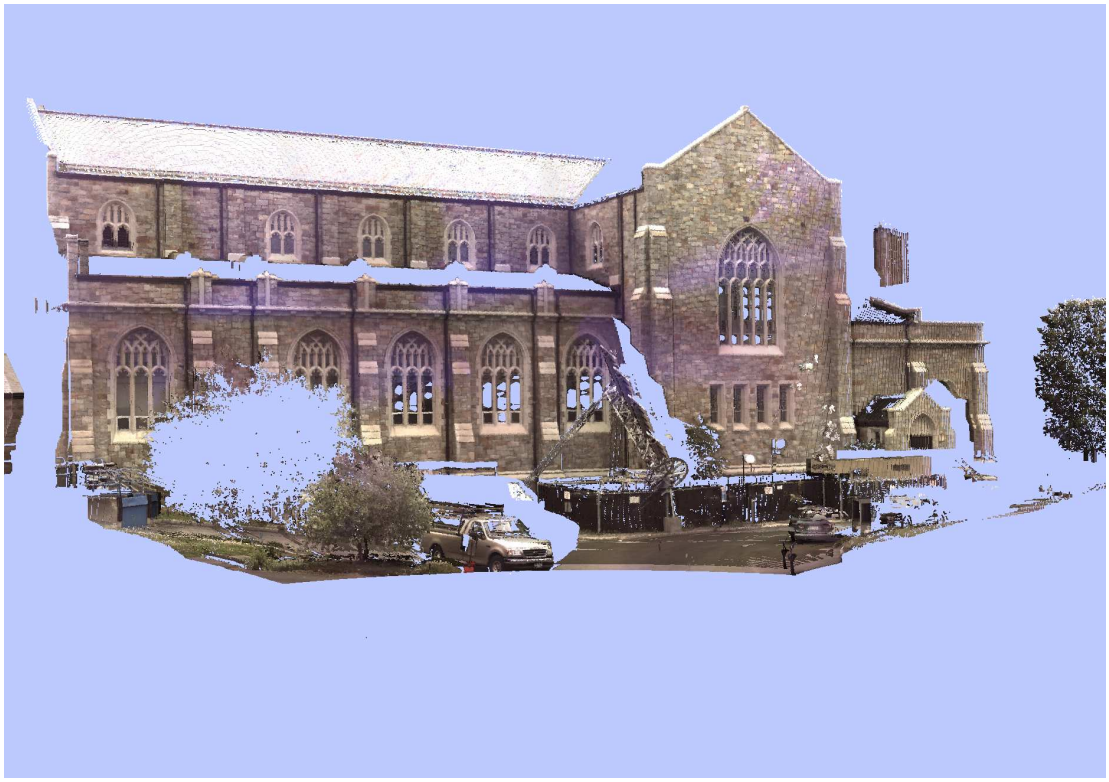
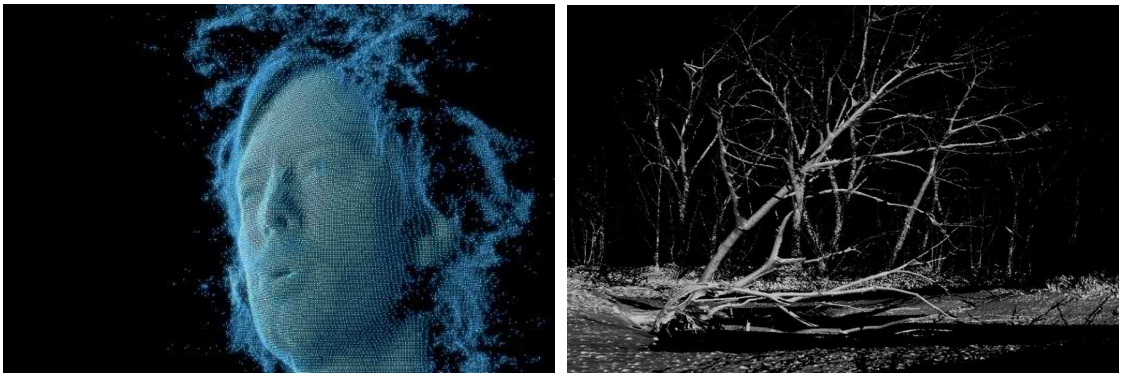


Figure 1.2: A LiDAR scan of a building. This scan contains 1.6 million colored points.

Visual effects artists in Hollywood as well as other consumer media sources have recently become very interested in 3D data acquisition techniques for scanning built sets, props, and actors' bodies and faces. In fact, an entire book [127] has been published about how to manipulate this type of data in a way that is useful for this type of user. The band Radiohead brought this type of data into view of the general public in their music video "House of Cards". This video featured video-rate LiDAR scans of the musicians performing, and 3D flythroughs of a suburban environment. A frame from the video is shown in Figure 1.3a. Modern artists also use LiDAR directly in their work. During this thesis, we worked with artist Sophie Kahn to obtain scans which were post-processed into art pieces. One such piece is shown in Figure 1.3b.



(a) (b)

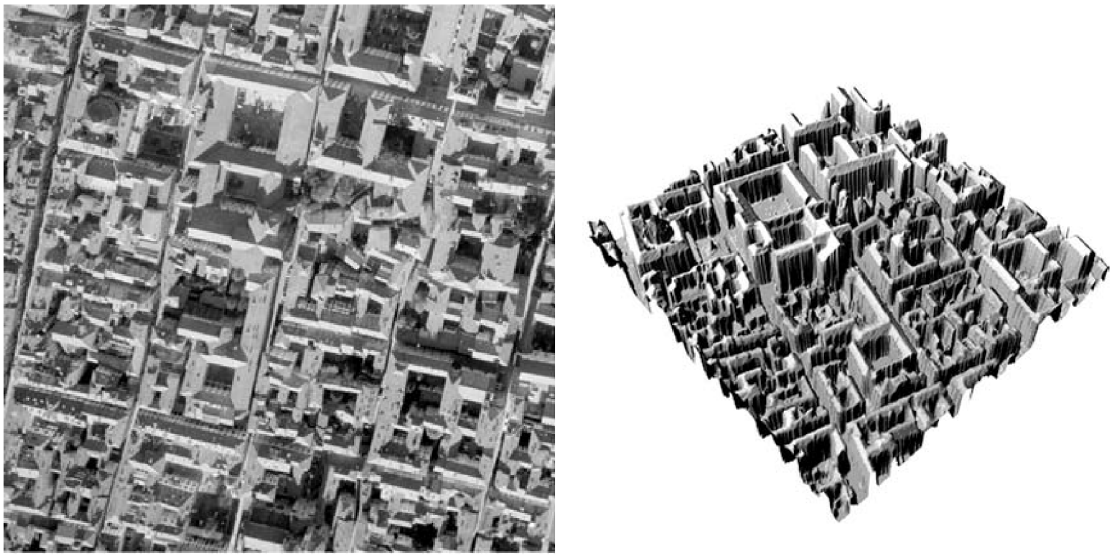
Figure 1.3: Non-technical uses of LiDAR. (a) A frame form Radiohead's music video "House of Cards". (b) a piece of art made from a LiDAR scan.

While LiDAR has enabled interesting effects in the arts, it has become even more important to technical fields that rely on very accurate data to make mission critical and potentially life-saving decisions. Modern robotics relies heavily on 3D data for navigation, object detection, and decision making. Civil engineers use 3D data to construct and analyze very accurate models of the terrain on their build sites, as well as their prospective buildings. Architects can digitize 3D sculptures and mock-ups and insert them into CAD-type programs to see how they will look and feel. Archaeologists use LiDAR as a tool for preservation and analysis of historical sites and artifacts. LiDAR data allows for analysis of the tool marks produced by sculptors, as well as the use of computer algorithms to reassemble the pieces of broken artifacts (ceramic pottery, etc.) like a jigsaw puzzle. A popular example of this type of work is the Digital Michaelangelo project [105], in which researchers performed extremely high resolution scans of the Statue of David for historical preservation and analysis.

LiDAR data has also become extremely important to the military. It is used for mission planning, reconnaissance, intelligence gathering, as well as for creating extremely accurate simulations of battlefield phenomena such as helmet deformation and vehicle armor performance. Additionally, a non-obvious military use of LiDAR is to study the composition of clouds. This allows the creation of highly accurate local weather models and forecasts, which can be used to precisely time covert missions.

The type of data we are interested in throughout this thesis is referred to as "ter-

restrial LiDAR", meaning that it was acquired from a scanner on the ground. However, a second technique of performing LiDAR scans is to attach the scanner to a plane or drone and fly over regions to produce 3D models of terrain and buildings. These scans can be used for many purposes, such as topographic studies, urban planning, or military mission planning. A typical aerial scan is shown in Figure 1.4.



(a) (b)

Figure 1.4: Aerial LiDAR scanning. (a) An aerial image of a suburban environment. (b) An aerial LiDAR scan corresponding to the region shown in (a) (images from [71])

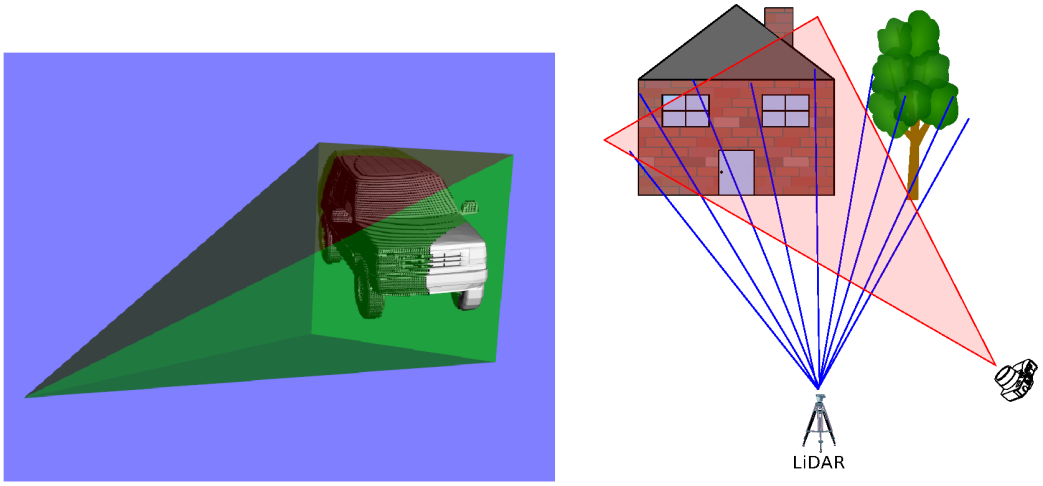
1.1 Contributions

In this thesis we study the analysis, interpretation, manipulation, and synthesis of 3D data acquired from LiDAR scanners. The work is motivated by a larger, encompassing problem of "3D Image Editing" — that is, a "Photoshop"-like tool for LiDAR scans. We are primarily interested in two main types of editing: first, the detection and segmentation of 3D objects in a complex environment, and second, the realistic filling-in of holes caused by removed objects. The same technique used to fill holes left by removed objects can also be used to fill LiDAR shadows (discussed in detail in Section 2.1.4.4), making the LiDAR data much more visually appealing and useful to a user. The work in this thesis is divided into several chapters, as described below:

- In Chapter 2, Related Work, we discuss previous work on the problems that we

address in this thesis. This chapter provides an overview of the literature related to all of the following chapters. The technical details of the prior work that is directly applied and extended in this thesis are deferred to the beginning of each chapter in which they are relevant. This allows us to provide the necessary background information and terminology immediately before it is needed.

- In Chapter 3, Custom LiDAR Tools, we outline some supporting work in the form of new software tools designed to complete the work in this thesis. The most important of these tools are a synthetic LiDAR scanner and a technique to recolor LiDAR scans using externally acquired images from a digital camera. The synthetic LiDAR scanner allows us to acquire realistic LiDAR data sets from artificial 3D object models. This enables us to fully control the situation so that we can accurately study the effects of different phenomena in LiDAR data. Our scan recoloring procedure uses computer vision techniques based on resectioning to align a digital image with a 3D scan. This is necessary as a preprocessing step in every part of this thesis, as it drastically improves the quality and alignment of the color portion of the data.



(a) (b)

Figure 1.5: A visual summary of the work in Chapter 3. (a) A synthetic LiDAR scanner. (b) Recoloring a LiDAR scan using an external image.

- In Chapter 4, Consistency and Confidence: A Dual Metric for Verifying 3D Object Detections in Multiple LiDAR Scans, we propose a registration method-independent strategy for confirming the proposed positions of objects in a scene. A dual metric, consistency and confidence, is computed to produce a human-interpretable metric of the certainty that an object has been correctly identified. The consistency measure uses a free space model along each scanner ray to determine whether the observations are consistent with the hypothesized model location. The confidence measure collects information from the model vertices to determine how much of the model was visible. The metrics do not require training data and are more easily interpretable to a user than typical registration objective function values. We demonstrate the behavior of the dual measures in both synthetic and real world examples.

(a) (b)

Figure 1.6: A visual summary of the work in Chapter 4. Our new metrics allow a user to be certain of a correct object detection (a) and identify incorrect detections such as (b).

- In Chapter 5, LiDAR Segmentation, we propose a two step technique to interactively segment entire objects from the background in LiDAR scans. We treat the LiDAR scan as a 4-channel image — the associated 3-channel RGB image, with the depth image channel appended. This allows us to treat the problem with existing image graph-cut segmentation techniques. Using a graphical interface, a user can mark as little as two strokes in the image, one on the object of interest

and one on the background, and separate an entire object from the background. The interface also allows the user to interactively refine the resulting segmentation if necessary. Often, attempting to segment a color image alone is a difficult problem. Likewise, attempting to segment the depth image alone usually does not produce satisfactory results. We present an algorithm which takes advantage of the best qualities of depth image segmentation while combining them with the best qualities of color image segmentation to obtain an accurate full-object segmentation very easily. Our main observation is that segmenting the depth image alone can provide an excellent segmentation in some regions of the object, but has trouble in other regions, such as the attachment point of an object to the ground. Our goal is to use these clean depth segmentations in the well-behaved regions of the object, and then refine the segmentations using color where we are less confident. We propose a two step algorithm. First, a conservative estimation of the object is computed using the depth image alone. In our experiments, this procedure has never resulted in labeling a background point as the object, but it also never separates the entire object. We then use this result as the initialization to a color image segmentation step that refines the segmentation in the ambiguous areas while preserving the sharp object boundary obtained by the depth segmentation. This provides a significantly more complete description of the object for the second segmentation than simple user scribbles (the typical input to an object segmentation algorithm) provided alone. The result of the two step segmentation is an accurately segmented object without the need for training data. We demonstrate this technique in several real-world data sets.

(a) (b) (c)

Figure 1.7: A visual summary of the work in Chapter 5 (a) User provided strokes on the foreground (object) and background. (b) The 3D scene. (c) The segmented object.

- In Chapter 6, LiDAR Inpainting, we study the problem of filling a large hole in LiDAR data. These holes occur in two main cases. First, using a technique such as the one we proposed in Chapter 5, object can be selected and removed from a scene. Second, since the LiDAR laser cannot penetrate opaque objects, a phenomenon referred to as a "LiDAR shadow" appears behind every object in the scene. These shadow holes are present in nearly every real-world scan. We present an algorithm to synthesize, or hallucinate, data in these types of holes that looks like a plausible representation of what could have been present in the scene. The major benefit of filling this type of hole is to make a scan viewable from a position other than the original acquisition viewpoint, which is critical for data exploration. Our algorithm combines and extends existing gradient-domain image editing techniques and greedy patch-based inpainting techniques. We copy depth gradient patches intelligently from elsewhere in the image into the hole and then reconstruct the scene structure by solving a variational problem resulting in a Poisson equation. We present several real-world examples of this technique with excellent results.

(a) (b)

Figure 1.8: A visual summary of the work in Chapter 6. (a) A LiDAR scan with a large LiDAR shadow. (a) The inpainted LiDAR scan using our algorithm.

- In Chapter 7, Post-Candidacy Work, we discuss the work to be completed post-candidacy. First, we introduce an additional step in the patch-based inpainting algorithm which we refer to as a "patch acceptance test." The concept is very similar to our work in Chapter 4, as it provides a search-technique independent way of determining if the proposed "best patch" is actually a good patch to copy at every location. We have observed that highly discriminative operations are far too computationally intensive to perform at every candidate source patch during the patch-search stage of the algorithm. Instead, we search for a small set of candidate patches using a simple and fast sum of squared differences method, and then perform intensive computations on each of these candidates to determine if it should be accepted or not. The user is prompted to take an action such as selecting a different patch from a list of top candidates, or manually selecting a better patch from the image. Second, using this idea of acceptance tests, we make the inpainting algorithm less greedy by searching for matches to multiple candidate target patches simultaneously. We will show that while only a few bad patch matches may be used in each inpainting problem, by eliminating these errors the results can be greatly improved.

(a) (b)

Figure 1.9: A visual summary of the work in Chapter 7. (a) An analysis of erroneously good patch matches. (b) An improvement to patch-based inpainting that considers multiple target patches simultaneously.

1.1.1 Publications

To date, the material in Chapter 4 has been published in the Proceedings of 3-D Digital Imaging and Modeling (3DIM) 2009 , held in conjunction with the International Conference on Computer Vision (ICCV) [62]. The material in Chapters 5 and 6 is in preparation for ACM Transactions on Graphics with a submission goal of May. A more detailed version of the material in and Chapter 6 has been submitted to the International Workshop on Point Cloud Processing 2012, a workshop in conjunction with Computer Vision and Pattern Recognition (CVPR). The material in Chapter 3 has been summarized in several technical reports in online journals an magazines, including the Insight Journal [46, 60, 56, 57, 55, 58], Visualization Toolkit (VTK) Journal [45, 51, 50, 52, 61, 53, 47, 49, 48, 59, 54] , and Kitware Source Magazine [42, 44, 43].

CHAPTER 2
Related Work

In this chapter, we discuss prior work in the areas that we extend in this thesis. This chapter is broken into sections corresponding to the following chapters in this document.

2.1 3D Data Acquisition

In this section, we describe several techniques for obtaining 3D information from a *scene*, or a 3D environment in which we live. We start by explaining why traditional digital images are not sufficient for 3D data acquisition and analysis. We then describe several techniques for both indirectly and directly estimating and measuring 3D scene points.

2.1.1 Optical Image Acquisition

While images contain an enormous amount of information, they suffer from a major drawback. By the nature of the acquisition process, there is an ambiguity inherent to each pixel captured in the image. Namely, we cannot be sure which 3D point in the scene generated each pixel in the image. In fact, there is an infinity of points (a ray) which could have produced every image pixel. In Figure 2.1, we show two 3D points, P' and P'' which both could have produced the observed pixel p in the image.

2.1.2 Passive 3D Data Acquisition - Reconstruction from Images

In many cases, *stereo vision* algorithms can be used to estimate 3D information about an environment. With two images of the same scene, we can estimate 3D structure in the scene. The heart of the problem is finding *correspondences* in the two images. That is, for a pixel in the first image, we must determine which pixel in the second image was produced by the same 3D point. If these correspondences can be established, reconstructing the 3D scene is a straightforward mathematical procedure. However, identifying these correspondences unambiguously can be challenging. The

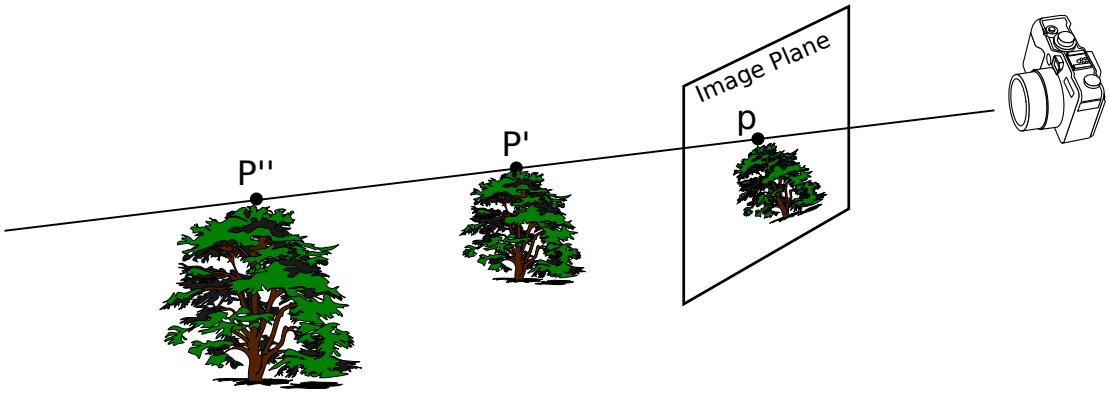


Figure 2.1: The problem with obtaining 3D information from a single image. Both 3D points P' and P'' could have produced the image point p.

fundamental problem is locating and matching small patches of pixels in the two images which can be difficult in low texture areas (we discuss this problem in depth in Chapters 6 and 7). There are some constraints induced by the camera configuration (the *epipolar geometry*), but the problem is still prone to error.

An extension of the stereo vision problem, *multi-view stereo*, relies on multiple (tens to hundreds) cameras capturing the same instant of a scene from varying viewpoints. An equivalent effect can be achieved by using only one camera and moving it relative to the scene (a video sequence). This procedure typically works on the assumption that the scene is static (does not change from frame to frame). Large sets of images of the scene taken from significantly varying viewpoints must be available for the process to work well. A survey of multi-view stereo techniques is presented in [137]. Excellent results have been obtained with an algorithm called Patch-based Multi-View Stereo (PMVS) [77], as shown in Figure 2.2.

In recent work [4, 3], Agarwal et al. demonstrated the state of the art of these techniques by attempting to reconstruct the city of Rome from a massive collection of tourist photographs. Thousands of images were used to create the reconstruction of Trevi fountain, shown in Figure 2.3.

(a) (b)

Figure 2.2: A 3D reconstruction of a solider using the PMVS algorithm. (a) An image of a statue of a solider. (b) A 3D reconstruction of the soldier from 48 images. (Images from [77])

Figure 2.3: A 3D reconstruction of Trevi fountain from thousands of images. The locations of the cameras as shown as black pyramids. (Image from [4])

While the 3D information reconstructed via these techniques can look good in many cases, the accuracy of the reconstructed scenes are not as good as those obtained directly by active 3D acquisition methods (discussed in Section 2.1.3). These methods suffice when the data needs to be gathered quickly and the data sets produced will be used for aesthetic purposes, but when the accuracy of the scans is critical, active techniques are highly desirable. In this thesis, we are interested in algorithms working on large-scale, highly accurate 3D information which is difficult to obtain with passive methods.

2.1.3 Active 3D Information Acquisition

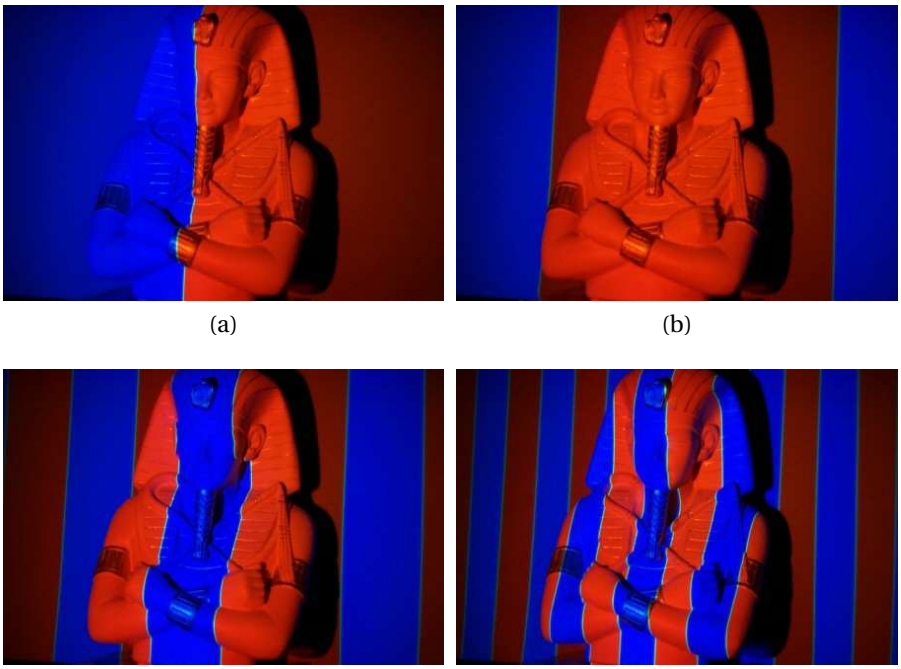
2.1.3.1 Structured Light Scanning

Structured light scanning is the process of projecting known patterns of light into a scene, taking images of the resulting overlap of the patterns with the objects in the scene, and then computing the underlying 3D geometry from these images. Typically several different sequences bars are projected into the scene sequentially, as shown in Figure 2.4.

The way that the pattern deforms when projected onto objects in the scene can be interpreted to compute the depth of points in the scene. These patterns can also be projected using infrared wavelengths, so that they are not visually distracting to a human observer of the scene. The major drawback of this acquisition method is that the size of the scene that these patterns can be projected onto is typically quite small ($< 10m$). However, the results can be quite good, as shown in Figure 2.5

2.1.3.2 Time-of-Flight (TOF) Cameras

A time-of-flight camera is a device which projects infrared light into a scene and computes a depth measurement at each pixel in a sensor array simultaneously. These devices are often inexpensive, but their low resolution (usually $< 200 \times 200$ pixels) and low accuracy are limiting factors in their usefulness to large scale 3D data acquisition. However, the devices have some very appealing properties such as their small physical size (the Mesa Swiss Ranger 4000 shown in Figure 2.6 is only about 3 inches cubed) and their ability to capture 3D data at video frame rates.



(a) (b) (c) (d)

Figure 2.4: A scene with multiple light patterns projected on a statue in a temporal sequence. (Images from [129])

This technology has recently been brought to the public eye by Microsoft, as a similar concept drives their Kinect system. Amazingly, the Kinect sells for only $100, allowing this technology to rapidly develop new and interesting uses by the public.

For detailed information time-of-flight acquisition, we refer the reader to [100, 38].

2.1.3.3 Light Detection and Ranging (LiDAR)

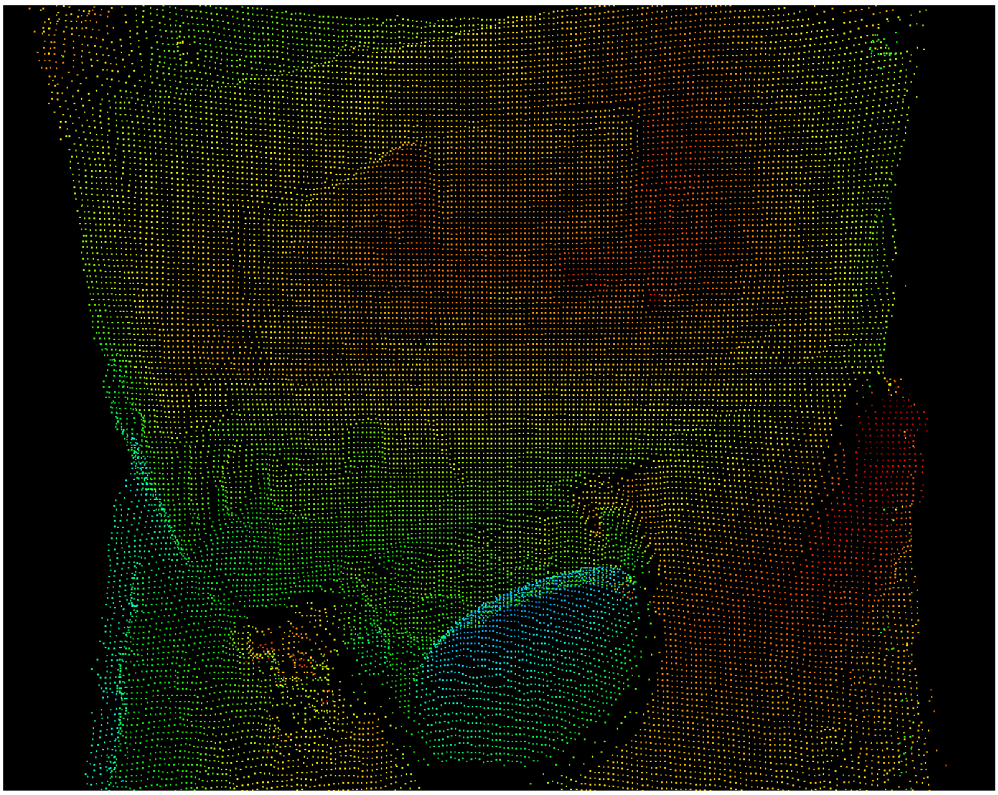
Light Detection and Ranging (LiDAR) is a data acquisition method which computes the distance to 3D points by measuring reflected laser light projected into the scene. Two technologies are popular in commercial systems. The first uses time-of-flight calculations on laser pulses to obtain the 3D point samples. The system pulses the laser, waits for it to return to the sensor after reflecting off of the scene, and computes the distance to the reflection point by dividing the time the pulse took to return to the scanner by $2c$ (where c is the speed of light, and $2c$ because the time measured was a round trip from the scanner to the point.) The second technology uses the phase

Figure 2.5: The resulting mesh after scanning a sculpture of a head with a structured light system. (Image from [129])

difference between the emitted pulse and the reflected pulse to determine the distance to the 3D point in the scene. In both cases, a LiDAR scanner sends out thousands of such pulses, creating a "2.5-D" image of a scene. We do not refer to these scans as "full 3D" because the LiDAR laser cannot penetrate opaque objects to see their reverse side. Data sets obtained using these methods contain accurate 3D information about a scene that is impossible to obtain with standard optical imaging, such as the absolute scale of objects, the position of heavily occluded objects, and texture information.

In this thesis we are concerned with *scanning LiDAR* scanners. This type of scanner works by moving internal motors and mirrors to direct the laser into the scene. A mirror sweeps the laser pulses (by varying its angle) across the scene acquiring a "strip" of points, then the mirror is turned slightly and then swept across the scene again, acquiring an adjacent strip of points. This process is continued until the scan is complete. This type of scanner is highly accurate (the Leica HDS3000, for example, has 6 mm accuracy at 50 meters). They typically also have a very long range (100+ me-

(a)



(b)

Figure 2.6: (a) A Mesa SR4000 Time-of-flight camera. (b) A scan of an office cubicle produced by the SR4000. We note that it is low resolution, and quite noisy in many areas.

ters). Some scanners have multiple (up to 64) sensors in a linear array, and this array is physically rotated around the device. This method allows the scene to be captured much faster. For example, the Velodyne HDL-64E scanner can scan at up to a 15 Hz

frame rate, which results in acquiring over 1.3 million points per second. This type of scanner is often mounted on autonomous vehicles, such as DARPA Grand Challenge vehicles, as shown in Figure 2.7.

Figure 2.7: The winning vehicle of the DARPA Grand Challenge. Five LiDAR scanners are mounted across the top of the vehicle.

The LiDAR scanner that we used to acquire the data sets seen throughout this thesis is the Leica HDS3000 scanning LiDAR scanner, shown in Figure 2.8.

A very useful property of LiDAR scanners is their ability to "see through" foliage. As shown in Figure 2.9, since each laser pulse travels on an independent path through the scene, some of the pulses will hit a tree, while others will find small gaps between the leaves and hit the objects behind the tree. Although the points behind the tree are sparse, they can still be used to detect objects. This is in contrast to an image of the scene, which would likely not show any detectable traces of the existence of the object.

2.1.3.4 Spherical Grid Acquisition

In this thesis, the LiDAR scanner used scans the scene in a spherical grid. That is, the motors in the scanner aim the laser through imaginary, uniformly spaced cells on sphere to capture the 3D points. The sampling in the θ (azimuth) and ϕ (elevation) directions need not be equal. This process is shown in Figure 2.10.

Figure 2.8: The Leica HDS3000 LiDAR scanner, used to acquire all of the data sets in this thesis.

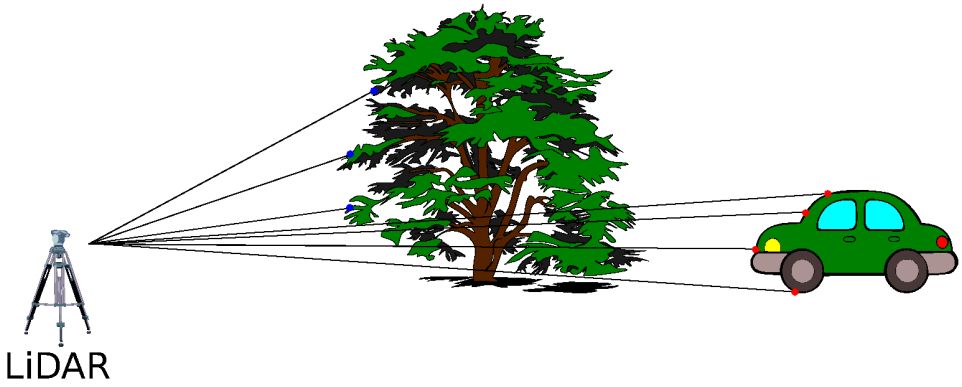


Figure 2.9: A demonstration of the foliage penetration capability of LiDAR. The blue dots show points that were acquired from the occluding object, while the red dots show points which reached the object of interest. A photograph from this same viewpoint may not have shown any, or extremely limited, traces of the car.

It is because of this grid that we can alternatively view the points as a "depth image." A depth image is displayed as a flat grid of pixels, the values of which are the distance from the scanner to the corresponding point. These pixels are float-valued, so to display them they must be pseudo-colored, as shown in Figure 2.11.

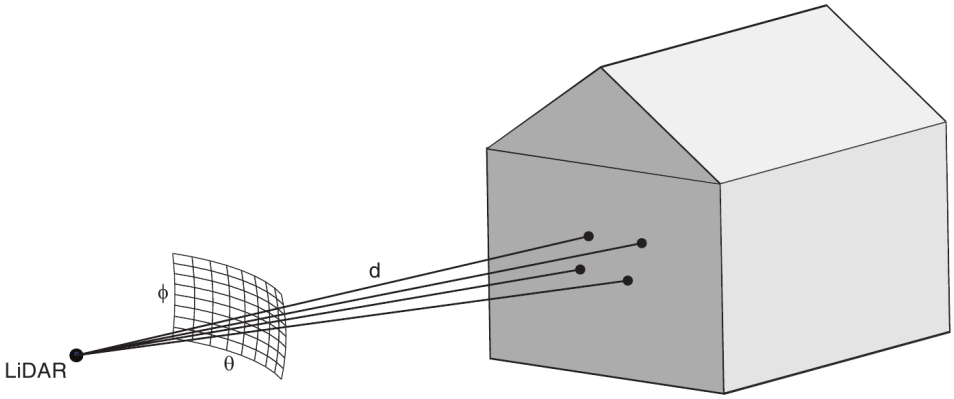


Figure 2.10: A sketch of the spherical grid acquisition process of many LiDAR scanners. (image from [127])

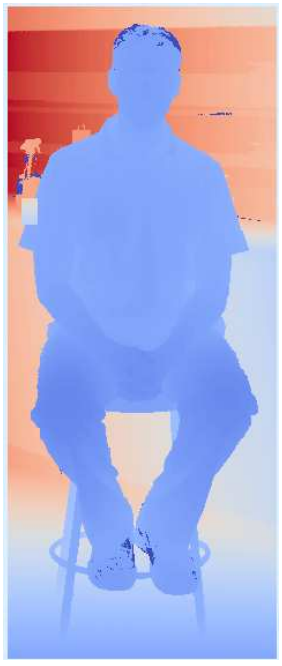


Figure 2.11: A depth image of a man sitting on a stool. Dark blue points are closest to the scanner, while dark red points are farthest from the scanner.

2.1.4 Problems with LiDAR

While LiDAR has many positive qualities, it is not perfect. This section details some of the problems that can arise in LiDAR data.

2.1.4.1 Bulkiness

While scanners are often vehicle mounted, many are tripod mounted. All of the data in this thesis was acquired using the Leica HDS3000 scanner. This scanner requires a car-battery sized power source, a surveyor's tripod, and a laptop to operate. Moving this bulky setup into position can be laborious and time consuming. A typical setup is shown in Figure 2.12.

Figure 2.12: The setup of a typical scanning session with the Leica HDS3000.

2.1.4.2 Point Density Variability and the Glancing Angle Problem

As the distance to the surfaces in the scene is unknown a-priori, the resulting points acquired in a uniform spherical grid do not uniformly sample the scene surfaces. Even points that are on a surface perpendicular to the scanner will have different depths, as shown in Figure 2.13.

Additionally, not all surfaces are sampled equally densely, especially those at an oblique angle relative to the scanner. This phenomenon is shown in Figure 2.14.

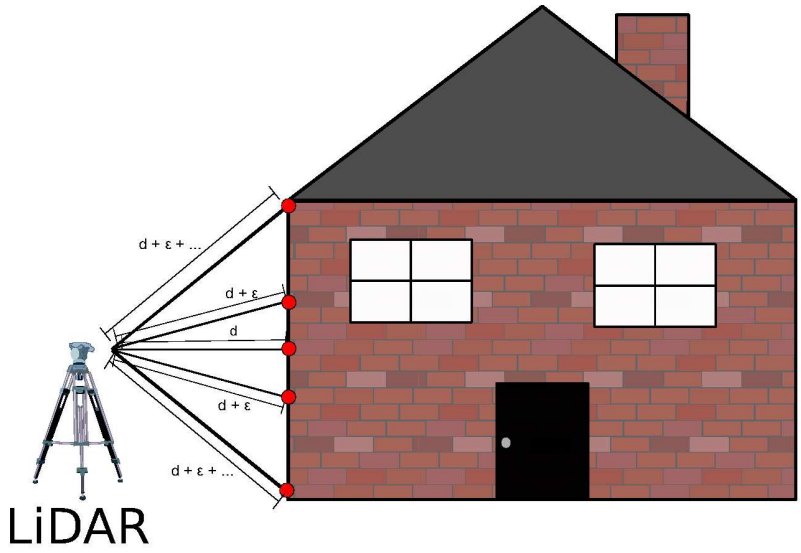


Figure 2.13: The variable point spacing of a LiDAR scan.

(a) (b)

Figure 2.14: The glancing angle problem. (a) A scan of a mailbox and a building. (b) We see that the flat surface at the base of the mailbox was seen by a very small number of LiDAR rays compared to other surfaces in the scene.

2.1.4.3 Thin Objects

An optical camera accumulates the contribution from many points in the world to produce each pixel. On the other hand, a LiDAR pulse approximates seeing a single

point in the scene. Because of this, thin structures can be missed entirely. A common and important example of this problem is missing power lines in a scene. In Figure 2.15, we show a scene in which data interpolation has been applied to detect and fill in these missing power lines.

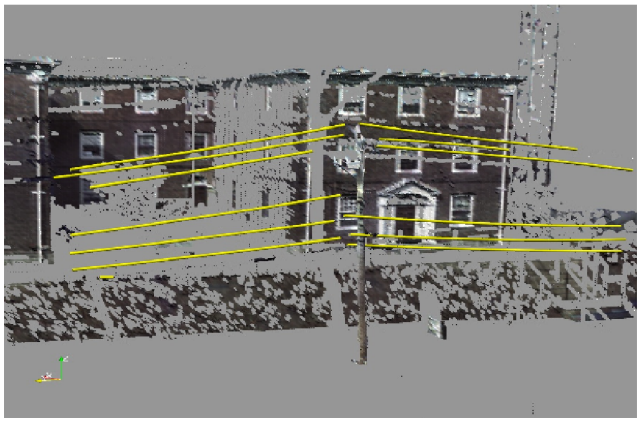
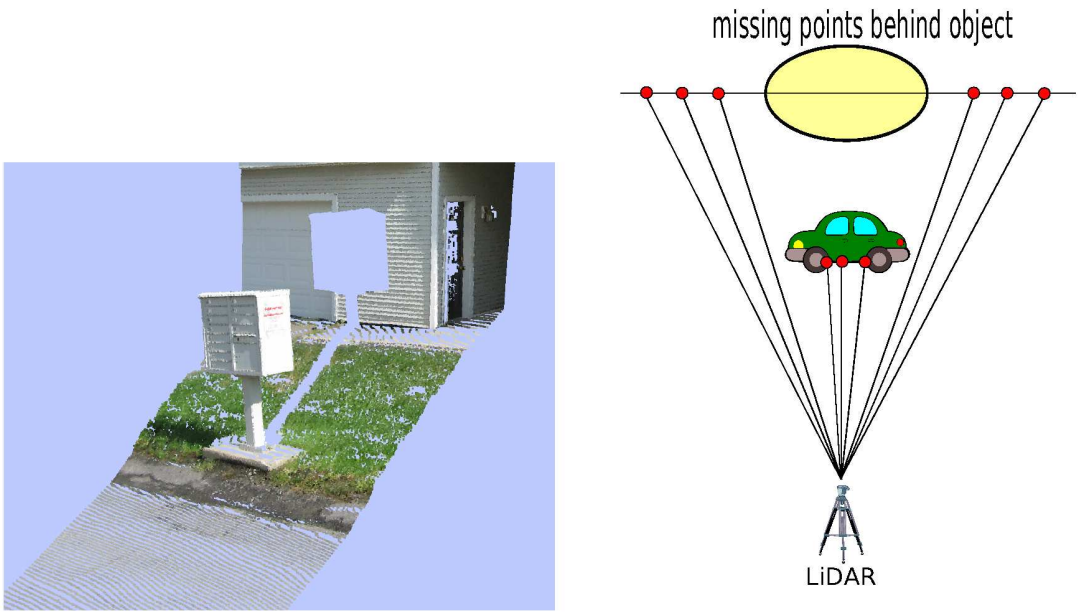


Figure 2.15: Interpolated power lines.

2.1.4.4 LiDAR Shadows

The problem of *LiDAR shadows* is of particular note, as it is the focus of our contribution in Chapter 6. When a LiDAR scan is taken of an opaque object in front of a background, the laser cannot go through the object, so the scene behind the object is not observed. When the scan is viewed from a viewpoint other than the one from which it was acquired, it leaves a hole, or shadow, in the scene as shown in Figure 2.16.



(a) (b)

Figure 2.16: The shadow left behind an object in a LiDAR scan. (a) A real example of a LiDAR shadow in a LiDAR scan. (b) A sketch of the situation which causes the shadow to occur. Here we show a scan of a car with a wall in the background. We see that there are no red dots (LiDAR returns) behind the car, as the laser reflected off of the car before reaching the wall.

If this missing data is required to be known, there is no other choice than to observe and record the scene from a different viewpoint. Even if this area is not of particular interest, it is still distracting to see large holes in the scene when inspecting other regions.

2.1.4.5 Invisible Surfaces/Missing Returns

Due to their surface properties, some objects are poor candidates for scanning with LiDAR. These objects include glass (often found in urban scans in the form of car and building windows), black surfaces (again, many vehicles have this property), and other reflective surfaces (metal signs, etc). The laser never returns to the scanner because it reflects most of its energy in a different direction, so the data cannot be recorded.

For example, when scanning vehicles, the painted surfaces are very reflective so there are many missing returns, and the windows are entirely invisible, as shown in Figure 2.17.

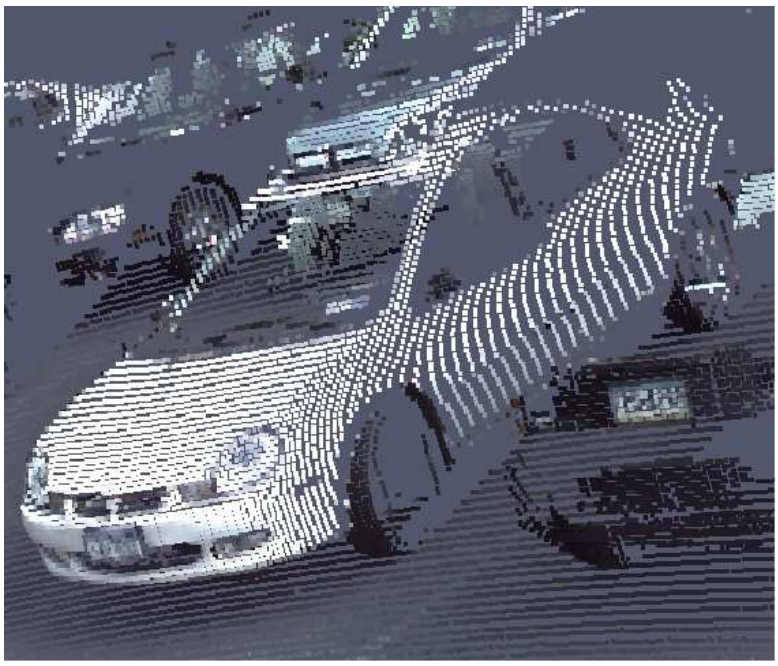


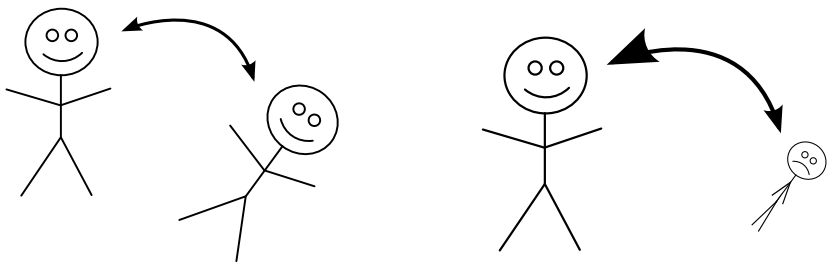
Figure 2.17: Missing points in the glass windows and metal wheels of a car. When the LiDAR laser encounters these reflective surfaces, its energy is reflected away from the scanner, rather than returning to the scanner in the case of non-reflective objects.

In this section we have outlined several methods of data acquisition. In the remainder of this chapter, we review techniques and algorithms for analyzing this type of data set.

2.2 Registration

Registering, or aligning, two or more objects is a common problem across a variety of fields. While registration is common in image processing, we restrict the discussion in this section to techniques that apply to 3D data. The problem can be described finding the best transformation to align one object with another. By *objects* in this discussion, we mean 3D meshes or point clouds. This seemingly benign definition of the problem has two major caveats. First, the definition of "best" usually domain specific,

and it may be difficult to find a mathematical formulation that agrees with human intuition. Second, the transformation we seek can be as simple as a 2-degree of freedom (DOF) rigid transformation, or as complicated as a very high degree of freedom deformable transformation. Figure 2.18 shows two problems that, although appearing quite similar, are actually extremely different in complexity.



(a) (b)

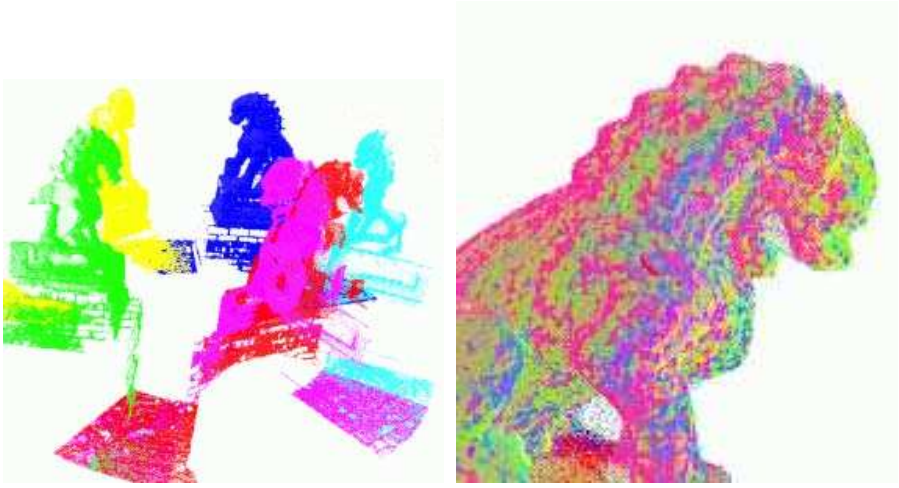
Figure 2.18: Two similar registration problems with very different levels of complexity. (a) A relatively easy registration problem. A 2-DOF rigid transformation can exactly align the two stick figures. (b) A very difficult registration problem. A complex deformable transformation or articulated model would be necessary to align the two stick figures.

One use of registering point clouds is to combine multiple data sets into a more complete representation of a scene. For example, if two point clouds of the same scene are acquired from different view points, they will both be partially incomplete. By aligning the point clouds, the available data in one cloud can "fill in" the missing parts of other cloud. An example of this is shown in Figure 2.19.

There is an extensive body of work on 3D object registration. A very popular technique to register point clouds is Iterative Closest Points (ICP) [17, 164]. ICP is a two step iterative procedure. First, given an initial 3D rigid transformation between point sets A and B, for each point in A, we find its closest point in B. In Equation 2.1 we state mathematically the condition to find this closest point to the i^{th} point in A.

$$b_i = arg \min_{\forall p \in B} d(a_i, p) \tag{2.1}$$

This distance function $d(a_i, b_i)$ is often taken to be the 3D Euclidean distance between the two points. Once these correspondences are computed, the best trans-



(a) (b)

Figure 2.19: Point cloud registration. (a)Several partial point clouds a lion statue acquired from different perspectives. (b) The resulting registration of all of the point clouds. The resulting combined point cloud is a much more complete description of the statue. (images from [114])

formation between the point sets is found. In Equation 2.2, we show the minimization problem that must be solved to compute the best (in a least-squares sense) rigid transformation between the two point sets. This transformation consists of a rotation matrix R and translation vector t.

$$\min_{R,t} \frac{1}{N} \sum_{i=1}^{N} \|Ra_i + t - b_i\|^2 \tag{2.2}$$

An iterative solution to this optimization problem was originally proposed in [164] and later a closed form solution using quaternions was introduced [81]. After applying this transformation to point set B, the correspondences are re-estimated and this procedure is repeated until convergence. This algorithm is guaranteed to converge to a local minimum. However, in practice, there are many local minima, so the initial transformation from the feature correspondences must be very close to the true alignment or ICP will not improve the solution.

The correspondence search step above works well when aligning two entire objects, but researchers [164] have realized that to align partially overlapping objects, only points that have a correspondence within a specified distance threshold, D_{max}

should be used when computing the aligning transformation. That is, the correspondence search should instead be the one shown in Equation 2.3.

$$b_i = \begin{cases} arg \min\limits_{\forall p \in B} d(a_i, p) & \text{if } d(a_i, b_i) < D_{max} \\ \text{undefined} & \text{otherwise} \end{cases} \tag{2.3}$$

Our contribution in Chapter 4 is motivated by ambiguities that arise from the use of this correspondence function.

There have been many improvements to ICP. Phillips and Tomasi [125] noted that the original ICP algorithm performs poorly in the presence of outliers and addressed this problem by introducing a fractional point set distance, which accounts for outliers in the correspondence identification step. It has also been noted that distance metrics other than a point-to-point Euclidean distance can be valuable when computing the nearest neighbor from a point in one set to the other set of points. Namely, the point-to-plane distance, introduced by Chen and Medioni [33] has been widely used. Rusinkiewicz and Levoy [132] provided a rigorous performance evaluation of several ICP methods. Mitra et al.[118] generalized ICP by, instead of considering the problem as a point-set to point-set registration problem, treating the problem as aligning a point set with the surface which the target point set represents. Fitzgibbon [69] proposed an energy function which can be directly minimized and has a large basin of attraction, which is of great practical value. Finally, Specht et al. experimentally compared the ideas of registering range images directly versus registering their corresponding meshes [144]. They concluded that the performance is data set dependent, but proposed future work on combining the two concepts.

There has also been work on registering colored point clouds — the type we are interested in in this thesis. Kevin et al. [92] iteratively used color and structure information separately to obtain a final registration. Wu et al. [156] introduced a technique called Viewpoint Invariant Patches (VIP) that creates a viewpoint invariant descriptor from local shape information, allowing colored point clouds to be efficiently and accurately registered with as little as a single descriptor match. Smith et al. [143, 142] extended a very popular feature based image registration method, the *Scale Invariant Feature Transform* (SIFT, [112]) to register large, colored point clouds. Their method

constructs descriptors at points which are aware of the scale of structures in the scan, avoiding many potential false matches. An example of such a registration is shown in Figure 2.20.

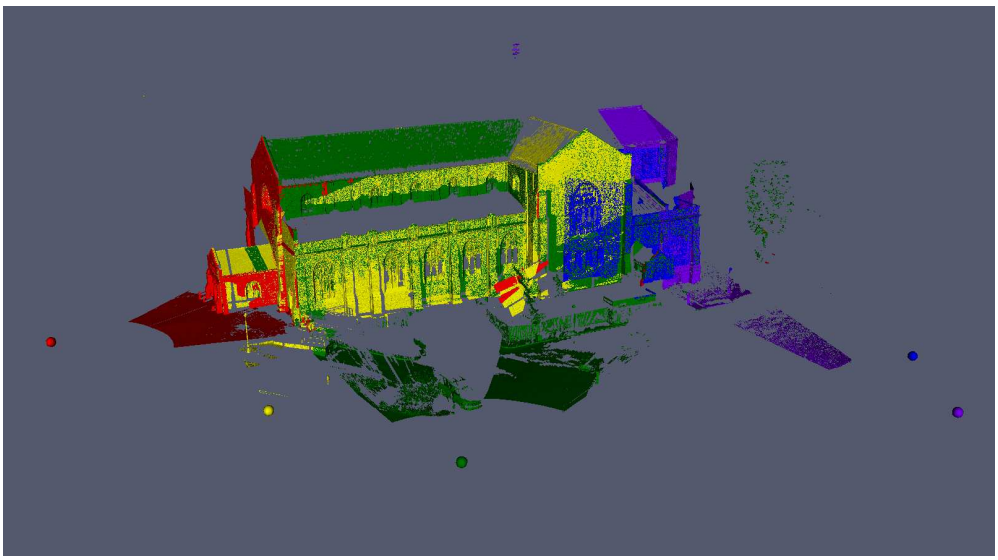


Figure 2.20: The registration of several very large LiDAR scans of a building to produce a more complete building model. The position of the scanner that acquired each scan is represented as a sphere colored to match the corresponding scan points. (image from [143]).

2.3 3D Object Detection

Object detection is a very important step in many applications in computer vision and robotics. Given a 3D mesh or point cloud, the *query object*, the goal is to determine where in a *target scene* the object exists, if at all. To motivate this problem, in Figure 2.21 we show a LiDAR scan with several cars that have been detected.

In Chapter 4 of this thesis, we present a technique to verify that the resulting position of a detected object is actually correct. As our contribution is independent of the object detection method that was used to propose an object position, we are not specifically interested in the inner workings of specific object detection algorithms, but we describe the basic ideas here as it will ease our later discussion.

Before we begin, we note that Bravo and Farid [26] analyzed the effects of clutter and orientation on the ability and accuracy of a human observer to recognize objects

Figure 2.21: Several cars detected in a LiDAR scan. (image from [123])

in images. Their finding was that this problem is indeed very hard, even for a human. In this section, we describe algorithmic methods to perform such detections.

There are three broad steps in most object detection processes. First, we generate some type of *descriptors* for both the query object and the scene. Next, we perform a matching procedure that attempts to identify correspondences between descriptors in the query object and the target scene. Finally, we use the descriptor correspondences to compute a transformation that aligns the model with its detected position in the scene. In the following sections, we discuss each of these steps in more detail.

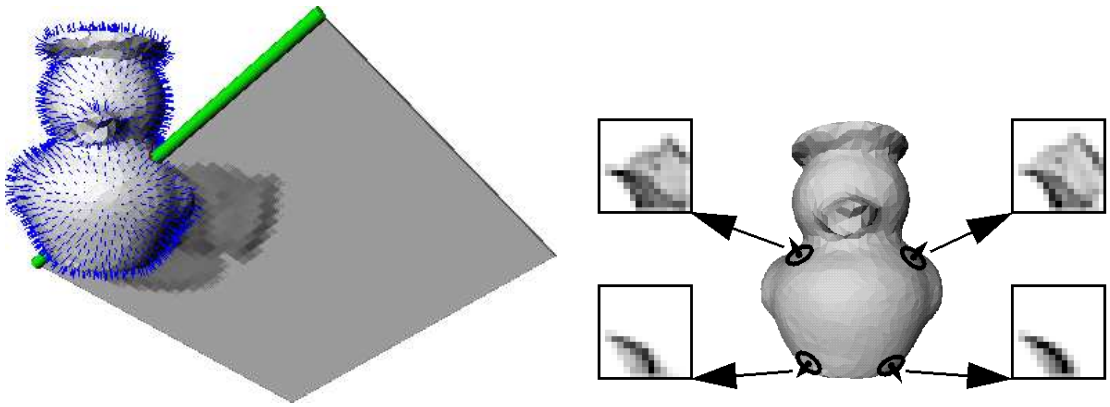
2.3.1 3D Descriptors

The purpose of a descriptor is to, as its name suggests, describe something about an object. Descriptors can be either local or global. Local descriptors describe something about a small piece of an object (such as the curvature at a specific point), while global descriptors attempt to describe the entire object simultaneously. In either case, the descriptors can then be compared to other descriptors of the same type, typically as a simple difference operation in the vector space that the descriptors occupy.

2.3.1.1 Local Descriptors

Local descriptors are computed at particular points of a scene or model using information from points immediately surrounding the query point. For example, in-

formation like the point's surface normal, the curvature near the point, or histograms of the relative positions or neighboring points can be used to describe the point. To provide a concrete example of such a descriptor, we will explain a very popular local descriptor known as the *spin image* [87, 86]. This feature is well named, as it is constructed by "spinning" a half-plane around the axis described by a point's surface normal, constructing an image of the intersections of points with this plane. This procedure and descriptor are shown in Figure 2.22. In Figure 2.22a, the surface normal has been computed at a point on a 3D model of a duck and is indicated as a green cylinder. The half-plane (gray rectangle) is then rotated around this cylinder, collecting points in bins defined on this plane as it rotates (the dark gray "pixels" on the half-plane). Figure 2.22b shows the resulting descriptor at four specific points on the model. We see that the two points on top generate very similar looking descriptors, because locally the model looks very similar at these two points. Conversely, the two points on the left produce very different descriptors, as the model surface behavior is very different at the two points.



(a) (b)

Figure 2.22: The spin image descriptor. (a) A model of a duck showing the construction process of the spin image descriptor at a point. The gray plane is rotated around the green axis, constructing a cylindrical histogram of the positions of the points that it hits. (b) Four resulting spin image descriptors at different points on the duck. We see that the two points on the left produce very different descriptors, as the model surface behavior is very different at the two points. However, the two top points have very similar descriptors, even though they are on different sides of the model. This symmetry of the descriptor over the model leads to complications in the matching process. (images from [86])

Brusco et al. [27] extended this descriptor to include color information corresponding to the points. Other popular local descriptors include 3D Shape Contexts [98, 73], and Point Feature Histograms [133, 134]. We refer the reader to these sources for specific details of each descriptor.

Once several local descriptors have been matched (we denote the number of matched descriptors N_d), the transformation that takes the model to its correct position in the scene can be computed using the matching model points m_i and scene points s_i as shown in Equation 2.4.

$$\min_{R,t} \frac{1}{N} \sum_{i=1}^{N_d} \|Rm_i + t - s_i\|^2 \tag{2.4}$$

The idea is to find the transformation (R, t) (a rigid transformation in this case, described by a rotation matrix R and translation vector t) that minimizes the distance between the transformed model points and their corresponding scene points. More flexible transformation models can be used (affine, etc.), but the problem becomes significantly harder as more degrees of freedom are added to the transformation.

Computing and matching a descriptor at every point in the model or scene is extremely computationally prohibitive. Because of this, several attempts have been made to speed up this local descriptor computation and matching. Carmichael et al. [30] used a coarse-to-fine approach, first computing descriptors on a downsampled version of the model and scene, and then moving to higher resolution versions as the algorithm progresses. Carmichael and Hebert [29] introduced a method which uses Bayesian classification to perform a first pass through a large scene to identify points that could potentially belong to the object, which significantly reduces the search space for any chosen detection method. Matei et al. [116] detected automobiles in highly cluttered, real-world scenes by speeding up the feature matching using an approximate nearest neighbor algorithm.

Unfortunately, if even one of these matches is incorrect, the computed transform will also be incorrect. To remedy this, we attempt to ensure that the matches are geometrically consistency. Consider a case where a feature at point A of the object matches well to a point B in the scene, and a point C on the object matches well to a point D in the scene. If the transformations from A to B and C to D are similar, we are

much more confident that we have actually found two correct matches. A sketch of the problem that is addressed by ensuring geometric consistency is shown in Figure 2.23.

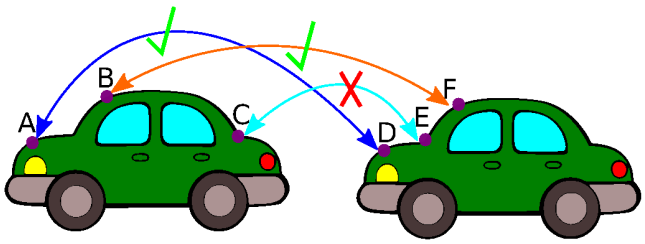


Figure 2.23: Diagram showing geometric consistency.

By using a method such as Random Sample Consensus (RANSAC [68]), subsets of these local descriptor matches can be used to determine the transformation, and the transformation that is most consistent over these subsets is selected as the final transformation.

2.3.1.2 Global Descriptors

An alternative approach to object detection is using global descriptors. That is, rather than only using local information to compute a descriptor at several points on the object, the entire object is used to generate a single descriptor that describes it in its entirety. One example of a global descriptor is the Extended Gaussian Image (EGI) [82]. Horn introduced the EGI as a description of a shape that is constructed by placing point masses on a unit sphere corresponding to every model point. These masses have a value proportional to the curvature of the object at the model point, and are placed on the sphere where the normal of the sphere is equal to the normal of object surface at the object point. Following this construction, we see that EGI's encode the entire shape of the object. EGI's have the nice property that they are translation invariant, and rotationally well behaved, as the EGI rotates exactly as does the object. As such, if we can align the EGI's, the alignment of the object is immediately obtained. Matching these global descriptors can be performed efficiently in the frequency domain, as demonstrated by Makadia et al. [114], allowing for fast object detect in large scenes.

Another global descriptor is Eigenshapes [28]. In this technique, range images of an object are constructed by projecting the points onto several different planes (mul-

tiple viewpoints). Each of these range images is vectorized and the set of vectorized range images are concatenated to form a matrix. The eigenvectors of this matrix have been shown to be an excellent description of the object. By matching these eigenvectors, we can determine which views, if any, of an object are present in a scene.

2.3.1.3 Composite Descriptors and Other Techniques

Researchers have found that combinations of local and global descriptor methods can outperform either technique alone. For example Patterson et al. [123] combined the local and global approach by first using spin images to detect candidate object positions, and then used EGIs to verify these positions.

In addition to local and global shape descriptors, researchers have used a probabilistic approach based on free-space models and an "occupancy grid" to determine possible positions of objects in a scene [161]. While theoretically sound, these methods are typically too slow to use to detect objects in real-world scenes.

2.3.2 3D Object Detection Verification

Upon completion of an object registration algorithm, a natural question to ask is "how well did we do?" Several researchers have mentioned in passing that they have performed some sort of verification at the end of their algorithm, but to the best of our knowledge there has not been a study entirely dedicated to this very important step. More importantly, often the verification procedure that is explained will necessarily produce a good value after a specific kind of registration is performed, as the verification procedure used is very closely linked to the registration method. In fact, many times the "score" or "energy" of the registration is used directly as the level of certainty that the registration was correct. A few methods that have been described are detailed here.

Vasile and Marino [150] attempted to locate military vehicles in LiDAR scans of outdoor scenes. Their final verification procedure, a "goodness of fit" test, used a weighted spin image correlation coefficient. Chevalier et al. [34] located ground targets in large, outdoor scenes, first removing many scene points using a priori information (e.g., that the scene contains a large ground plane and many tall, thin trees). Their

verification procedure is the same as in [87], where the raw score from a point matching algorithm (ICP [164]) is used to represent the quality of the detection. Patterson et al. [123] also used a verification procedure based on ICP scores, and additionally required hand-labeling parts of the input to provide exemplars of the objects of interest. Smith et al. [142] proposed a verification function based on a learned linear combination of several measures of registration accuracy, including variation in the normals of corresponding points, the stability of the covariance matrix of the estimated transformation, and a novel boundary alignment check. While excellent results were obtained, this training procedure limits the algorithm to use in cases where significant training data is available, which is often not the case.

In each of these cases, the verification procedure is tightly linked to the object detection procedure, inherently biasing it towards believing an object that was detected using its metric is a good one. In contrast, our verification procedure described in Chapter 4 could be used to provide an analyst with a method-independent, easily-interpretable physical check of the final detected object position.

A technique which partially motivated our approach was introduced by Huber [83, 84] who used a method based on *visibility consistency* to determine the quality of alignment between two surfaces derived from range scans . A free space violation occurs if, after alignment, points in one of the scans occur in the free space of another scan's perspective. A diagram of this situation is shown in Figure 2.24.

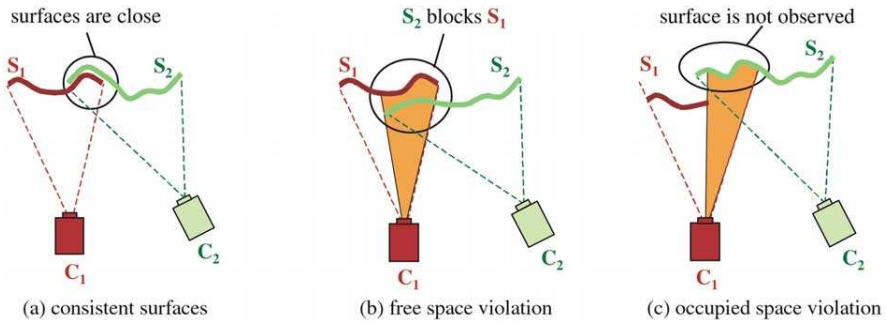


Figure 2.24: Diagram showing the concept of visibility consistency. (image from [83])

This technique requires preprocessing to extract surfaces from the range images, and hand-labeled training data to estimate the probability distributions of the dis-

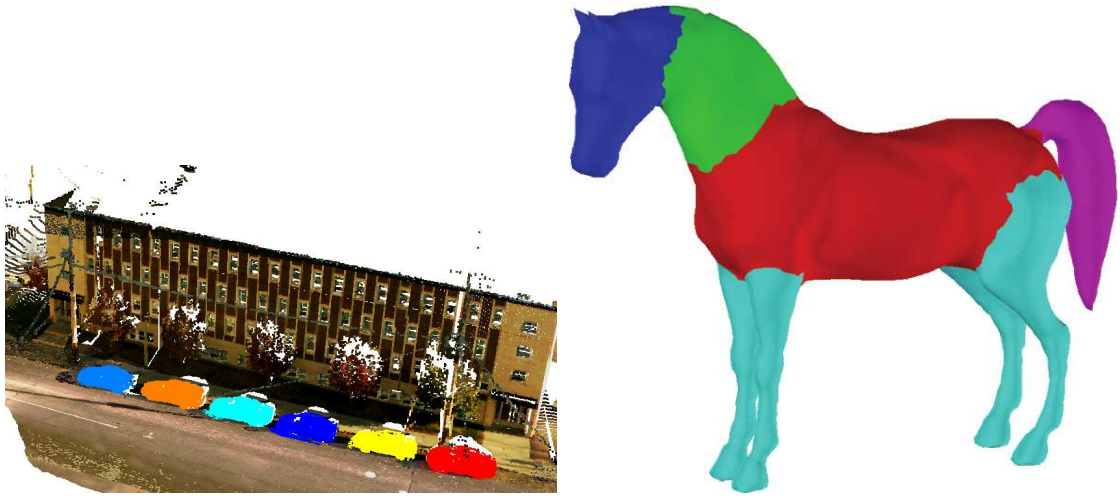
tances between two surfaces along each ray in the case of correct and incorrect alignments. Our method maintains the idea of the free-space reasoning, but discards these undesirable properties.

Mian et al. [117] introduced the related concept of "active sensor space violation" as a means of determining the accuracy of a model-to-scene registration . This technique requires the scene and the model to have approximately equal sampling densities and is based on the number of model points that have a scene point within a specified distance threshold. They also used the difference between the volume occupied by the registered sets of points and the volume occupied by the model itself to determine a "bounding dimension" constraint that provides a coarse idea of whether the point sets are approximately correctly aligned.

2.4 Object Segmentation

There are two main types of object segmentation, each conceptually very different. The first type is *object/scene segmentation*, also known as *foreground/background segmentation* or *whole object* segmentation. In this problem, we are given the known approximate location of an object, and wish to separate it from the background of the scene. The task can also be thought of as a labeling problem. That is, we wish to assign a label "object" or "not object" to each *member* of a data set. These "members" are the basic unit of representation in the data set, for example, pixels in image segmentation, points in point cloud segmentation, or polygons in mesh segmentation. The goal is to label all members that belong to the object as "object", while labeling all of the non-object points as "not object." For example, if the "object" in question is a person, we we would want to know which members in the scene belong to the person. The set of members in this problem is very heterogeneous. That is, the intra-object members will be wildly different from each other, even though these members all belong to the same object. For example, the hair, skin, and clothing of a person all have very different colors, textures, and surface orientations. This type of segmentation is very useful in practice. One important use case is constructing model databases by quickly extracting objects of interest. An example of this type of segmentation is shown in Figure 2.25a.

The second type of object segmentation is *object part segmentation*, or *parts-based segmentation*. Here we are interested in extracting coherent collections of members which belong to a "part" of an object. For example, we wish to divide a human into "arms", "legs", "head", "torso", etc. It can be argued that this problem is much easier, since the intra-part description variation is much lower than the intra-object variation. In this problem, we are often interested in taking an object, and decomposing it into its constituent parts. This type of segmentation is useful for building hierarchical descriptions of objects. For example, 3D animators wish to be able to replace, modify, and independently move different parts of a 3D character. They can use some techniques described in this section to obtain initial segmentations that they can then refine to a very accurate result. An example of this type of segmentation is shown in Figure 2.25b.



(a) (b)

Figure 2.25: Whole-object vs parts-based 3D segmentation. (a) Whole object segmentation. Each object is shown in a different color. (image from [123] (b) Object part segmentation. Each part of each object is shown in a different color. (image from [89])

2.4.1 Image Segmentation

The problem of image segmentation is the problem of dividing an image into parts, or regions, with different properties. This is exactly the problem we generically described in the previous section, where now the scene is an image and the members are its pixels. We wish to construct groups of pixels, where pixels belonging to the same

group are spatially close as well as similar in value. An example of a whole-object image segmentation is shown in Figure 2.26.

(a) (b)

Figure 2.26: An example whole-object image segmentation. (a) An image of a soldier. (b) The soldier has been segmented from the background. The foreground pixels are shown, and the background pixels are omitted (white). (images from [130])

In some cases, we are interested in coarsely separating an image into large regions, for example “ground” and “sky.” This type of segmentation is useful for categorizing images (urban, landscape, etc.), and for other “higher level” vision tasks.

The ultimate goal of both of these types of segmentation is generally scene understanding. We wish to acquire input from sensors and then automatically, or least semi-automatically, infer high level information about what is present and/or happening in the scene.

A good survey of image segmentation techniques is provided in a publication by Pal and Pal [121]. These methods can roughly be broken up into “clustering methods”, “contour methods”, and recently “graph-cut methods.”

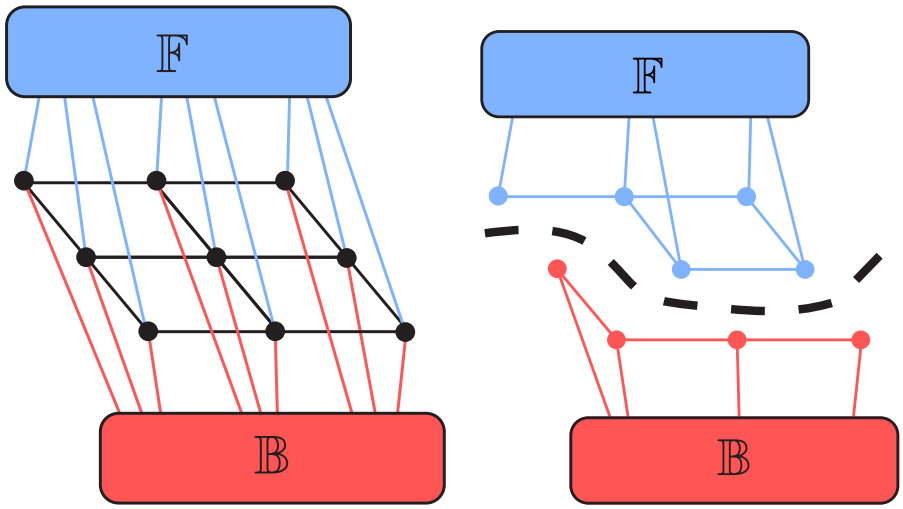
Clustering-based segmentation methods, also known as “region” methods, use statistics and spatial proximity of the pixels in an image to separate them into distinct groups. Statistical clustering methods [99] can be applied directly as a naive attempt to solve this problem. Kurita recognize that pixels are not natural entities, but rather a side effect of the discretization and storage of the scene information [76]. They show that by first locally grouping pixels into “superpixels”, it is easier and faster to obtain

semantic information about the scene. There have been several enhancements, particularly to the efficiency of this idea [2, 103]. In a similar approach presented in a paper by Xu et al., the gradient of the image is smoothed, making the clustering more robust to naturally occurring textures [157]. Felzenszwalb and Huttenlocher [67] shows that this type of over-segmentation can be performed very efficiently using graph-theoretic techniques. However, as we show in Chapter 6, these naturally occurring textures often prevent algorithms which sound nice in theory from performing as we would expect.

In contour-based segmentation methods, also known as "boundary" methods, the goal is the same but the reverse approach is taken. Rather than directly attempt to group pixels together, the problem is formulated so that we attempt to divide groups of pixels by studying their boundary. These methods operate under the assumption that pixel intensities should change abruptly between different regions. In contour-based segmentation, we change the shape of, or evolve, an initial boundary so it reaches an optimal state with respect to an energy function in the hope that it nicely divides the pixels into distinct sets with similar properties. These methods typically evolve the boundary following a forcing function which is the solution to a partial differential equation. The seminal works in this area [35, 90] have been followed by significant study and improvement, including recent hybrid techniques [158].

A class of very powerful methods that have recently become very popular in image processing is based on graph techniques. An image has an obvious representation as a graph - each pixel is a node in the graph, and adjacent pixel nodes are connected by edges. In Figure 2.27, we show the typical formulation of the graph constructed from an image to which we can apply graph-theoretic techniques to obtain a segmentation.

Graphs on images are a special case of a general graph (i.e. a structure with nodes and edges), and this high connectivity is often exploited explicitly while solving graph problems [95]. Boykov and Jolly [25] popularized the idea of performing segmentation on an image interpreted as a graph, as they demonstrated that the problem could be solved extremely efficiently. This spurred a decade of follow up work [24, 23], as it proved to be an extremely efficient way of solving many different image processing problems. In these techniques, the user is required to *scribble* on the foreground and background of the image, constraining the graph-cut problem to produce a useful ob-



(a) (b)

Figure 2.27: The graph constructed from an image. Each pixel is connected to its neighbors as well as to a foreground and background node. A cut divides the foreground and background nodes by disconnecting edges. (Images adapted from [127])

ject segmentation. This user interaction is very non-invasive and produces excellent results with very minimal effort on the part of the user. In a paper by Rother et al., instead of asking the user to scribble on the foreground and background, the user is asked to position a rectangle around the object of interest [130]. This serves a similar purpose of proving some information of where the object to be segmented is in the image, and a very coarse idea of which colors are contained in the foreground and background. A similar graph-cut based segmentation technique, the *normalized cut*, was introduced in [140]. This technique optimizes both the total dissimilarity between the segments as well as the total similarity within the segments. A follow up work by Zeng et al. [163] proposed a topological constant on the segmentation which is a very desirable property in real world segmentation problems. Boykov et al. [22] demonstrated that the binary graph-cuts based segmentation can be extended to a multi-label problem. Their technique of "α expansion" iteratively solves binary graph cut problems, allowing multiple objects to be segmented simultaneously.

Though graph-cut based segmentation methods are typically very fast, work has been done to make them even faster. Li et al. [106] first performed an over-segmentation using superpixels and then perform the graph cut segmentation on superpixels of the

image. The assumption is that all pixels in a superpixel belong the same terminal (object or background), making the problem much much smaller and therefore faster to solve.

In this thesis we directly utilize graph-cut based image segmentation, so we present a full discussion of the methods in Section 5.1.

2.4.2 LiDAR Segmentation

In this thesis we are interested in segmenting objects LiDAR data sets. Due to the huge size of these data sets, segmentation is a particularly important step in any processing pipeline. By first segmenting objects from a scene, the size of the problem for other tasks such as object detection is greatly reduced. While there has been large amounts of work on the image segmentation problem, much less work has been done to segment 3D objects from LiDAR point clouds.

Several methods have been proposed to perform parts-based segmentation in depth images and colored point clouds. Dal Mutto et al. [39] attempted to automatically segment an entire image into regions consistent in color and depth. The most common technique for depth image object-part segmentation [126, 10, 7, 9] is to identify planar surfaces and label each surface as a part. Planes occur very frequently in urban environments. For example, man made objects such as streets, buildings, etc. tend to be composed of many planar pieces. Because of this, it is a reasonable assumption that a scene can be approximated as piecewise-planar. Trucco and Fisher [149] segmented points by directly detecting and extracting planar surfaces in point clouds. Yang [160] used the information theoretic principle of the Minimum Description Length (MDL) along with the Random Sample Consensus (RANSAC) idea to find the planes that best describe the data in a very principled way. Yu et al. [162] used an iterative plane fitting technique to segment small planar surfaces in an attempt to compress the representation of a data set into a collection of polygons. An example of their result is shown in Figure 2.28.

Biosca and Lerma [20] drew from recent work in fuzzy clustering to segment point clouds into planar segments. While these techniques are very useful in industrial applications such as quality control, as well as several tasks in robotics, in this thesis

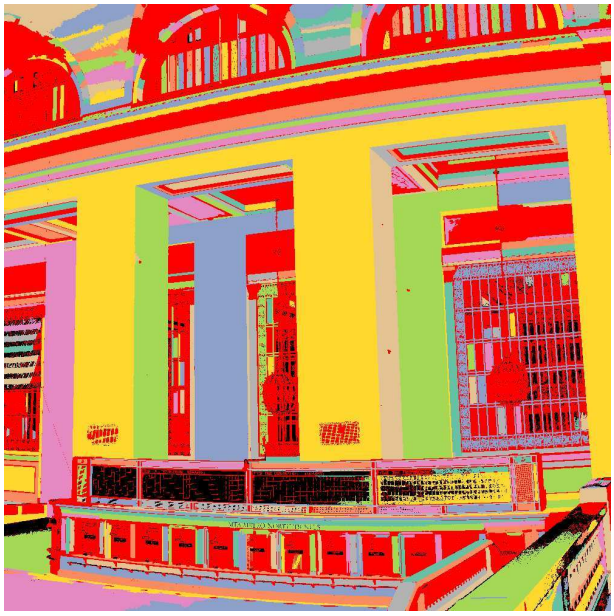


Figure 2.28: The segmentation method proposed by [162]. (Image from [162])

we are interested in whole-object segmentation, so these techniques do not apply.

Other techniques, while still interested in parts-based segmentations, remove the restriction of segmenting the data into planar objets. Liu and Zhang [111] used spectral clustering to extract components of a mesh. They constructed an affinity matrix between all of the points and then compute eigenvectors of this matrix which can be interpreted as clusters. Kalogerakis et al. [89] separated meshes into meaningful parts by creating a conditional random field (CRF) on the mesh, with a unary term that indicates how likely a polygon is to belong to a particular class, and a binary term indicating how likely two adjacent polygons are to be next to each other. The parameters of their model are learned from a training database. Klasing et al. [94] as well as Klasing and Wollherr [93] used a single pass clustering algorithm to segment point clouds into groups based only on their spatial proximity. A radially bounded nearest neighbor (RBNN) tree is computed on the points, and then use iterative cluster merging strategy is used to compute segments. This method is extremely fast, but the segmentation produced is quite coarse. Jagannathan et al. [85] used a curvature based region growing approach that is parameter free. They used techniques of graph morphology to grow a region by dilating the mesh graph and only keeping nodes attached that pass a spec-

ified curvature criterion. The curvature values of the newly added nodes are median filtered so that the subgraph corresponding to the current segmented region behaves similarly, and outliers are rejected. Anguelov et al. [8] relied on training data to *learn* classes of objects. They then attempted to determine which one of the specified set of classes a group of points belongs to. They created a Markov Random Field over the point set to enforce the constraint that points near to each other are likely to belong to the same object. Graph cut inference was then used to classify the points in a scene.

A very different approach the point cloud segmentation problem is to use principles of perceptual organization. For example, Lee and Schenk [101] used principles like parallelism, continuity, connectedness, and others to segment point clouds. Their method is hierarchical, first grouping points into patches, then patches into surfaces, and finally surfaces into objects.

The work that is most similar to ours was proposed by Golovinskiy and Funkhouser [78]. The authors explored creating a graph directly on a point cloud, and using a min-cut algorithm to separate objects from the background. They exploit the fact that objects touch the ground plane in only a very small region, so the min-cut on the graph should separate the object from the ground. Foreground points are selected by the user, and an approximate object radius must be provided. The weights on the graph edges decrease with the distance from the foreground points. The relationship of this method to graph-cuts based image segmentation techniques will become clear in our discussion in Chapter 5. A example of the process and result of this technique is shown in Figure 2.29.

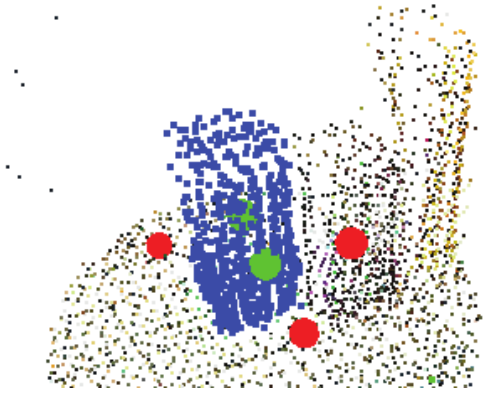


Figure 2.29: The graph-cut method proposed by [78] to segment point clouds. Foreground (green) and background (red) constraints placed by the user are shown, with the resulting segmentation the short post object in this scene shown in blue. (Image from [78])

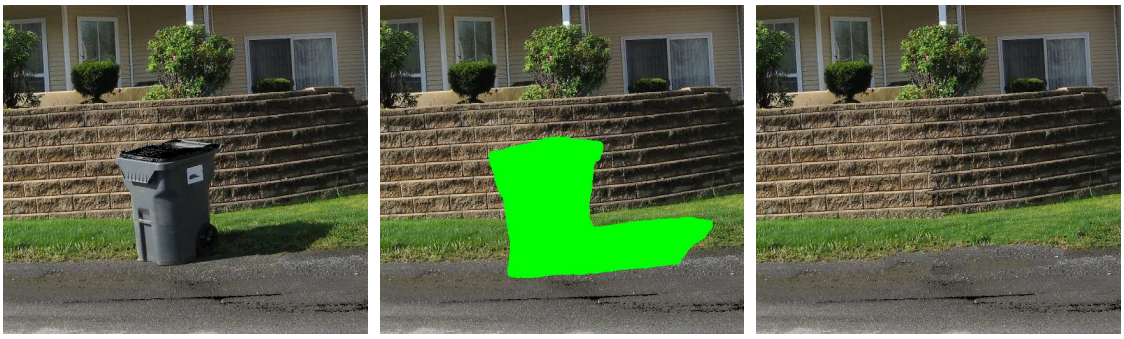
2.5 Inpainting

We next discuss inpainting, the problem of filling in a “hole" of missing data in a data set. We start by reviewing prior work on image inpainting, and then go on to discuss work related to the problem of inpainting in 3D data sets including meshes and point clouds.

2.5.1 Image Inpainting

There are three main reasons for the existence of a hole in an image. First, originally valid data may have been corrupted. For example, a photograph may have been folded or torn leading to corruption of the data it originally contained. Second, an occlusion in the scene itself may have caused the missing data. For example, a person could have been standing in front of a building so that the part of the building behind them was never seen at all. Last, the hole could have been introduced by a user performing operations on the image, such as moving or deleting objects. The name *inpainting* comes from the name given to the process by which an artist manually corrects blemishes in a physical painting.

As a motivating example of the power of inpainting, we show an inpainted image in Figure 2.30.



(a) (b) (c)

Figure 2.30: A motivational example of inpainting. (a) An image of a trashcan with a wall in the background.(b) The region to inpaint is indicated in bright green.(c) The resulting inpainted image. If presented with this image alone, it would be very difficult to notice that the image has been modified.

As we can see, this technique is extremely powerful and can produce very impressive results.

The inpainting problem is typically approached in one of two ways. The first way is by solving a differential equation to produce the "smoothest" possible region inside a specified hole. The second approach attempts to copy patches from elsewhere in the image into the unknown region in a plausible arrangement. We discuss both of these methods in the following sections.

2.5.1.1 Differential Equation Based Image Inpainting

One class of inpainting techniques solves differential equations over a hole in an image, attempting to propagate information smoothly from the boundary into the hole. These techniques have roots in equations from physics such as the heat equation. The heat equation describes the behavior of heat propagating in a medium. The intuition is that if heat is applied to a system, it will be dispersed over the medium in a smooth and well-defined fashion. In an analogous way, the colors in an image can be dispersed over a missing region. Bertalmio et al. [15, 16] noted that this smoothness should be in the direction of linear structures in an image. The direction of these linear structures can be found by rotating the gradient of an image by 90 degrees, obtaining the directions of *least* change. These vectors are referred to as the *isophote* directions

of the image. We denote the vector field of isophote directions over an image $I(x,y)$ as $\nabla^{\perp}(x,y)$. The condition that makes the pixel values in the hole continue the isophotes as smoothly as possible is shown in Equation 2.5.

$$\nabla(\nabla^2 I(x,y)) \cdot \nabla^{\perp} I(x,y) = 0 \tag{2.5}$$

If we think of the Laplacian, $\nabla^2 I(x,y)$, as representing the edges of the image, this equation shows that we want the change in these edges to be zero in the direction of the isophotes.

Researchers have used this technique [15, 32, 31] to attempt to fill small holes in an image. Oliveira et al. [120] presented a simpler and much faster approach with similar results. The authors repeatedly convolved a diffusion kernel with the image, building up color in the hole during each pass. Similarly, Telea [148] "pushed" the image values directly into the hole with a simple summation operation. Using these techniques, reasonable results can be obtained on small, thin holes as shown in Figure 2.31.

(a) (b)

Figure 2.31: An example of differential equation-based inpainting of a thin hole. (a) An image with a thin region to inpaint indicated in bright green.(b) The resulting inpainted image. If we look very closely, there are some artifacts, but the result is acceptable.

However, differential equation based methods are typically not suitable for filling

large unknown regions because they cause heavy blurring artifacts, as shown in Figure 2.32.

(a) (b)

Figure 2.32: An example of differential equation-based inpainting of a large hole. (a) An image with a large hole to inpaint indicated in bright green.(b) The resulting inpainted image. The inpainted region is much too blurry to be of any use. This result was produced using our implementation of the technique in [124].

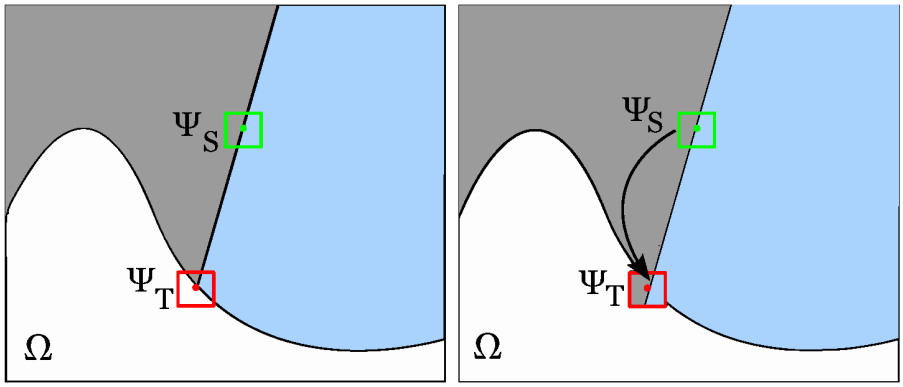
Pérez et al. [124] showed that though differential equation-based methods alone are not adequate to fill large holes, modifying these techniques by introducing a *guidance field* can produce very interesting effects. Further discussion of these techniques is provided in Section 6.4, as we use the idea of a guidance field in our LiDAR inpainting algorithm.

In this section we have shown that differential equation-based inpainting techniques are not acceptable for inpainting large holes. As large holes are the focus of our contribution in this thesis (Chapter 6), we will not focus on these differential equation-based methods.

2.5.1.2 Exemplar/Patch Based Image Inpainting

A second, and recently popular, class of techniques to inpaint image holes is referred to as “exemplar-based” or “patch-based” methods. As their name indicates, these methods attempt to fill a hole in an image by copying pixels from elsewhere in

the image into the hole. These patches of pixels should have good continuation from the known region into the unknown region. Of course, since we have no hints about what actually appeared behind the hole in the image, our goal is to create an image which seems plausible. A sketch of filling one patch using another from elsewhere in the image is shown in Figure 2.33.



(a) (b)

Figure 2.33: A conceptual demonstration of patch based inpainting. (a) An image with a hole (white) to be inpainted. A source patch, ψ_S that would be good to use to fill the target patch ψ_T is indicated. (b) The target patch properly filled.

In Figure 2.34, we show intermediate outputs of a patch-based inpainting algorithm used to inpaint a large region in the image. We can see that the hole is filled gradually one patch at a time rather than all at once, as was the case in the differential equation-based methods in the previous section.

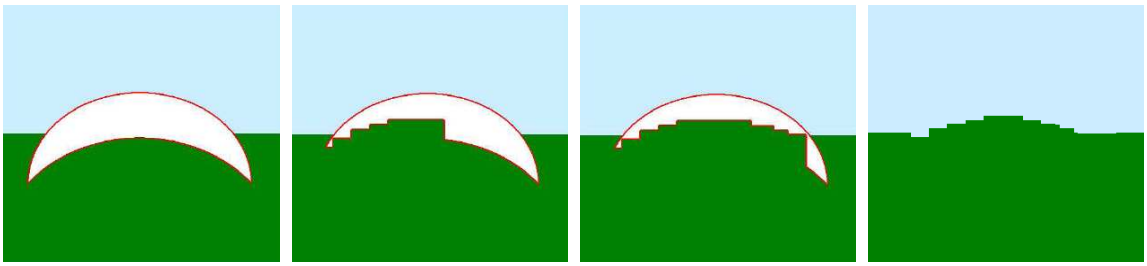
Instead of simply copying the patches as-is into the hole, Drori et al. [63] demonstrated that compositing patches using a multi-scale Laplacian pyramid approach helps ensure that no artifacts are visible at the edges of patches.

It has been repeatedly shown that the order in which these patches are copied is important. In a very popular work, Criminisi [37] suggested that patches should be copied in an order which attempts to preserve linear structures in the image. By preferentially selecting to fill patches where the gradient of the image is strong at the hole boundary, the chance of preserving these linear structures is much higher. They pointed out that the continuation of these linear structures is critical to human in-

(a) (b) (c) (d)

Figure 2.34: Intermediate output of a patch based inpainting. (a) An image to be filled. The region to be filled is indicated in bright green. (b) The resulting image after 20 iterations. (c) The resulting image after 100 iterations. (d) The final inpainted image.

terpretation of the resulting inpainted image. If the linear structures are broken, it is almost always obvious that the image has been modified. In Figure 2.35 we show what can happen if the fill order is chosen inappropriately.



(a) (b) (c) (d)

Figure 2.35: The effect of bad fill order on an inpainting result. (a) An image to be inpainted with a the hole indicated in white. The region to be inpainted is indicated in white. (b), (c) The resulting image after several iterations. In this case, the hole has been filled from the bottom up. In this extreme example, there is nothing near the bottom-center of the hole indicating that it is incorrect to copy full green patches. (d) The final inpainted image. We see that the blue/green boundary was not well preserved. (Images from [37])

We see that the linear structure (the line separating the green and blue regions of the image) was not inpainted as we would like. By starting at the bottom and filling upwards, the algorithm had no reason not to copy green patches into the hole, even though with a global perspective we can see that this was not the best choice. We discuss the criteria used to select which patch to fill in detail in Chapter 6. Following

similar motivation, Xu and Sun [159] determined the filling priority of a target patch by computing a measure of the similarity to nearby target patches. Target patches prioritized in this fashion are conceptually similar to detecting linear structures explicitly, but this technique seems to be more robust to noise and to perform better on several example images.

There have been a very large number of small improvements on the general concept of patch-based inpainting. Goyal and Diwakar [79] limited the region to search for source patches to a window of specified size around the target patch rather than searching the entire image. Sun and Jia [146] introduced two very useful ideas. First, source patches are not allowed to be only exactly from the original image, but also rotated and mirrored version of the image. This allows for a much larger set of candidate patches, which is intended to improve the overall quality of the inpainting. A second contribution of this work is to allow the user to indicate a collection of paths that must be completed first. Not only that, but since these paths are a 1D chain of patches, the problem is small enough to be solved globally. By using dynamic programming, the lowest total energy completion along the paths is found. The idea is that the most important structures in the image will definitely be completed and as well as possible, leaving only near-uniform regions left to inpaint which should be relatively easy to successfully inpaint.

(a) (b) (c) (d)

Figure 2.36: An example of the idea proposed in [146]. (a) An image to be inpainted. In this example we wish to remove the pumpkin. (b) The hole to inpaint is shown in blue, and the user specified paths to fill first are shown in bright green. (c) The resulting image after filling only along the paths. (d) The final inpainted image, essentially solving several much smaller and easier problems. (Images from [146])

Ramsing and Ruikar [128] proposed several small modifications the basic greedy patch-based inpainting algorithm, including a regularization term to prevent the usual

patch priority values from becoming less discriminative as the filling proceeds towards the center of the hole.

While most patch based inpainting techniques are greedy, there has been an attempt at extending this idea to use a globally optimal solution. Komodakis and Tziritas create a grid of unknown patches, similar to a jigsaw puzzle [97]. The problem is posed as a massively multi-label graph labeling problem, the solution to which is exactly the inpainted image. While this formulation is certainly appealing, this type of graph problem has only recently had a reasonable computational solution [6]. The inpainting framework via a labeling problem is introduced, but the key to the tractability of the technique is priority belief propagation and dynamic label pruning. Even with this massive speed up to traditional belief propagation, the technique is extremely slow. For very low resolution images ($< 200 \times 200$), the technique takes tens of minutes. Since the complexity grows exponentially with the image size, this technique is not yet feasible for real-world images.

Simakov et al. [141] introduced the notion of "bidirectional similarity" between two images. That is, all of the information in one of the images must be present in the other image, and vice versa. This metric can be optimized to perform image completion, as it helps ensure that structure cannot be present in the output image if it was not present in the input image, leading to necessarily artifact-free output. While the authors apply a general version of this metric to solve many different image processing problems (re-targeting and automatic cropping), by optimizing their *coherence* term the metric can be applied to inpainting. The authors define this metric as the deviation of the target T from coherence with respect to the source S. Namely, it measures if there are any patches in T which have not originated from S. This is a very good definition of undesired visual artifacts, and can be written as in Equation 2.6.

$$coherence(S,T) = \frac{1}{N_T} \sum_{Q \subset T} \min_{P \subset S} D(Q,P) \tag{2.6}$$

We search for the target region T which minimizes this metric, as shown in 2.7.

$$arg\min_{T} = coherence(S,T) \tag{2.7}$$

Here, N_T is the number of patches in the unknown region, and $D(Q,P)$ is any distance metric between two patches. This is an intuitive formulation, but minimizing it directly is computationally prohibitive. As such, the authors go on to explain an iterative algorithm to minimize this objective function. While this method works well, it is very slow, taking 5 minutes for images on the order of 250x200 pixels. Barnes et al. [11] presented a randomized algorithm known as "Patch Match" for constructing the Nearest Neighbor Field (NNF) which can be used to significantly speed up any patch matching function. It is based on the premise that if the location of a good source patch for a particular target patch is known, the locations of many neighboring patches can also be assumed. More recently, this idea was taken one step further by Barnes et al. [12] by extending the possible patch set to allow for rotations of patches, while still keeping the massive performance increase. Though these results are very impressive, in this thesis we have chosen to use a traditional greedy patch-based inpainting method for our work in Chapter 6 because it is a better framework in which to to isolate, explain, and analyze our contributions, versus having them be only a small part of a very large and complicated system.

Some researchers [21, 155] have extended image inpainting techniques to inpaint sequential series of images (video) using similar methods. Wexler et al. [155] introduced a multi-scale method of inpainting, where the solution is found in very highly down-sampled versions of the images, and used to help complete the next highest level, until the original resolution solution is reached. The authors note in their problem, finding a patch that is temporally consistent is more important than finding the patch that actually minimizes the patch distance function. It is exactly this notion that motivates our work in Chapter 7.

2.5.1.3 Working in the Gradient Domain

Several researchers have noted that working in the gradient domain rather than directly on an image can produce better results. Shen et al. [139] argued that instead of inpainting pixels directly, the image gradient field should be inpainted, and the image reconstructed from this gradient field. Liefers and Tan used a similar technique to remove shadows from images while maintaining the texture beneath them [107]. Levin

et al. realized that by synthesizing image gradients and then reconstructing the image from the gradients, a much smoother completion can be obtained [102]. As we use this concept as the basis for a part of our LiDAR inpainting algorithm, we explain this type of procedure in detail in Section 6.4.

2.5.1.4 Inpainting Quality Analysis

The quality of an inpainting algorithm result is very difficult to quantify. An obvious thing to do is remove a known section of an image, inpaint the hole, and compare the resulting image to the original image. However, this is not necessarily a good judge of the quality of the completion. The inpainted region can look very different (especially at a per-pixel level) from the ground truth region, yet still look very convincing. This "convincingness" is exactly our goal. Kawai et al. [91] used the idea of "evaluation by questionnaire". The authors simply ask humans questions like "which image looks better?", etc. to validate that their algorithm produces believable results.

2.5.1.5 Texture Synthesis

Texture synthesis is a very similar problem to inpainting, but it is specified slightly differently. Rather than have an image with a missing region, in texture synthesis we start with a small patch of texture, and wish to extend that patch to make a larger image with the same texture. As an example, one could start with a small patch of strawberries and create a large patch of strawberries, as shown in Figure 2.37.

(a) (b)

Figure 2.37: An example of texture synthesis. (a) A small patch of strawberries. (b) A large patch of strawberries created using a texture synthesis algorithm. (Images from [66])

We are not directly interested in this problem in this thesis, but as some techniques are similar to patch-based inpainting, a brief review of the prior work is warranted.

Efros and Leung [65] constructed a Markov Random Field (MRF) on the source texture and synthesized one pixel at a time based on the probability of the occurrence of a pixel given its neighbors in the already known region of the resulting texture. Efros and Freeman [64] appended existing patches to the existing texture using a graph-cut technique to find the best boundary at which to join two patches. Essa and Kwatra formulated the texture synthesis problem as a global energy minimization problem so that the entire textured region is optimized simultaneously [66]. As with most global techniques, it is much slower than competing algorithms, taking around 10 minutes for even a very small (256x256) texture.

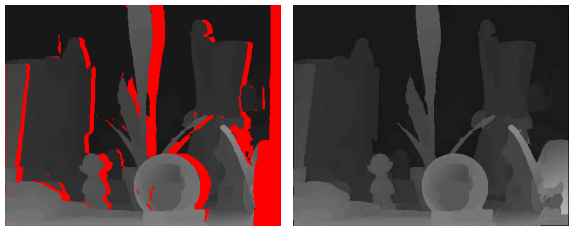
2.5.2 3D Hole Filling

In Chapter 6 of this thesis, we present a solution to the problem of filling large holes in LiDAR data. There has been previous work on filling holes in similar types of

data. The approaches that have been demonstrated fall into two very separate categories; inpainting depth images and inpainting 3D structure directly.

2.5.2.1 Depth Inpainting

Stavrou et al. [145] used 2D image repair algorithms directly on the depth image of a LiDAR scan to directly determine reasonable depths in the hole. However, their technique is only demonstrated on very simple scenes with very simple objects. Bhavsar and Rajagopalan inpaint depth directly [19]. As with any technique that performs inpainting on the depth image directly, the result is necessarily very coarse, and certainly does not lead to accurate reconstructions of the 3D surfaces. Wang et al. [154] simultaneously inpaint color and depth values, but again only in very small holes, as shown in Figure 2.38. We will show in Chapter 6.3 that this approach indeed only works for extremely small holes, or holes that have approximately uniform depth, neither of which are properties of the holes that we are interested in.



(a) (b)

Figure 2.38: An example of depth image hole filling. (a) A depth image with several small holes. (b) The holes filled using the algorithm presented in [154]. (Images from [154])

2.5.2.2 3D Inpainting

There have been several previous approaches to filling holes in 3D data sets. Sharf et al. [138] proposed a hole-filling algorithm for point sampled surfaces that used a coarse-to-fine approach. First, the rough geometry in the hole was estimated, then detailed structure was copied from elsewhere in the model to refine the initial estimate. Finally, a series of elastic warps was applied to ensure the copied patch matched the surrounding hole. While acceptable results were shown, the authors noted several

problems with this approach. These problems included the high number of degrees of freedom in aligning two point-sampled surface patches in 3D, the difficulty of defining a coordinate system in which to work, and the boundary of a hole being ill-defined in a point-sampled surface.

Park et al. [122] extended this work to point clouds with associated colors. In the final step, rather than applying elastic warping transformations, a height field is formed and a Poisson equation is solved to join the copied patch of points smoothly to the points defining the hole. Becker et al. [13] also proposed copying 3D patches of structure directly into a 3D hole. This technique relies on solving a computationally complex 3D registration problem at each iteration of the inpainting. Additionally, there is no guarantee that the inpainted 3D points correspond to a reasonable depth map from the scanner's original perspective. In this paper, we take a similar approach to copying structure from elsewhere in the scene into the hole, but since we work in the depth image gradient domain rather than directly in 3D, the complexity of the process is greatly reduced and we avoid the problems mentioned above.

Salamanca et al. [135] took a very similar approach to fill holes in meshes. An example of their results is shown in Figure 2.39.

(a) (b)

Figure 2.39: An example of mesh hole filling. (a) A mesh of a sculpture with several small holes. (b) The holes filled using the algorithm presented in [135]. (Images from [135])

Wang and Oliveira [153, 152] used a Moving Least Squares (MLS) approach to create a smooth surface through the data points on the outside of the hole boundary. While conceptually simple, this technique is only capable of generating very smooth surfaces, which are non-ideal in many situations.

Verdera et al. [151] extended the differential equation based inpainting methods mentioned in Section 2.5.1.1 to work directly on meshes. The authors view the existing points as a surface that is the zero level set of an unknown function. By solving for this function with a variational approach, the surface is automatically smoothly interpolated. Naturally, this suffers from the same problems as in differential equation based image inpainting, namely that the reconstruction over holes of any appreciable size is noticeably too smooth.

CHAPTER 3
Custom LiDAR Tools

While image processing can be considered a relatively mature field, LiDAR processing is a more recent problem. As such, there are many tools readily available for doing basic image processing tasks. For example, Matlab has tools for basic image processing operations (blurring, differencing, frequency analysis, etc.) There are also several user-level software packages for editing images, including GNU Image Manipulation Program (GIMP) and Adobe Photoshop. No such packages are available for general-purpose LiDAR data processing and manipulation, but we need these types of tools to carry out the work in this thesis. We outline here the main tools that we developed and used to support the contributions in our research. First, we developed a synthetic LiDAR scanner to produce data sets similar to those that would be acquired from a real LiDAR scanner, but based on artificial 3D models instead of real world scenes. Second, we implemented an interactive recoloring algorithm based on resectioning to transfer the color from a digital photograph onto a LiDAR scan. Finally, we mention several other tools that we developed to study and test known algorithms and to support related work. We released all of these tools under open-source licenses for other researchers to find and use, accelerating the advancement of the field.

3.1 A Synthetic LiDAR Scanner

When starting to study any problem, it is useful to start in the most controlled setting possible. For example, one should usually study an algorithm in a noise-free environment, before moving on to study its operation in the presence of noise and outliers. Unfortunately, LiDAR data is inherently noisy and full of outliers. Therefore, starting to work on the problems in this thesis directly on a real, "in the wild" data set is non-ideal. To control the situation and allow for the gradual increase in complexity that we would like, we designed a synthetic LiDAR scanner to allow us to produce data sets of scans of 3D models.

Additionally, while LiDAR scanners are becoming more common, and their cost

is steadily decreasing, they are still prohibitively expensive. A research lab would need to be very sure they wanted to head in the direction of LiDAR research before deciding to purchase a scanner, funds permitting. Even if a LiDAR scanner is available to a researcher, it can be quite time consuming to physically set up a collection of objects and scan them. This tool allows a researcher to compose a digital scene of 3D models and "scan" it by finding the intersections of many rays with the scene using techniques from ray tracing. This allows the researcher to quickly and easily produce his or her own LiDAR data. The synthetic LiDAR scan data can also be used to produce data sets for which a ground truth is known. This is useful to ensure that algorithms are behaving properly before moving to real-world LiDAR scans. If more realistic data is required, noise and false/missing detections can be added to the points to attempt to simulate a real LiDAR scan.

The synthetic LiDAR scanner developed here was submitted to the Insight Journal [41] so it could be used by other researchers. It was received well and was published in a special edition of the *Kitware Source* containing the strongest submissions of 2010 [42]. It has since been adopted by and integrated into the Point Cloud Library (PCL), a recently released software toolkit for 3D data processing.

3.1.1 Scanner Model

We based the synthetic scanner on the Leica HDS3000 LiDAR scanner. This scanner acquires points on a uniform spherical grid. The first point acquired is in the lower left corner of the grid. The scanner proceeds to acquire strips of points bottom to top, and then moves to the next strip from left to right. Each point is computed by ray casting. That is, we compute the 3D point corresponding to the intersection of the directed ray with the scene triangle closest to the scanner. This ray casting is performed using a Modified BSP Tree, a spatial data structure to facilitate ray-triangle set intersections that is much faster than a full linear search. This data structure proved significantly faster than an Octree [136] for this task.

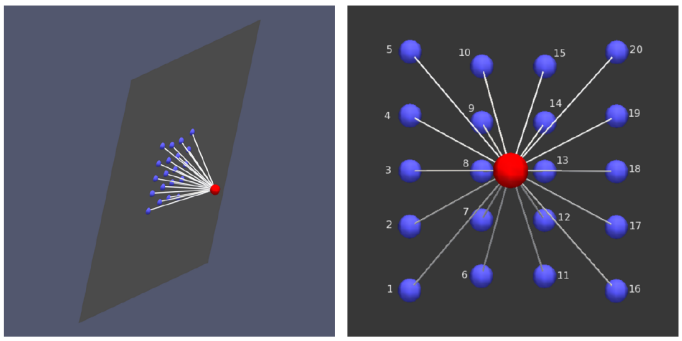
Several parameters defining the scanner orientation, angular resolution, angular bounds, and noise model can be specified. The synthetic scan output is identical to that from the real Leica scanner, in the proprietary PTX ASCII format.

3.1.2 Synthetic Scanner Parameters

It is necessary to set several parameters before performing a synthetic scan.

- Scanner position - the 3D location of the scanner in the scene
- Min/Max elevation angle (ϕ)
- Min/Max azimuth angle (θ)
- Scanner transformation - the orientation of the scanner in the scene, specified as a 4x4 transformation matrix.
- Number of strips (sampling in the θ direction).
- Number of points per strip (sampling in the ϕ direction).

To demonstrate the angles which must be specified, a ($\phi = 5, \theta = 4$) scan of a flat surface was acquired, as shown in Figure 3.1a. Throughout these examples, the red sphere indicates the scanner location, the cylinders represent the scan rays, and the blue points represent scan points (intersections of the rays with the object/scene). The points were acquired in the order shown in Figure 3.1b.



(a) 3D View of the acquisition of the scene.

(b) Order of point acquisition.

Figure 3.1: Diagram of acquisition process.

The azimuth (θ) angle of a ray is shown in Figure 3.2a, 3.2b, a top view of the scan of a flat surface. The min and max azimuth angles are labeled. Its range is $-\pi$ to π, allowing the scanner to acquire points in a full 360° range (to scan the inside

of a model of a room, for example). Since we often perform scans of scenes only "in front of" the scanner, we have oriented the coordinate system so that $\theta = -\frac{\pi}{2}$ is left and $\theta = \frac{\pi}{2}$ is right, so that scans with symmetric angles from either side of "forward" can be specified intuitively.

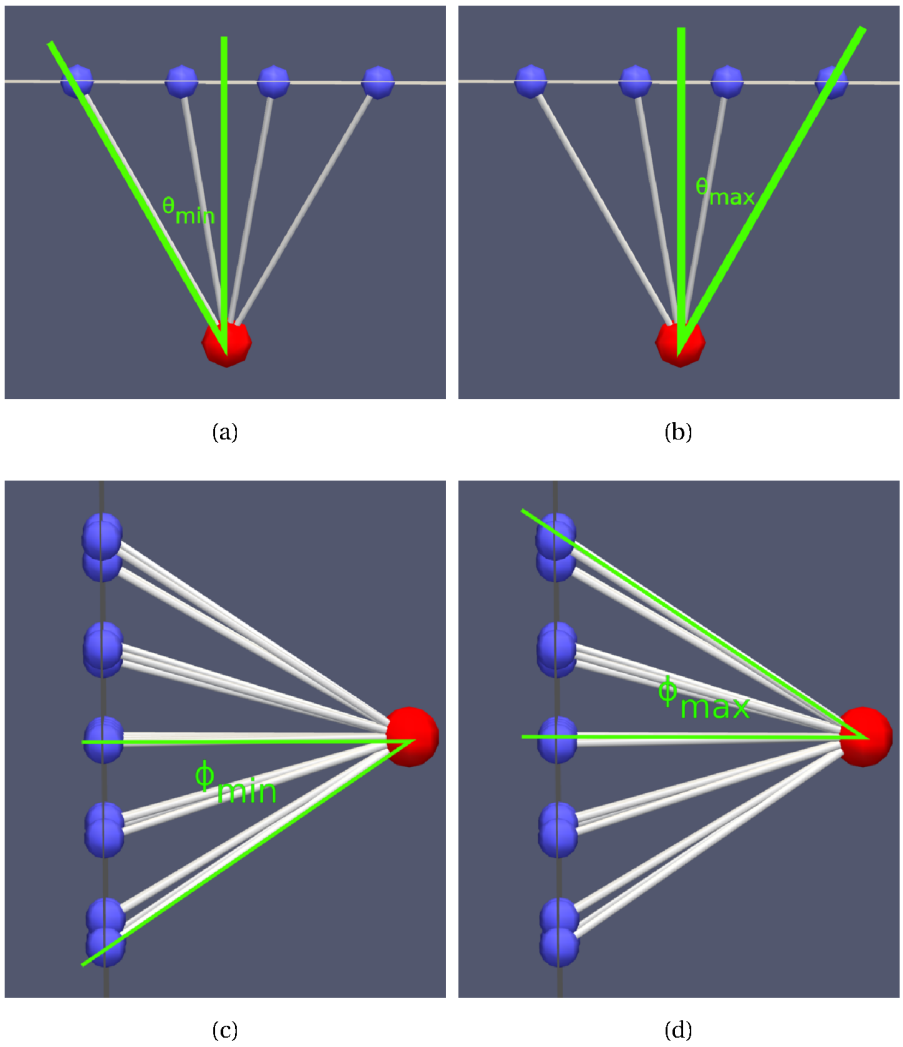


Figure 3.2: Diagram of the scanner angle settings. The scanner is represented as a red sphere, while the acquired points are shown in purple. (a)Minimum azimuth angle (viewed from above) (b) Maximum azimuth angle (viewed from above) (c) Minimum elevation angle (viewed from the side) (d) Maximum elevation angle (viewed from the side)

The elevation angle, ϕ, is shown in Figure 3.2c and 3.2d. The minimum and maximum elevation angles are labeled. The elevation angle can range from $-\frac{\pi}{2}$ (down) to $\frac{\pi}{2}$ (up).

3.1.3 Obtaining Point Normals

In a real LiDAR scan, the only information gathered is the 3D location of the points relative to the scanner. In a synthetic scanner, we get extra information, namely, the normal of the surface that the ray intersected. This is a nice addition as it allows for even more ground truth data of a scan - that is, one can compare the normals estimated by an algorithm to the actual normals of the surface that was scanned, a luxury that is never possible from a real scene. A scan of a sphere is shown in Figure 3.3 to demonstrate this.

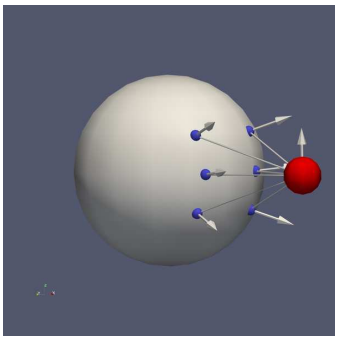


Figure 3.3: Scene intersections and their normals

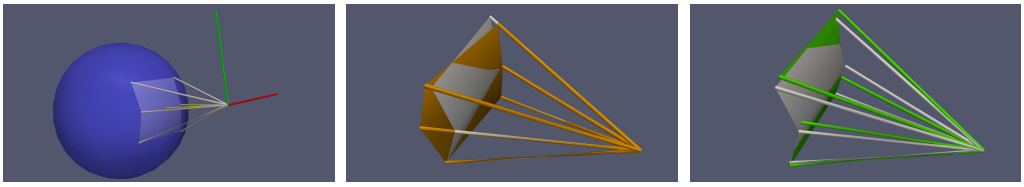
3.1.4 Noise Model

By default, a synthetic scan is "perfect" in the sense that the scan points actually lie on a surface of the 3D model as in Figure 3.4a. In a real world scan, however, this is clearly not the case. To make the synthetic scans more realistic, we modeled the noise in a LiDAR scan using two independent noise sources: line-of-sight and orthogonal.

Line-of-Sight (LOS) noise is error in the distance measurement performed by the scanner. It is a vector parallel to the scanner ray whose length is chosen randomly from a Gaussian distribution. This distribution is zero mean and has a user-specified variance. An example of a synthetic scan with LOS noise added is shown in 3.4b. The

important note is that the orange (noisy) rays are exactly aligned with the gray (noiseless) rays.

Orthogonal noise models the angular error of the scanner. It is implemented by generating a vector orthogonal to the scanner ray whose length is chosen from a Gaussian distribution. This distribution is also zero mean and has a user specified variance. An example of a synthetic scan with orthogonal noise added is shown in 3.4c. Note that the green (noisy) rays are not aligned with the gray (noiseless) rays, but they are the same length.

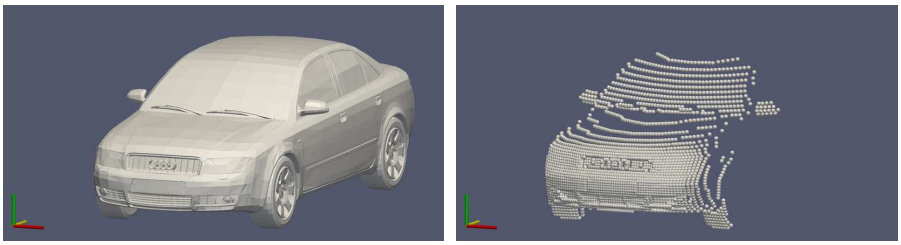


(a) (b) (c)

Figure 3.4: The noise model of our synthetic scanner. (a) A noiseless synthetic scan. The returned points are exactly where the rays intersect the model triangles. (b) A synthetic scan with line-of-sight noise added. The returned points are the intersections of the rays with the model triangles, with error added in the direction of the rays. (c) A synthetic scan with orthogonal noise added. The returned points are the intersections of the rays with the model triangles, with error added in the direction orthogonal to the rays.

3.1.5 Example Scene

As an example, a car model with ~ 20k triangles was scanned with a 100×100 grid. On a computer with a Pentium 4 3GHz processor with 2GB of RAM, the scan took 0.6 seconds. Figure 3.5 shows the model and the resulting synthetic scan.



(a) (b)

Figure 3.5: A car model and the resulting synthetic scan. (a) A 3D model of a car. (b) A synthetic scan of the car model. Each point indicates an intersect with a LiDAR ray with the model.

3.2 Recoloring a LiDAR Scan using an External Image

Some laser scanners come with an internal color camera inside the device. This camera makes it easier for surveyors to align their scans in the field, as well as recall the data that they captured during later review. Unfortunately, these cameras are often low quality compared to a consumer digital camera. Furthermore, they are not accurately registered to the scan points, and this alignment worsens over time without expensive factory recalibration. In practice, by moving the scanner around to perform scans, it inevitably is rattled and shaken enough for the camera to become misaligned. After 6 years of scanning, our scanner's camera is highly misaligned with the 3D measurements, to the point that the reported colors are nearly useless.

An additional problem with scanner's internal camera is setting the exposure. In the case of the Leica HDS3000, one must set the scanner's camera exposure manually. The image is viewed on a laptop, and the exposure can be adjusted until it looks reasonable. Unfortunately, in real scanning conditions (a sunny day, glare on the laptop screen, etc.) this is very hard to do accurately.

One solution to all of these problems is to capture an additional image of the scene with an external camera. Even a consumer level digital camera is more than enough - the scans are typically 500-1000 pixels square, so even a 1 mega-pixel camera has more than enough resolution, and the color quality is superior to the internal scanner camera.

3.2.1 Correspondence Selection

There have been some attempts to register color images to point clouds automatically [74, 115, 75, 5, 40, 1, 110], but for our purposes a manual method was simple and effective. We present here an interface to manually select corresponding points in two data sets. The data sets can each be either an image or a point cloud. If both data sets are images, the functionality is equivalent to Matlab's 'cpselect' (control point select) function. There are many uses of selecting correspondences. If both data sets are images, the correspondences can be used to compute the fundamental matrix, or to perform registration. If both data sets are point clouds, the correspondences can be used to compute a landmark transformation. If one data set is an image and the other is a point cloud, the camera matrix relating the two can be computed.

3.2.2 Graphical User Interface

The GUI is divided into a left pane and a right pane. Each pane supports either an image or a point cloud, allowing the user to select correspondences between a pair of images (for image registration seeds), a pair of point clouds (for point cloud registration landmarks), or correspondences between an image and a point cloud (for resectioning style operations). Figure 3.6 shows a screenshot of the correspondence selection interface.

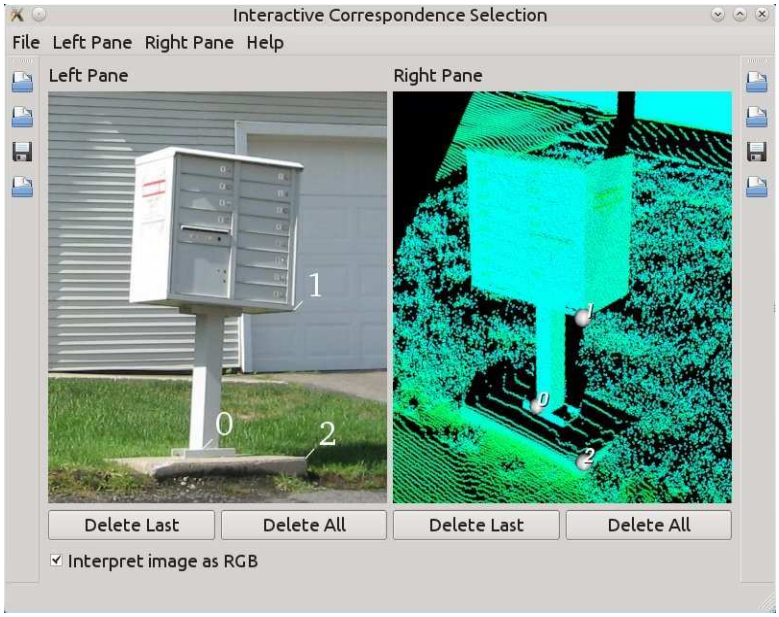


Figure 3.6: Screenshot of the correspondence selection GUI. Three corresponding points have been selected in an image and a point cloud. The point cloud is pseudo-colored by the intensity of the returns.

In the example the point cloud is pseudo-colored. In this application, we are finding correspondences between an image and a LiDAR scan, which will be used to compute the location and orientation of the camera in the scanner's coordinate system. This makes it possible to re-color the LiDAR points with the image colors. The pseudo-coloring of the points encodes their return intensity, which makes it possible to differentiate different materials, making it much easier to identify and select appropriate correspondences. If the points are all colored with the same color, it is very hard to identify such correspondences.

3.2.3 Resectioning

In this section, we discuss the procedure of *resectioning*, or estimating the camera matrix that maps a set of 3D points to their known 2D correspondences in an image. A more complete theoretical discussion is available in [80].

As we mentioned in the previous section, we have acquired an image of the scene with a camera external to the LiDAR scanner. As this camera cannot be placed exactly

at the source of the laser (the scanner housing is physically in the way), it will necessarily capture an image from a different perspective than the scan points were acquired. To map the colors from the external camera onto the scan points, we can compute the camera matrix that represents the projection of a 3D point onto a 2D image plane. We assume a perspective projection model, that is, a mapping from the scene points (X, Y, Z) to the image points (x, y) via a 3×4 projection matrix P, as shown in Equation 3.1.

$$\begin{pmatrix} \hat{x} \\ \hat{y} \\ \omega \end{pmatrix} = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.1}$$

There are several important things to mention here. First, we have used $\hat{x}$ and $\hat{y}$ because we will later normalize this vector to obtain the x and y that we are interested in. Additionally, the point that results after applying the projection is only defined up to a scale factor, which we have indicated via ω. Finally, we must append a 1 to our 3D Cartesian coordinate to create a 4-component homogeneous point.

We divide the left hand side by its last entry, and make the substitutions $x = \frac{\hat{x}}{\omega}$ and $y = \frac{\hat{y}}{\omega}$ to obtain

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.2}$$

Multiplying the right hand side and diving the result by its last entry we obtain

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{P_{11}X+P_{12}Y+P_{13}Z+P_{14}}{P_{31}X+P_{32}Y+P_{33}Z+P_{34}} \\ \frac{P_{21}X+P_{22}Y+P_{23}Z+P_{24}}{P_{31}X+P_{32}Y+P_{33}Z+P_{34}} \\ 1 \end{pmatrix} \tag{3.3}$$

We see that each correspondence generates two equations (one for x and one for y).

To solve for P, we need at least 6 correspondences (since there are 12 entries in P

and each correspondence generates 2 equations). Since these equations are linear in the entires of P, we can rearrange them to write the equation in terms of a vectorized version of P, P_{vec}. That is:

$$AP_{vec} = \begin{pmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -1 \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -yZ & -1 \end{pmatrix} \begin{pmatrix} P_{11} \\ P_{12} \\ P_{13} \\ P_{14} \\ P_{21} \\ P_{22} \\ P_{23} \\ P_{24} \\ P_{31} \\ P_{32} \\ P_{33} \\ P_{34} \end{pmatrix} = 0 \tag{3.4}$$

However, in practice, with real world data sets of the accuracy we are used to seeing with laser scanners, 8-10 correspondences are necessary to obtain a reasonable result.

The two rows of A generated by each correspondence are stacked to form an N matrix. A well known result from linear algebra is that the linear least-squares solution to an equation of this form such that P_{vec} has unit norm ($\|P\| = 1$) is the singular vector corresponding to the smallest singular value of A. After performing the singular value decomposition of A, $A = UDV^{\top}$, the last column of V is the singular vector corresponding to the smallest singular value of A. We reshape this vector into a $3x4$ matrix, forming our solution, P.

Though this is correct from a mathematical perspective, it is strongly suggested to use the normalized direct linear transform, or "normalized DLT". In this method, the exact same procedure is performed, but instead of using the image points x_i and the 3D points X_i directly, each set of points is first normalized so that it has zero mean

and the average distance of the image points to the origin $\sqrt{2}$ and the average distance of the 3D points to the origin is $\sqrt{3}$. This is carried out by constructing the normalizing similarity transformation matrices T and U and then transforming the points as $\tilde{x}_i = Tx_i$ and $\tilde{X}_i = UX_i$. The DLT procedure is then performed to compute the normalized camera matrix $\tilde{P}$, and the final camera matrix is obtained by applying the inverse similarity transformations as $P = T^{-1}\tilde{P}U$.

In Figure 3.7, we show an example of recoloring a scanner image using an image from an external camera.

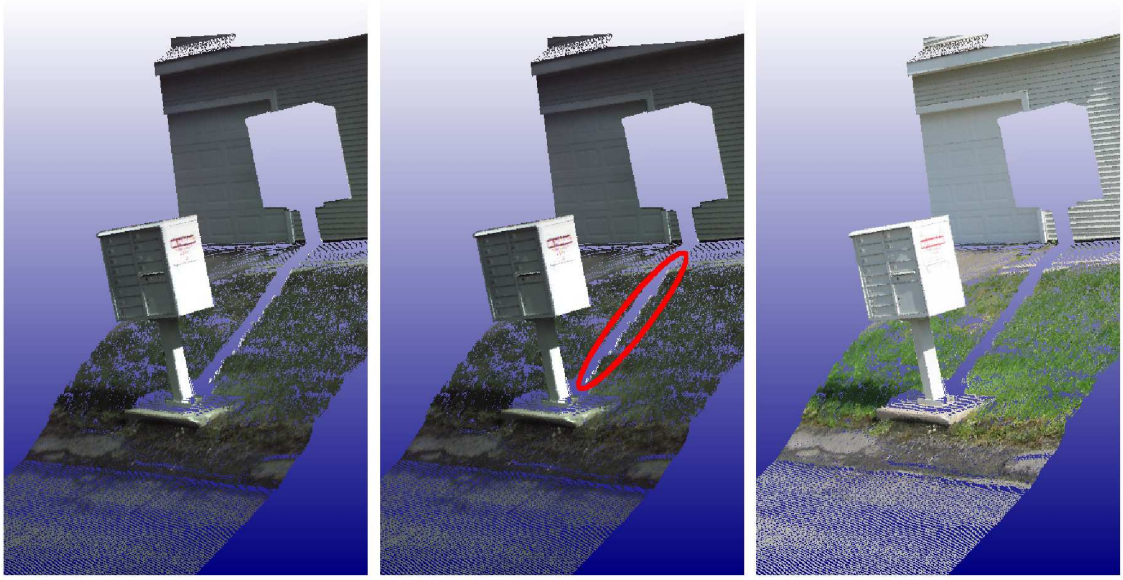
(a) (b) (c)

Figure 3.7: Recoloring the scanner image from a digital camera image. (a) The image from the scanner's internal camera. The image exposure is not ideal, and the image is of low quality. (b) The image acquired by an external digital camera. (c) The colors from the external camera mapped onto the scanner's image.

In Figure 3.8, we show the point cloud colored by the new image. We see that not only are the colors and contrast vastly improved, but the alignment is also correct. We note that there are some white pixels along the

3.2.3.1 Recoloring Scan Points

Once we have P, the projection matrix which maps 3D points to 2D image coordinates, we can project all of the scan points using this projection into the image. That is, for every 3D point X, we compute $x = PX$, the projection of the homogeneous point. After converting from homogeneous coordinates to image coordinates by dividing x by its 3rd component, we check if the resulting image coordinate actually lies inside of the image. If it does, we apply the color at this pixel to the scan point.



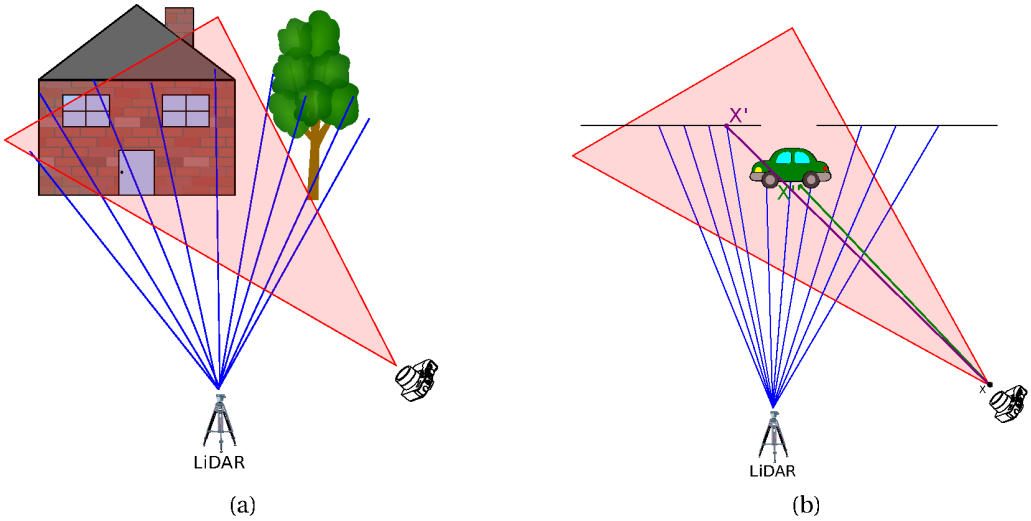
(a) (b) (c)

Figure 3.8: Corrected alignment of color images with 3D points. (a) The points colored by the original scanner image. The colors are poor quality, and not well aligned with the points. (b) A highlighted region where the misalignment is noticeable. Note the white pixels from the edge of the mailbox that appear in the grass. (c) The scan points recolored by the external image after performing our recoloring procedure.

3.2.3.2 Recoloring Issues

There are two cases which present problems to this recoloring procedure. The first is when the scanner sees parts of the scene that are not seen by the camera, as shown in Figure 3.9a. Here, the scanner acquired points on the tree, but the tree did not appear in the image. If we require these tree points in the scan to be colored, we must perform the recoloring procedure again with a different image. The second case are self-occlusions that occur in the LiDAR data from the cameras perspective, as shown in Figure 3.9b. Here the point X' naively projects to the same image point as X'', but we see that if we were to apply the recoloring procedure described in the previous section directly to this point, the wall in the background would take the color of the foreground (car) object. To compensate for this occlusion, instead of directly projecting a 3D point into the image, we intersect the line segment from the 3D point to the camera location with a mesh of the scan points. If there is no intersection, the point should be projected into the image and take on the color of the image pixel as usual. If there is an

intersection, the point has no way to obtain new information from the image, so its color should remain unchanged.



(a) (b)

Figure 3.9: Recoloring problems. (a) The view of the scene seen by the scanner and the camera. (b) Due to the different viewpoint, some image pixels have multiple 3D points that project to them.

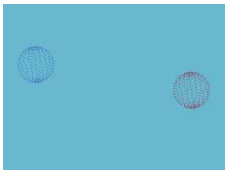
3.3 Other Tools

The synthetic LiDAR scanner and correspondence/resectioning tools were used extensively throughout this thesis, and as such were worth discussing in detail. However, we have developed many other tools which are listed here for completeness. These tools are all written as VTK filters so that they have a uniform interface and are easily accessible to researchers already familiar with the library. We have published all the work in the table below as various technical reports and magazine articles. Their citations are provided in line.

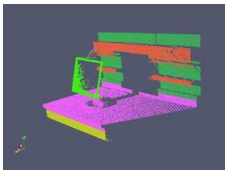
A Greedy Patch-Based Inpainting Framework: A flexible platform for testing patch based inpainting algorithms which allows easy replacement of the patch difference function and the priority function. Published in a *Kitware Source* special edition: strongest submissions of the year 2011 [43]

Poisson Editing: An implementation of the hole filling and cloning techniques described in [124]. Published in a *Kitware Source* special edition: strongest submissions of the year 2011 [44]

Point set processing: We have written several operations on point sets including outlier removal, curvature estimation, normal estimation, and normal orientation. [45]



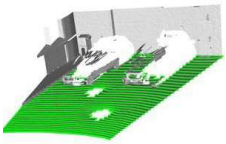
Clustering segmentation: An iterative clustering procedure based on a Radially Bounded Nearest Neighbor search. [54]



Hough Plane Detector: Find planes in 3D point cloud data. [59]

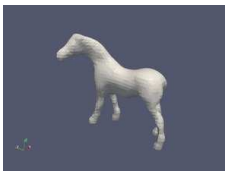
K-Means Clustering: K-Means clustering is an excellent technique for clustering points when the number of clusters is known. We have implemented the algorithm along with the K-Means++ initialization method which finds the global optimum much more frequently than a naive/random initialization. [48]

Mean Shift Clustering: Mean shift clustering is an excellent technique for clustering points when the number of clusters is not known. [49]



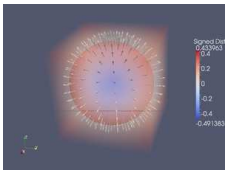
A Conditional Mesh Front Iterator: A flexible mesh region growing algorithm with easy-to-change boundary conditions. [47]

Super Pixel Image Segmentation GUI: A GUI to interactively set the parameters of and see the resulting segments from three superpixel segmentation methods including a graph-cuts based method, SLIC, and QuickShift. [53]

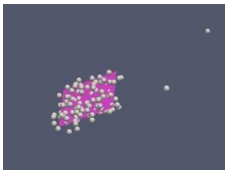


Poisson Surface Reconstruction: A VTK wrapper and Paraview plugin for Poisson surface reconstruction from point sets. [61]

Stratified Mesh Sampling: A Paraview plugin for stratified mesh sampling. Stratified sampling is a technique to uniformly resample a mesh. [52]



Point Set Surface Reconstruction: We have implemented a basic algorithm to produce a mesh from a point cloud assumed to be a point sampled surface. [50]



RANSAC Plane Fitting: RANdom SAmple Concensus (RANSAC) is an iterative method to estimate parameters of a model. It assumes that their are inliers in the data which are well explained by the chosen model. We have implemented such an algorithm to estimates the best plane in a point set. [51]

CHAPTER 4
Consistency and Confidence: A Dual Metric for Verifying 3D Object Detections in Multiple LiDAR Scans

Object detection (described in Section 2.3), or locating a given object in a large, cluttered environment is an important step in many 3D data processing algorithms.

In problems including object detection, object recognition, and 3D model construction, a common issue is aligning a 3D object model with a larger LiDAR scan. The model is typically in the form of a triangulated mesh. In this chapter, we propose a verification procedure using two complementary metrics that can be used on the outputs of any such alignments. We take as input a triangulated mesh representation of a 3D model, one or more LiDAR scans of a scene, and an estimated transformation of the model into the scene. We wish to evaluate the hypothesis that the object is present at the given position. The verification procedure is independent of the method that produced the location hypothesis, so it is objective and unbiased in deciding if the position is indeed reasonable and correct. The advantage of the dual metric is that we can answer two independent questions simultaneously. We use a measure of *consistency* to determine if the object is in a position that makes sense physically. We use a measure of *confidence* to determine, if indeed the object is at a reasonable position, how much of it we have observed. The values produced by our consistency and confidence measures are both between 0 and 1, so they are easy to interpret for any data set. Together, the metrics enable a user to make a well-informed decision about the likelihood of an object's presence or the need for more scans of the scene to answer the question more conclusively. The work in this chapter was published in [62].

A common choice of registration algorithm is Iterative Closest Points (ICP) [164]. ICP directly minimizes a cost function describing the difference between the two objects at their current positions and orientation. This value is often directly used as the final "quality of match" value. To enable a model to be matched to a partial scan, the ICP cost function is typically modified to include only points whose nearest neighbor is within some threshold [132]. This drives the cost function to a very low value for a

correct, partially overlapping match, but for an incorrect match, the value is still, by definition, fairly low. Furthermore, the ICP cost function value generally depends on the scale, sampling density, and parameterization of the problem, and is impossible to interpret as an absolute measure of match quality. Throughout this chapter, we show that our metrics are much more discerning of the actual quality of a match.

As a motivating example of the contribution in this chapter, consider the scene in Figure 4.1.

(a) (b)

Figure 4.1: A motivating example of two detected object positions with similar scores, one of which is correct, and the other incorrect. (a) A car model correctly registered to the scene. (b) A car model incorrectly registered to the scene.

We will show in Section 4.3 that a typical ICP cost function will indicate that these two positions are equally good, when we see that this is obviously not the case.

4.1 The Consistency Measure

The first measure we propose is *consistency*, which is based on the violation of free space. That is, for a LiDAR ray to have reflected off of a scene point s, there must have been no objects along the line segment from the scanner origin to s.

We place the model in the scene at a hypothesized location. For each detected point in the scene, s, if the ray from the scanner through the point intersects the model, we have a “comparable pair" with which we can reason about free space. We compute this intersection efficiently by storing the model triangles in a modified BSP tree and

using standard ray-triangle intersection techniques [113], [72]. The number of comparable pairs is denoted N_c. We know the direction of each scene point, s, from the scanner, and denote its distance from the scanner as d_s. The distance from the scanner to the model intersection, m, is denoted d_m. By considering the difference $d_m - d_s$, we can decide the consistency of the pair. If $d_m - d_s \geq 0$, the scene point is in front of the model point. This point could have been produced by either an occluding object or the object in the correct position, so we label it consistent. If $d_m - d_s < 0$, the scene point is behind the model point, which indicates that the LiDAR ray has passed through the object. This is a contradiction to the model being located at the hypothesized position, so we label the point inconsistent. To allow for noise in the acquisition process as well as slight error in the alignment, we introduce a mismatch allowance, a. We modify the conditions accordingly as given in (4.1) and Figure 4.2.

$$C_i = \begin{cases} 1 & (d_m + a) - d_s \geq 0 \\ 0 & (d_m + a) - d_s < 0 \end{cases} \tag{4.1}$$

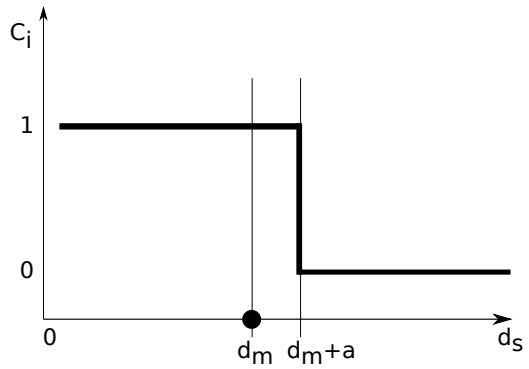


Figure 4.2: Diagram of consistency function

It is important to note the fundamental asymmetry in the consistency function. Model surfaces at equal distances in front of and behind a scene point would have very different consistency values, since the former is physically contradictory but the latter could have been produced by occlusion. Figure 4.3 illustrates the idea with three examples of comparable pairs. In ray A, the scene point is significantly behind the

model surface, so this point is inconsistent. In ray B, the scene point is only slightly behind the model surface, so this point is consistent. In ray C, the scene point is in front of the model, so this point is also consistent.

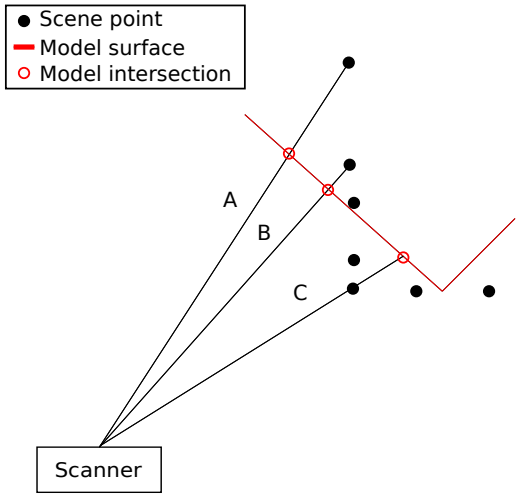


Figure 4.3: Consistency example for 3 rays.

We assign each comparable pair a binary value of 1 (consistent) or 0 (inconsistent) according to (4.1), and define the consistency of the model at the hypothesized location as the average consistency over all comparable pairs:

$$\text{Consistency} = \frac{1}{N_c} \sum_{i=1}^{N_c} C_i. \tag{4.2}$$

We note that this reduces the problem of verifying a 3D hypothesis to a combination of many 1D problems. We normalize by the number of comparable pairs to prevent the consistency value from being a function of the sampling density or the size of the object. A user can reasonably interpret this value between 0 and 1 without any other information.

If multiple registered scans of the scene are available, the consistencies of each scan should be combined into a total consistency score. It is assumed that the scene does not change between scans. Since the consistency of each scan is independent,

the total consistency after observing K scans is

$$\text{Total Consistency} = \frac{\sum_{k=1}^{K} \sum_{i=1}^{N_c^k} C_i^k}{\sum_{k=1}^{K} N_c^k}. \tag{4.3}$$

4.2 The Confidence Measure

If a model position is completely consistent, we can only declare the model *could be* at the hypothesized location, not that it *is* at that location. For example, any object model is consistent with being entirely behind a scanned wall. Our second measure, *confidence*, indicates the reliability of an estimate based on what proportion of the model has been captured by the scan(s).

The confidence measure is based on the idea that a certain amount of *information*, I_i, is associated with every model point. This information should be related to how locally distinctive the point on the model is. For example, a point on the side panel of a car should have low information, since it looks similar to any planar surface, while the uniquely-shaped front bumper should carry more information. We require that the information from all the points in the model sums to 1.

Generally, a 3D model is constructed by an artist who uses a higher density of vertices to model more complex regions; this is the case for all the models in this chapter. Thus, we can simply assign each point an equal amount of information, $I_i = \frac{1}{N_m}$, where N_m is the number of points in the model. If the model vertices are distributed uniformly (e.g., using an algorithm like [119]), the information content at each point could be related to the quality of a planar fit, with more locally complex regions containing more information. [109] proposed a more complicated method to determine the information content of a point based on point density, planarity, change in normals, and the uniformity of the change in normals.

Before any scans are acquired, we set the *observed information* O_i for each model point to 0. As scans are added, this value will increase to a potential maximum of I_i, the information content of the point. Each scanned scene point affects model points surrounding it at the hypothesis location. If a scene point is nearly coincident with a model point, it “uses up" that model point’s information- i.e., the model point has been completely “seen". We define the incremental update rule for the influence of

the jth scene point on the ith model point using a Gaussian function:

$$O_i \leftarrow \min\left(I_i, O_i + I_i e^{\frac{-d_{ij}^2}{2\sigma^2}}\right) \tag{4.4}$$

Here, d_{ij} is the distance between the two points. σ determines the radius of the sphere inside which model points are affected. One could reasonably choose σ to be a function of either the model bounding box volume m_v or the median model vertex spacing. For the experiments in this chapter, we set $\sigma = 0.01 m_v$.

Since the Gaussian function is negligibly small for $|d_{ij}| > 3\sigma$, we find all points within 3σ of the scene point using a KD-tree [14] and compute the update for only those points. Figure 4.4 illustrates an example of the information observation process, showing the influence of one scene point on three model points.

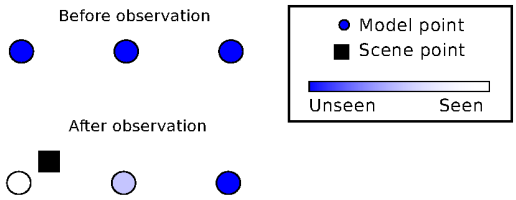


Figure 4.4: Confidence Example

The confidence that a model exists at a given location after all of the information has been collected is

$$\text{Confidence} = \sum_{i=1}^{N_m} O_i \tag{4.5}$$

where N_m is the number of model points.

We note that unlike the consistency measure, the confidence measures are not independent from scan to scan, because any overlap in scans will “see" some of the same model points. Therefore, the computation of the confidence over K multiple scans is computed as if all scene points came from a single scan. The confidence equation does not change; the only difference is that the observed information is iteratively accrued from all the points in all K scans.

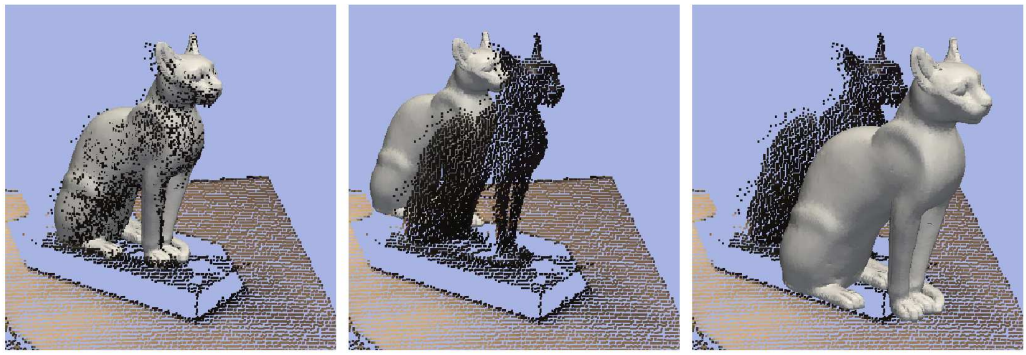
4.3 Experimental Results

In this section, we report the results of several experiments that demonstrate the properties of our dual metric. All real LiDAR scans were acquired with a Leica HDS 3000 scanner with sample spacing approximately 3 mm on the object surface.

4.3.1 Cat Sculpture — Varying Position

We obtained a high precision triangulated model of a real cat sculpture using a hand-held scanner. The dimensions of the bounding box of the model are 30x25x12cm. We then LiDAR-scanned the physical sculpture in an unoccluded scene. We used spin images followed by ICP to automatically estimate the position of the cat sculpture in the unoccluded scan (Figure 4.5a). We computed the baseline confidence and consistency values for this real-world registration. The confidence value is 0.544 because we only acquired one scan covering about half the model. The consistency value is 0.792, due to slight misalignment in the registration process as well as scanner noise. Throughout the experiments with the cat sculpture, we use a mismatch allowance of 2cm for the consistency calculations, and σ = 0.5cm for the confidence calculations.

We then placed the model behind the correct position, i.e., in the “shadow" of the LiDAR scan (Figure 4.5b). Table 4.1 shows that the consistency value in this position is very high, since almost all of the scan points do not contradict the hypothesized location. We then placed the model in front of the correct position (Figure 4.5c). The consistency is extremely low in this position, since the model is in front of the observed scene points, clearly a contradiction to the hypothesis. In both the in front and behind positions, the confidence measure is extremely low because there are almost no scene points near the model.



(a) Correct position (b) Model behind correct position (c) Model in front of correct position

Figure 4.5: Cat sculpture in varying positions.

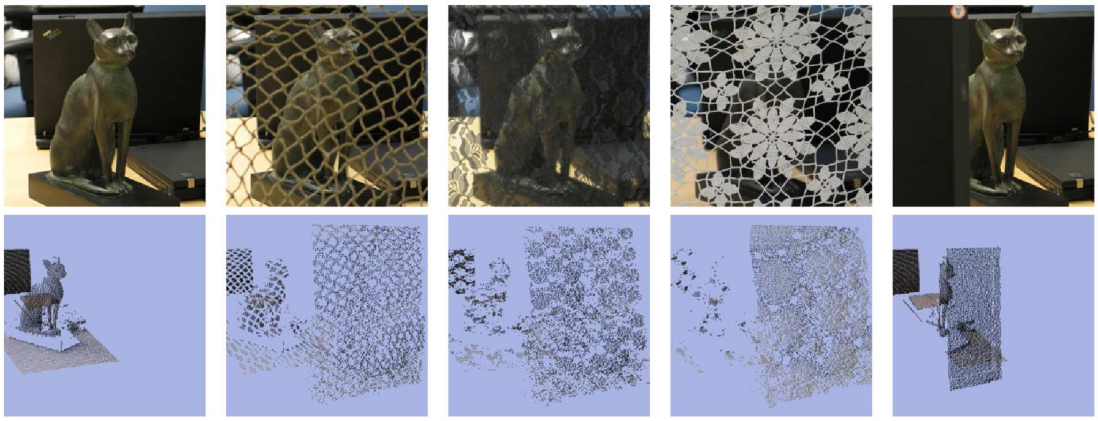
Position	Confidence	Consistency
Aligned correctly (a)	0.544	0.792
Model behind scene (b)	0.003	0.995
Model in front of scene (c)	0.000	0.151

Table 4.1: Consistency and confidence values for varying model positions in Figure 4.5.

4.3.2 Cat Sculpture — Varying Occlusion

To demonstrate the effect of occlusion on our metrics, we scanned the cat sculpture behind several different types of material. We used spin images and ICP to register the model of the cat sculpture in the scan with no occlusion, and used this position to compute the confidence and consistency metrics in five situations.

The first row of Figure 4.6 shows digital images of the cat sculpture under the varying occlusion conditions. The second row shows the LiDAR scans of the occluding object as well as the cat sculpture to illustrate the scan points that fell on the sculpture. Table 4.2 summarizes the consistency and confidence measures for the five cases. Figure 4.6a shows the scene with no occlusion and is provided as a baseline reference. The consistency is very high, and the confidence is 0.476, a typical value after observing the object from only one viewpoint. In Figure 4.6b, we scanned the scene through a net to imitate a scaled down camouflage net. The confidence of the model decreases



(a) (b) (c) (d) (e)

Figure 4.6: Cat sculpture scans with varying occlusions. (a) No occlusion, (b) Light, sporadic occlusion (net), (c) Heavy, sporadic occlusion (lace), (d) Heavy, sporadic occlusion (tablecloth), (e) Heavy, contiguous occlusion (monitor)

by about half, which agrees with our intuition that we only see about half as many points on the sculpture as we did in the unoccluded scan. However, the consistency is still very high. In Figure 4.6c, we scanned the scene through a piece of lace fabric to imitate extremely dense foliage. Again, the consistency value is still very high, but the confidence has decreased even further, as even fewer points on the sculpture have now been seen. In Figure 4.6d, we occluded the cat sculpture with a tablecloth. The results are similar to the lace fabric. Finally, in Figure 4.6e, we occluded the cat with a monitor. The consistency value is still high, but the confidence value is similar to that of the "net" case of Figure 4.6b.

Occlusion	Confidence	Consistency
None	0.476	0.879
Light, sporadic (net)	0.257	0.952
Heavy, sporadic (lace)	0.195	0.958
Heavy, sporadic (tablecloth)	0.083	0.985
Contiguous (monitor)	0.256	0.963

Table 4.2: Experimental values of consistency/confidence for different types of occlusion, cat sculpture.

4.3.3 Synthetic Cars — Multiple LiDAR Scans

In the next experiment, we demonstrate how additional LiDAR scans of a scene help improve our knowledge, as well as how the consistency and confidence metrics can be used to disambiguate similar objects. We considered a database of five synthetic automobile models, each with its center of mass at the origin. The models are all at life-size scale. We simulated sequentially LiDAR scanning each car from four different perspectives (front, driver side, rear, passenger side). The synthetic scans were created using custom software that we wrote to simulate the output from the Leica scanner that we use for real-world scans (see Section 3.1). The input is a scene consisting of triangulated meshes, a forward direction, spherical angle bounds, and spherical angle spacing. The output is a point cloud of the visible surfaces in the scene.

In Figure 4.7, the i^{th} row represents that we are hypothesizing the i^{th} model exists. The j^{th} column represents that we are comparing a hypothesis to synthetic LiDAR scans of the j^{th} model. For example, in cell $i = 2, j = 4$, we are hypothesizing the existence of the sedan2 model and comparing it to LiDAR scans of the SUV.

Each square cell in Figure 4.7 contains an independent coordinate system with confidence on the horizontal axis and consistency on the vertical axis. The k^{th} point from the left in each square represents the value of the confidence/consistency after seeing the first k scans.

Throughout this chapter, for experiments with automobiles we use a mismatch allowance of 10cm for the consistency calculations, and $\sigma = 0.3$cm for the confidence calculations.

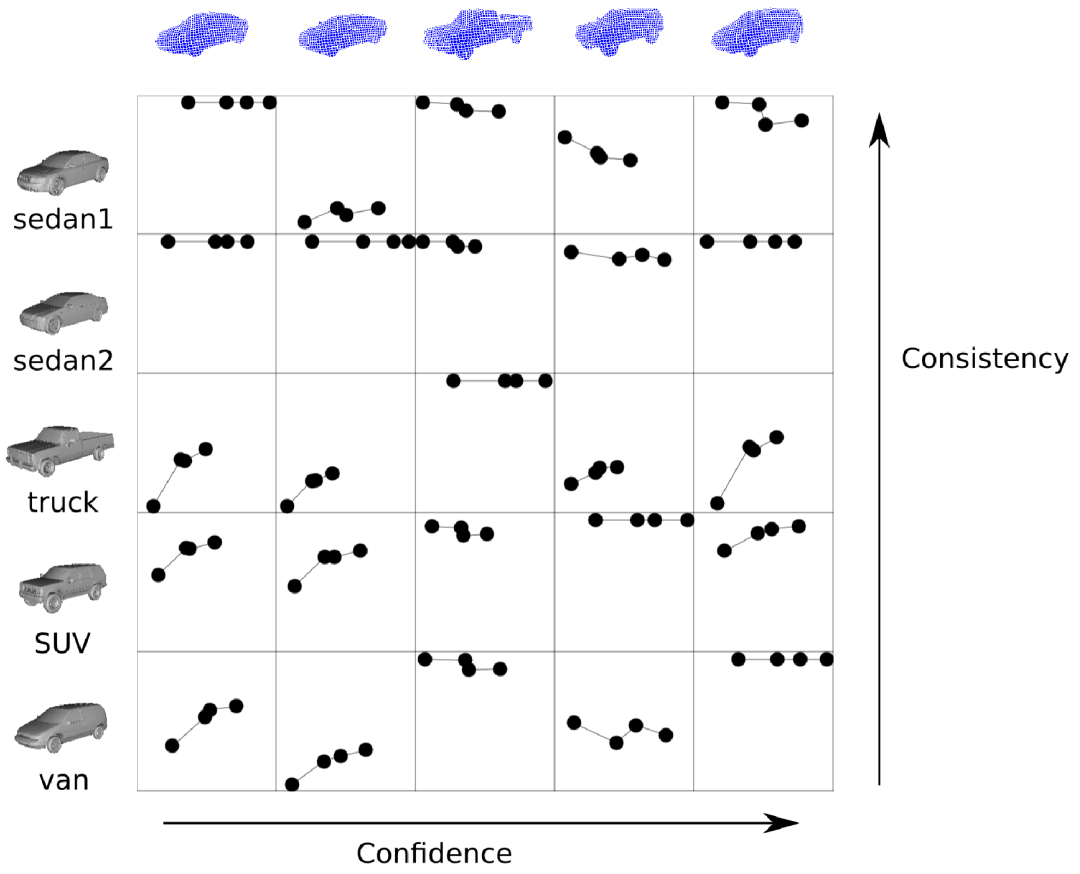


Figure 4.7: Confidence/Consistency evaluation between all combinations of five automobile models. Rows: models, columns: LiDAR scans. Each dot (left to right) represents an additional scan taken from the front, driver side, rear, and passenger side viewpoints, respectively.

Some noteworthy observations are:

- The consistency is always 1 for squares on the main diagonal. This indicates that each model's consistency with itself is 1.
- The confidence increases or remains constant with each additional scan.
- Since it is smaller, the sedan2 model is consistent with the scan of sedan1 (cell (2,1)), but the sedan1 model is not consistent with the scan of sedan2 (cell (1,2)).

- Three of the models are smaller than the van. Therefore, they are each consistent with the scans of the van (cells (1,5), (2,5) and (4,5)). However, the truck is longer than the van, so the truck model is inconsistent with the scan of the van (cell (3,5)).

- In cell (3,1), we can see that the front of the truck is inconsistent with sedan1, but the sharp increase with the second scan indicates that their sides are similar.

By inspecting this table, we see that the two measures behave as we expect, allowing a human to easily interpret the situation based on the two values.

4.3.4 Real Parking Lot Scans

Typical coarse registration algorithms produce several initializations that are refined by an ICP method. Some of these initializations produce high average point-to-point distances and can quickly be discarded. However, several positions often need to be manually discarded by the user. Such positions have a low average distance, but are physically very incorrect. Since a typical ICP cost function value depends on the scale, sampling density, and parameterization of the problem, it is very difficult to compare the quality of matches across multiple search objects and scales. Our metrics, however, are independent of object size and therefore can easily be directly compared. In this example, we demonstrate how our metrics are much easier to interpret than the ICP cost function values.

We acquired a LiDAR scan of two cars in a parking lot. Two hypothetical outputs of a coarse registration algorithm between an Audi A4 model and the scene are shown in Figures 4.8a and 4.8b (we have repeated the figure from the motivating example in the beginning of this chapter here for the readers convenience). One is correct, and the other is incorrect (it lies halfway between the two cars in the scan). Table 4.3 reports the ICP cost function value, consistency, and confidence for the two positions. We employed a standard ICP cost function, shown in Equation 4.6.

Position	ICP Cost Function	Confidence	Consistency
Correct	0.057	0.579	0.589
Incorrect	0.094	0.252	0.077

Table 4.3: Measures for Audi positions in parking lot scan.

$$\text{ICP Cost Function} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{R}\vec{X}_i + \mathbf{t} - \vec{Y}_i\| \tag{4.6}$$

Here, X_i is a scene point and Y_i is the nearest model point to X_i. Scene points for which the nearest model point is more than 0.2 meters away were not included in the ICP cost function, a common technique described in [132]. It is important to note that regardless of which variant of the ICP cost function is used, the value is always in meters, in contrast to our metrics which both take unit-less values between 0 and 1. Also, as the complexity of the selected ICP function increases (e.g., by weighting each point's contribution differently), the ability to intuitively interpret the value decreases.

Both positions have comparable average point-to-point distances (the ICP cost function value), which are both below 10 cm. For our new measures, the correct position has a high confidence value (given only one viewpoint) as expected, and the consistency is reasonable, though slightly lower than ideal. This is largely due to the transparent windshield in the real scene, which causes discrepancies in model fitting (see [116]). However, in the other position, the extremely low consistency value alone is grounds to declare this position incorrect. The confidence is non-zero because the each side of the model aligns with the adjacent side of one vehicle in the scene.

In Figures 4.8c and 4.8d, we show the observed information of the car model vertices in both positions. In the correct position, the front and driver side points are green (seen) and the rest are red (unseen). In the incorrect position (between the two cars), points on both sides of the model are seen, but the rest of the points are unseen.

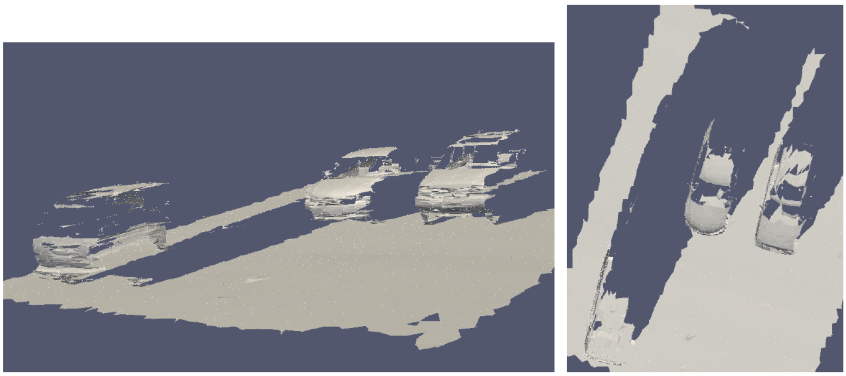
In Figures 4.8e and 4.8f, we see that in the correct position most of the points are consistent. The inconsistencies stem from the model not being a perfect match (i.e., the model is a 2000 Audi A4 and the scene is a 2009 Audi A4) as well as slight misalignment. In the incorrect position, almost all of the points are inconsistent because the scanner "saw through" the model to the back wall. This is a typical example of how

(a) (b) (c) (d) (e) (f)

Figure 4.8: Parking lot demonstration. (a) Model registered to correct position in scene. (b) Model registered to incorrect position in scene. (c) Model points at correct position colored by confidence (unseen: red, seen: green). (d) Model points at incorrect position colored by confidence (unseen: red, seen: green). (e) Scene points at correct position colored by consistency (inconsistent: red, consistent: green) (e) Scene points at incorrect position colored by consistency (inconsistent: red, consistent: green).

the consistency and confidence measures play a useful dual role for understanding if a hypothesized position makes physical sense.

Figure 4.9 illustrates a second LiDAR scan of three automobiles in a parking lot. We computed the consistency and confidence measures for an Audi A4 car model positioned at every 20 cm in the horizontal and vertical directions, assuming the model is major-axis-aligned with the parking space lines and located on the ground plane.



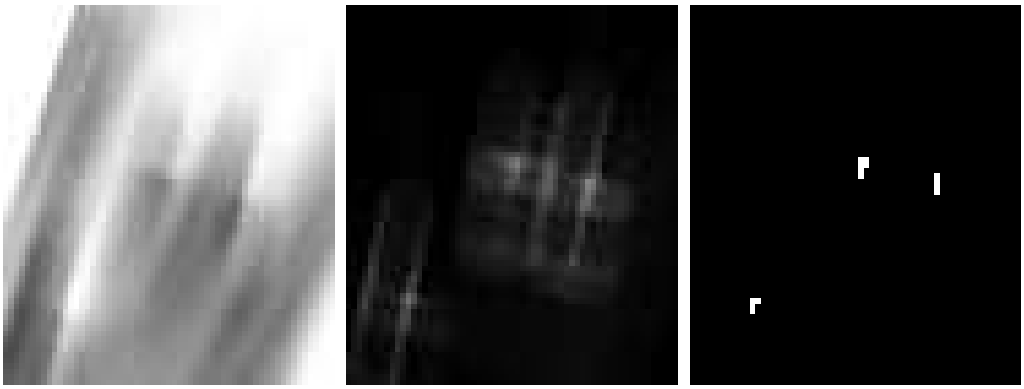
(a) 3D view (b) Top view

Figure 4.9: Parking lot scene with three cars.

Figure 4.10a shows a “heat map" of consistency values over the scene. We see that positions in the LiDAR shadow of the automobiles have high consistency. Figure 4.10b shows a heat map of the confidence values over the scene. There are several false positives. These can occur when significant parts of the model align with the scene, due to symmetries and the fact that any two near-planar objects tend to look alike. In Figure 4.10c, we thresholded the consistency map with a value of 0.75 and the confidence map with a value of 0.3 and boolean ANDed the resulting images. The position of all three automobiles are clearly verified with no false positives. However, we believe that considering both measures together leads to better-informed decisions than combining them into a single scalar value.

4.3.5 Demonstration: Sliding a Window Along a Building

In this demonstration, we will show how our two metrics behave for a model across different positions in a scene. In Figure 4.11, we show a model of a window.



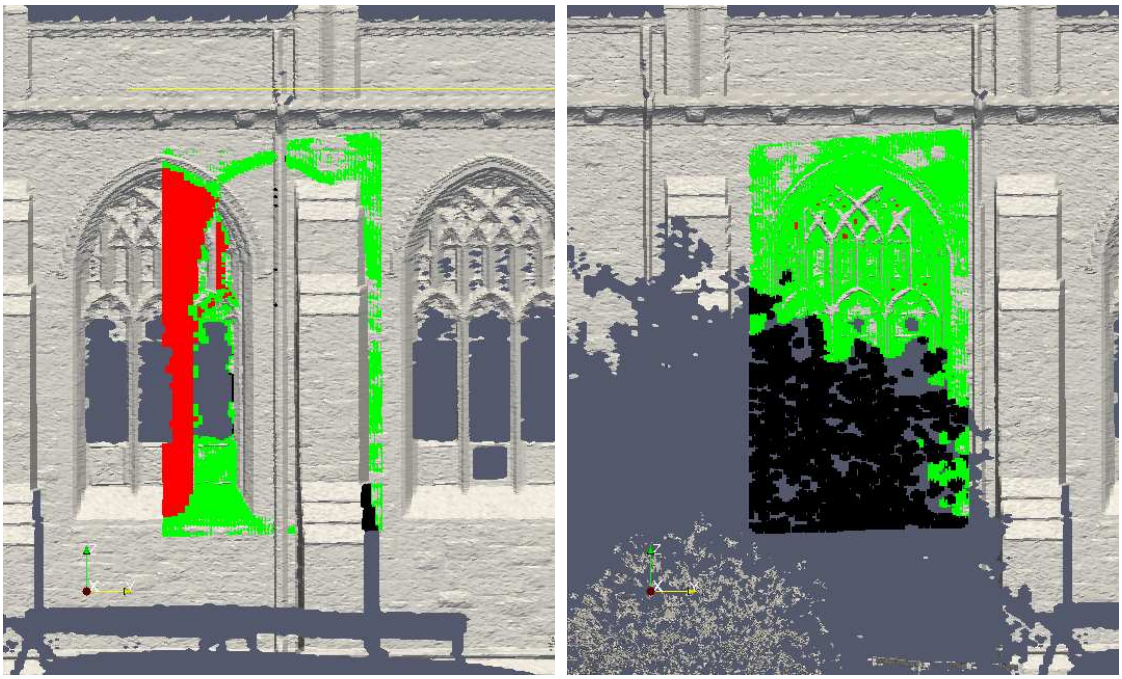
(a) (b) (c)

Figure 4.10: Consistency/confidence heat maps. (a) Consistency heat map (b) Confidence heat map (c) Dual thresholded with consistency > 0.75 and confidence > 0.3.

Figure 4.11: A model of a window.

In Figure 4.12, we show two positions of the window model in the building scan. In Figure 4.12a the window model has several points (red) that are inconsistent with the scene. In Figure 4.12b the window model has several points (black) which were not seen in the building scan, so these points do not contribute to collecting information that we are confident that a window actually exists at this position.

In Figure 4.13, we have slid this model of a window across the facade of a building



(a) (b)

Figure 4.12: Demonstration of Consistency and Confidence of points on a window. (a) At this position, the window model has several points (red) that are inconsistent with the scene. The points shown in green are consistent with the scene. (b) At this position, the window model has several points (black) which were not seen in the building scan. The green points indicate points which have a nearby point in the scan, which contribute positively to our confidence measure.

and overlayed the values of our two metrics. We can see that there are spikes in both values when the model is at a position that is aligned with a window of the building.

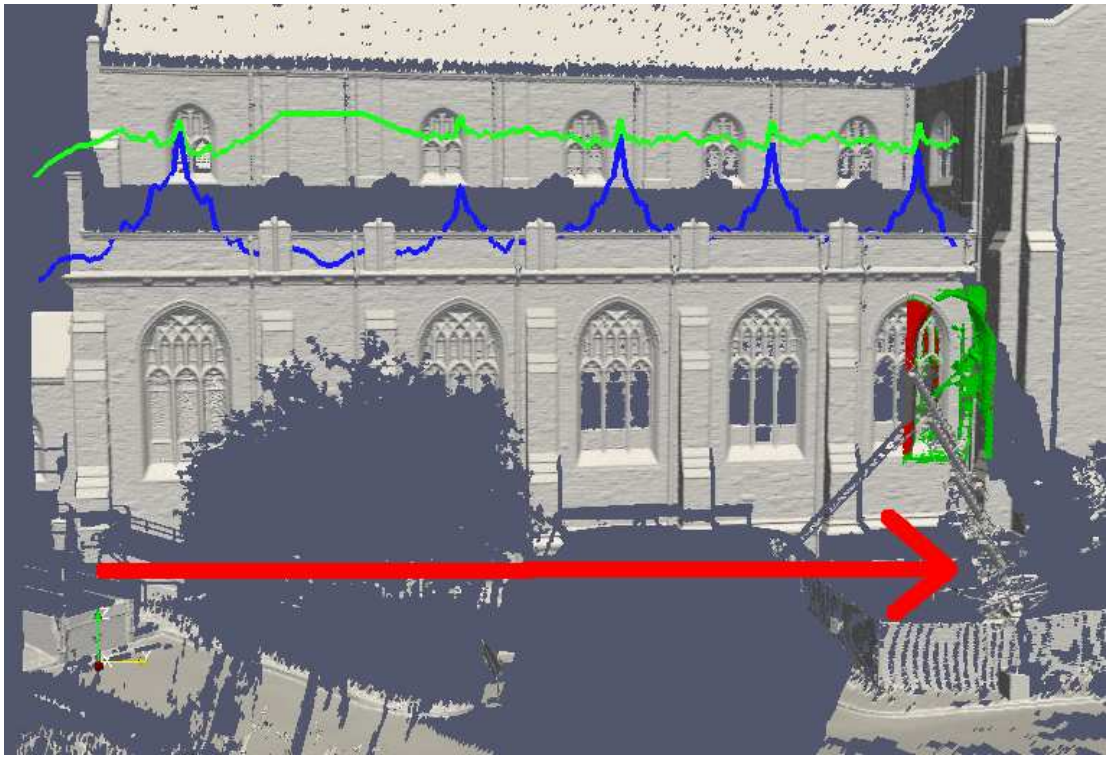


Figure 4.13: The Confidence and Consistency measures overlayed as we slide a model of a window across the face of a building. The blue line indicates the confidence value, while the green line indicates the consistency value.

In Figure 4.14, we have thresholded our metrics to classify each model point as ”good“ (high consistency and confidence), ”bad“ (inconsistent with the scene at the current position), and ”uninformative“ (there was no corresponding scene point near the model point). Again we can see that there are spikes and dips in all of these measures corresponding to the actual positions of the windows in the scene.

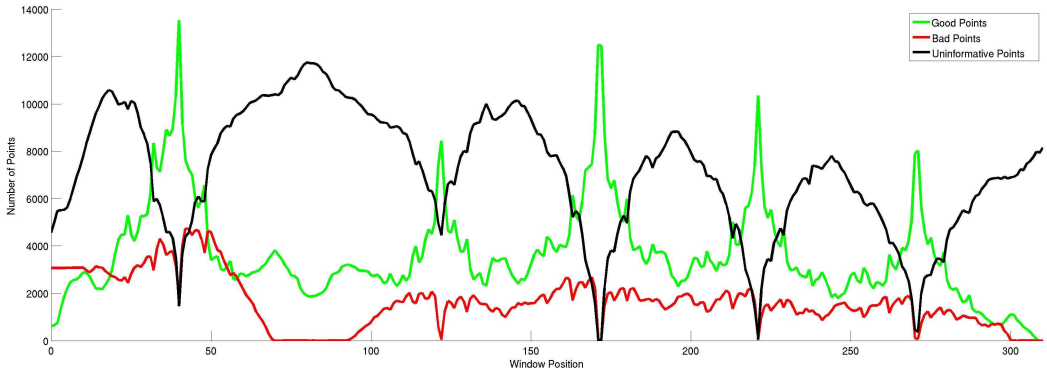


Figure 4.14: The Confidence and Consistency measures overlayed as we slide a model of a window across the face of a building. The blue line indicates the confidence value, while the green line indicates the consistency value.

4.4 Discussion

In this chapter we presented a dual metric for deciding whether a 3D object exists at a hypothesized location in a LiDAR scan. A set of such locations produced by any registration method can be verified using these measures, which together are able to provide physically meaningful values for a user to interpret. The experiments demonstrated the feasibility and accuracy of this method.

The consistency calculation currently takes an average of 0.3 seconds for each position in Figure 4.9 on a Pentium 4, 3GHz computer with 2GB of RAM. This could be improved by using a lower resolution model (our car model is ≈500,000 triangles). Typical models used in object detection are not designed with resolution variability in mind, so applying mesh decimation techniques is non-trivial, as they tend to fail to maintain the overall structure of the mesh. For this reason we chose to use the high resolution mesh throughout the experiments. We also could speed up the consistency calculation by employing a coarse-to-fine strategy. For example, we could evaluate the consistency using a uniformly downsampled set of the scene points. If the downsampled scene points are inconsistent with the model hypothesis, the probability that the entire set is also inconsistent is extremely high and further computation can be avoided. We could also use a depth buffer comparison rather than a ray-wise compar-

ison to tremendously speed up this computation. However, there are several difficulties with this approach. The scan is a point cloud, not a triangulated mesh, so a point rendering system such as [104] must be employed. The resolution of the rendering window must be chosen such that there are similar numbers of corresponding pixels as there are scene points (and thus rays).

Currently, our dual metrics return similar values for a given amount of occlusion without considering the contiguity of the occlusion. For example, in Figure 4.6, the "net" occlusion produces almost identical values to the "monitor" occlusion. A possible solution is to remove the independence assumption on the collection of 1D problems along each ray, e.g., by using a first order Markov random field. This approach would favor neighborhoods of scene points that had similar consistencies.

Our consistency calculations assume that multiple registered scans come from a perfectly static scene. Relaxing this assumption would open up new research questions. For example, we could determine that an object was present and still for one scan, and then was either moved or occluded before the next scan was acquired.

Finally, we note that an accurate 3D model is frequently not available for the objects we might want to locate in the scene. This calls for a non-model based approach, in which the consistency and confidence for a scan are determined with respect to several example pictures or scans of a model.

CHAPTER 5
LiDAR Segmentation

In this chapter, we first discuss in detail a popular method for segmenting images known as *graph-cut segmentation*. We go on to show that this method can be extended to allow a user to segment entire objects from a LiDAR scan with only a few mouse clicks. A two-step algorithm is introduced which operates on a combined color-and-depth image. It first takes advantage of sharp depth discontinuities present at some points of an object, and uses this initial result to perform a second segmentation using the available color information.

5.1 Traditional Image Segmentation

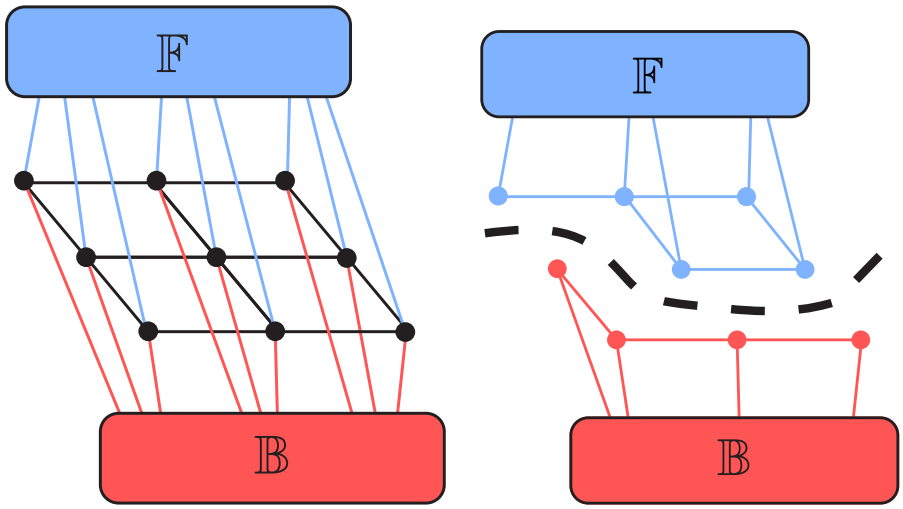
By segmenting an image, we mean that we want to classify each pixel as foreground (F) or background (B). Graph-cut techniques achieve this goal using two complementary ideas. The first is that the similarity of adjacent pixels should be high in regions that belong to one class or another. That is, a large group of connected red pixels will likely all be classified into the same group (F or B). Second, pixels belonging to the foreground or background should fit well to an a-priori description of their respective regions. This description is typically provided in the form of a probability distribution, which comes from either a domain-specific database, or is specified by the user by "scribbling" on the image, a technique which we will explain in the following sections.

We motivate the problem in Figure 5.1. Here we have segmented the soldier in Figure 5.1a from the background, and showed the resulting foreground pixels in Figure 5.1b.

(a) (b)

Figure 5.1: An example image segmentation. (a) An image of a soldier. (b) The soldier has been segmented from the background. (images from [130])

To discuss the graph-cut segmentation problem, we must first introduce some terminology. A graph G is composed of a set of vertices V and a set of edges E defining the connectivity of the vertices in V. To represent an image as a graph, we create a vertex in V for every pixel in the image. These vertices of the graph all lie on a grid in a plane, arranged in the same formation as the image pixels. Additionally, two special nodes are added to V, one which we call F for foreground, and the other B for background. These nodes are often referred to as *terminals*. The reason for these nodes will be made clear in the following discussion. In an image, the pixel connectivity is defined implicitly by the pixels' location. In a graph however, we must be more explicit. That is, we add an edge e_{ij} to E between each vertex corresponding to each pixel i and all of its neighbors $j \in N(i)$. The neighborhood connectivity $N(i)$ which is typically used is a 4-connected neighborhood, meaning that a pixel is only considered to be connected to its north, south, east and west neighbors. This is in contrast to an 8-neighborhood which also considers diagonal pixels to be connected. These edges are referred to as *n-edges*, indicating that they are connecting the standard nodes to each other. Next, we add edges E_{iF} and E_{iB} connecting each pixel i to the foreground and background terminals F and B. These edges are referred to as *t-edges*, indicating that they connect a node to one of the terminal nodes. The resulting graph shown in Figure 5.2.



(a) (b)

Figure 5.2: The graph constructed from an image. 5.2a The initial graph. The nodes corresponding to image pixels are shown as black dots. The n-edges are also shown in black. The t-edges are colored to match their respective terminals. 5.2b After the cut (black dashed line) is computed, the edges it severs are removed from the graph. We see that this leaves us with some nodes (now red) connected to only the background terminal, and other nodes (now blue) connected to only the foreground terminal. (Images adapted from [127])

Our goal in segmentation is to remove a subset of the edges of E such that there is no longer a path from F to B (F and B are disconnected). A *cut*, c, on the graph is defined as a set of edges that satisfies this condition. There is a huge set of possible cuts on this graph, but the segmentation we are looking for is a specific cut that minimizes an energy function formulated to have the properties we desire in a good segmentation. In the remainder of this section, we detail the construction of this energy function.

5.1.1 Edge Weights

Before we can formulate the energy function, we must assign a *weight*, w_{ij}, to each edge, e_{ij}. The n-weights and t-weights play very different roles in our graph-cut segmentation formulation, as shown in the following sections.

5.1.1.1 N-Weights

In this formulation, the n-weights indicate the similarity between the two vertices connected by the n-edges. That is, w_{ij} is small if vertices i and j are dissimilar and large otherwise. In graphs on images, the weights on n-edges are typically computed to be some function of the RGB values at the corresponding image pixels. The weight function often takes the form of a negative exponential of a distance function between the pixel vectors associated with the nodes, as shown in Equation 5.1.

$$w_{i,j} = e^{-d(I_i, I_j)} \tag{5.1}$$

The distance function $d(I_i, I_j)$ is often the squared norm of the difference between the RGB vectors at pixels i and j, as shown in Equation 5.2

$$d(I_i, I_j) = \|I_i - I_j\|^2 = (I_i^r - I_j^r)^2 + (I_i^g - I_j^g)^2 + (I_i^b - I_j^b)^2 \tag{5.2}$$

However, this could easily be replaced by any function of I_i and I_j, such as the difference between these pixels represented in a different color space (such as HSL [88]), or any other vector distance function such as the L1-norm, for example.

5.1.1.2 T-Weights

To compute the t-edge weights, a probability density function is constructed indicating the likelihood that a pixel belongs to the foreground or to the background. This function can be computed from two sources. First, a model of the foreground or background can be learned from a collection of training images of the objects of interest. For example, if we want to segment cars in images, we could collect many images of cars and manually label the pixels belonging to the cars (and implicitly, the pixels not belonging to the cars). We can then construct a probability density function from the entire set of pixels belonging to cars. Alternatively, if no model of the object to segment is known a-priori, one can be constructed on-the-fly by the user. Typically an interface is provided to allow a user to *scribble* on the image (shown in Figure 5.3), loosely indicating some constraints on the problem. The user can very casually mark pixels in the interior of the object as "definitely foreground", and far away from the object as

"definitely background".

Figure 5.3: An image with user drawn "scribbles" on the foreground (green) and background (red).

These selected pixels now serve as the model of the foreground and background, which are again used to produce a probability density function for both regions. A normalized histogram is often used as an estimate of the true probability density function.

For a pixel i, the weights to the foreground and background node can be written as shown in Equation 5.3. Note that these are the negative log-likelihoods of the *opposite* region.

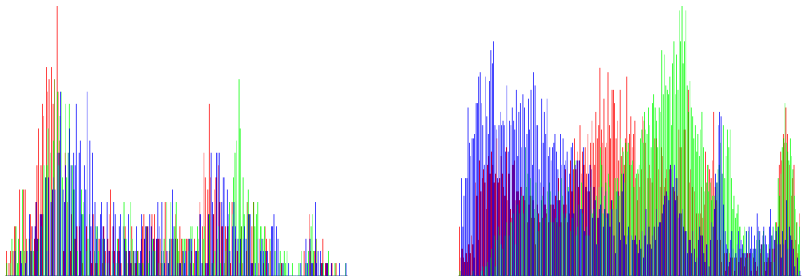
$$w_{i,F} = -\lambda \log P_B(I_i) \tag{5.3}$$

$$w_{i,B} = -\lambda \log P_F(I_i) \tag{5.4}$$

For example, if $P_B(I_i)$ is very low, then $w_{i,F}$ will be very high, making it much more likely that the edge between i and B is cut. Here, λ is a scalar that determines the relative magnitudes of the t-weights and n-weights. Throughout this thesis, we have chosen $\lambda = 0.01$.

In Figure 5.4 we show the histograms of the pixels selected by the user from Figure 5.3. In practice, we use a true 3D histogram (RGB), but it is much easier to visualize the 1D histograms of each color channel as a colored, interlaced histogram.

For pixels indicated by the user to be foreground, the corresponding weight on



(a) (b)

Figure 5.4: Histograms of the user indicated foreground and background pixels. (a) The histogram of the foreground scribbles. (b) The histogram of the background scribbles.

the t-edge connected to B is set to 0, indicating that this edge should be extremely easy to cut. Conversely, the weight on the t-edge to F is set to infinity (or a very large number) so that this edge can never be cut. User-marked background pixels are handled exactly oppositely.

5.1.2 Computing the Minimum Cut

A well known result from graph theory is that given a graph with non-negative weights, it is possible to find the *minimum cut* between two nodes, or cut which contains edges with the smallest sum. We call this sum the *energy*, J, of the cut as shown in Equation 5.5.

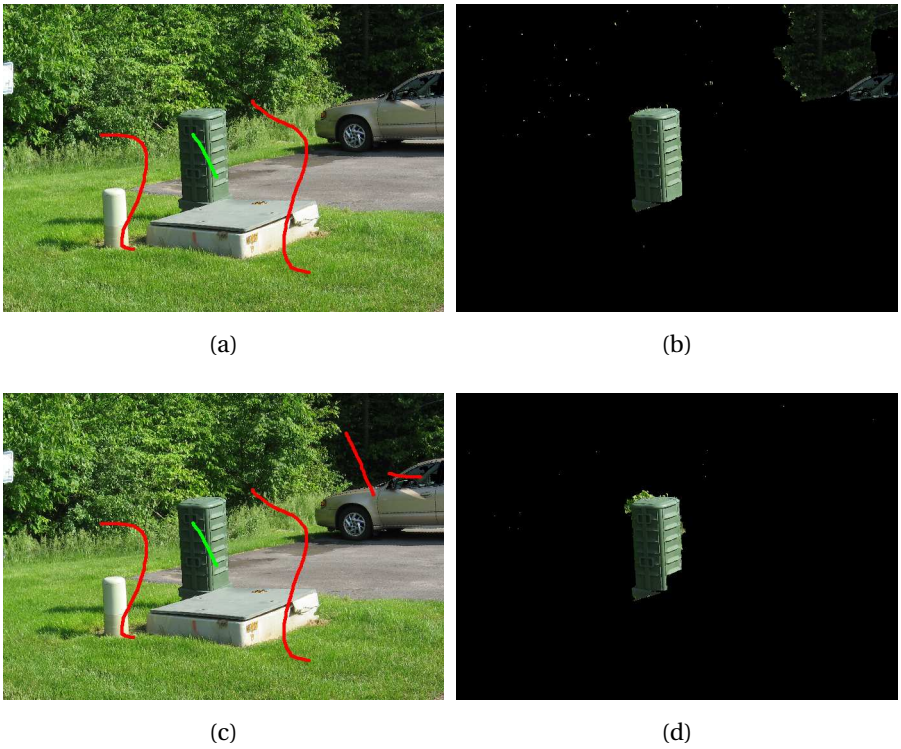
$$J(c) = \sum_{(i,j)\in c} w_{i,j} \tag{5.5}$$

There is an equivalence between the *max-flow* on a graph, and the min-cut of a graph. As graphs are often used to model transportation networks, the problem has been studied in the context of maximizing throughput from one vertex to another, taking the edge weights to be the capacity of each edge. Therefore, algorithms to compute these solutions are often known as "Min-cut/Max-flow algorithms." A famous algorithm to solve the max-flow on a graph (and therefore the min-cut) is the Ford-Fulkerson algorithm [70]. However, these solutions can be quite slow on large graphs, which is the case with our graphs on images (a 1 mega-pixel image has 1 million nodes

in the graph!) Recently, Boykov and Funka-Lea [23] introduced a method to compute this cut extremely efficiently, leading researchers to take advantage of this type of graph formulation extensively [25, 24, 23].

5.1.3 Interactive Refinement

This whole procedure (indicating which pixels belong to the foreground and background, using them to create models of the corresponding components, and computing the minimum cut) is often performed in an interactive way. That is, if the segmentation is determined to be incorrect or needs refinement, the foreground and background scribbles can be improved or extended and the solution recomputed.



(a) (b)

(c) (d)

Figure 5.5: Two steps of an interactive segmentation refinement process. (a) The initial foreground (green) and background (red) strokes. (b) The resulting initial segmentation. We see that a large region in the top right of the image was erroneously included in the foreground. (c) The user has refined the segmentation by adding additional background strokes. (d) The segmentation using these additional strokes is much better than the original segmentation.

5.2 LiDAR Image Segmentation

Our goal in this section is to extend the graph-cut technique described in the previous section to segment 3D objects from LiDAR scans. That is, we ask the user to place some simple strokes on foreground and background objects, and use this information to classify each LiDAR point as either foreground (object) or background. While it might seem intuitive that simply thresholding the distances from the scanner should produce good segmentations, we show that this is usually not the case.

5.2.1 Algorithm Overview

An overview of our proposed algorithm is given here, and detailed in the remainder of this section. Our algorithm is a two step process. First, we segment the depth image alone, producing an under-segmentation of the object. We will show that the region of the object far away from its attachment to the ground is easily segmented in this procedure, and the difficult region near the ground results in a naturally conservative segmentation. Next, we use the resulting foreground pixels from this first segmentation as known foreground pixels to seed a second segmentation, this time using the color information. Additionally, we propose a method to further constrain the second segmentation by computing background pixels near the boundary of the object to prevent this segmentation from crossing the object boundary in regions where the object has been correctly segmented by depth alone. We will show that the resulting composite segmentation is very accurate, and requires much less, if any, iterative refinement than using color-only segmentation techniques.

5.2.2 Computing Edge Weights for LiDAR Graphs

To approach the LiDAR segmentation problem using techniques from image segmentation, we must first represent the LiDAR data in a way that can be represented as a multi-valued function over a graph. We construct a *depth image* from the LiDAR points by computing their distance to the scanner's location, and placing this value in a float-valued image of the same size as the acquisition grid. (See Section 2.1.3.4 for a detailed explanation of this process.) Additionally, as described in Section 3.2, our LiDAR image is accompanied by a registered RGB image so that every scan point has an associated

color. We append the depth image as a 4th component, or channel, of the color image, producing a "color + depth" image, which we refer to equivalently as an $RGBD$ image (indicating the ordering of the channels - red, green, blue, depth). We refer to the i^{th} pixel in this $RGBD$ image as R_i (where we had previously referred to the i^{th} pixel of a standard RGB image as I_i).

5.2.3 Computing N-Weight for LiDAR Pixels

A critical step in the formulation of the graph-cut problem for LiDAR is determining an appropriate distance function between vertices. By using the sum of squared differences in Equation 5.2, we are implicitly valuing differences in each color channel equally. However, it certainly does not make sense to weight differences in the color and depth values equally, since the values in the depth channel have units in meters (typically from $10-20m$ in the scans in this thesis), while the RGB channels have arbitrary units, typically in the range $0-255$. Performing operations like the sum of squared differences directly on these 4-component $RGBD$ pixels does not yield a meaningful result. That is, two pixels that are very similar, say $A = (100,200,50,5)$ and $B = (101,199,48,5.1)$ have approximately the same difference as pixel A and $C = (100,200,50,9)$, even though these pixels are very different (their last component indicates that these pixels are 4 meters different in depth from each other). Because of this, we need some way of adjusting the components so that they lie in meaningful ranges that can be directly compared.

We extend Equation 5.2 with these new types of images by weighting each channel with a scalar α, as shown in 5.6.

$$d(R_i - R_j) = \alpha_r(R_i^r - R_j^r)^2 + \alpha_g(R_i^g - R_j^g)^2 + \alpha_b(R_i^b - R_j^b)^2 + \alpha_d(R_i^d - R_j^d)^2 \tag{5.6}$$

where each α_c is a scalar weight for channel c of the image. We substitute this new difference function directly into Equation 5.1 to obtain the n-weights for our new graph.

One method of setting these α's is to normalize the channels, a common practice in statistical data analysis. To normalize the channels, we find the mean and standard deviation of each of each component over the entire image. Then we subtract the cor-

responding means and divide by the corresponding standard deviations to obtain values have 0 mean and unit variance. For example, for the red component, we compute new values for each pixel as in Equation 5.7.

$$R_i^r = \frac{R_i^{r*} - \mu_r}{\sigma_r} \tag{5.7}$$

Here, R_i^{r*} is the original value of the red component of the i^{th} pixel, μ_r is the mean of the red channel of all pixels in the image

$$\mu_r = \frac{\sum_i R_i^r}{N_R} \tag{5.8}$$

(N_R is the number of pixels in R) and σ_r is the standard deviation of the red component of all pixels in the image

$$\sigma_r = \sqrt{\frac{1}{N_R}\sum_i (R_i^r - \mu_r)^2} \tag{5.9}$$

This procedure is applied to each channel separately. The resulting channels of the image are now directly comparable.

In addition to the usual normalization, we may additionally want to adjust these channel weights to manually set the relative importance of each channel. In the experiments in this chapter, we simply use equal weights on each channel after the channels have been normalized. In the best case, we could use machine learning to pick a set of weights that perform well on the data and problem at hand. However, this would require significant amounts of hand-labeled training data, so we have chosen not to require this in our approach.

5.2.4 Problems with Direct LiDAR Segmentation

Color and depth segmentation each come with their own sets of benefits and drawbacks. In Figure 5.6a, we show the resulting segmentation using both color and depth information separately, as well as the resulting segmentation using the RGBD image described in the previous section, with various channel weights. We see in 5.6c that the color-only segmentation results in significant "bleeding" of the foreground over the object boundary. We note that this type of error in this region of the object

should be easily fixed by using the depth information, as the difference in depth of the object from the background in this region is very large. However, using only the depth information as in Figure 5.6d, though the color bleeding has been resolved, the resulting segmentation is a quite conservative estimate of the object. This is because the depth values near the ground attachment point are very similar inside and outside the object boundary. In Figure 5.6e we show the result of segmenting the object using both color and depth information, directly with the normalized channels described in the previous section. We see that though the segmentation is better than both the color segmentation or depth segmentation alone, there are still errors. Figure 5.6f shows the best result we could achieve by tweaking the weights of the channels manually (we settled upon $\alpha_d = 5.2\alpha_r$ and all of the color channels weighted equally). While this result is reasonable, in this section we discuss a technique which does not require this manual trial-and-error weight setting procedure.

To show that this behavior can not always be achieved by setting the weights to the same values, we provide a second example of the same segmentations on a different data set. In fact, in this case we were unable to find a set of weights which segmented the object appropriately.

In both Figures 5.6d and 5.7d, we see that the resulting segmentation using the depth information alone is a conservative segmentation, or under-segmentation, of the object. That is, though we did not label all of the object points as foreground, we never labeled background points as foreground. The reason for this is shown in Figure 5.8. We can see that for pixels A and B at the top of the object, where the object is far from the ground and distant from the background, the depth differences are high and the weight in Equation 5.6 between adjacent foreground and background pixels is low (easy to cut). On the other hand, for pixels C and D at the bottom of the object, the depth values are almost identical, so the weights in this region are all very large and the boundary is difficult to determine correctly.

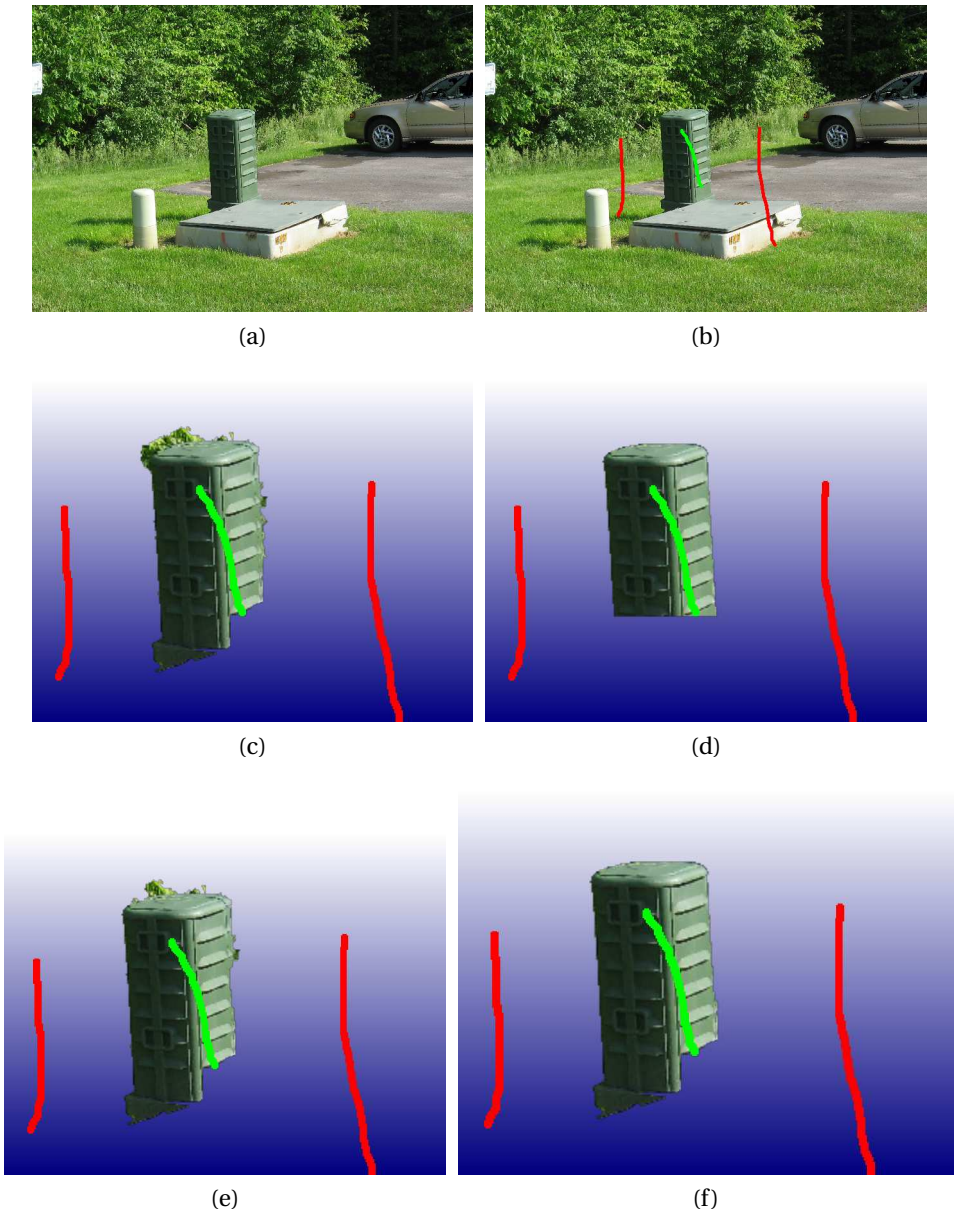


Figure 5.6: Color vs depth segmentation. (a) The original scene. (b) The user provided strokes on the image. (c) The resulting segmentation of the object using color information only. We see that the top of the electric box "bleeds" into the trees in the background because the colors are similar. (d) The resulting segmentation of the object using depth information only. We see that this a conservative estimate of the object. (e) The resulting segmentation of the object using equal weights on each of the normalized RGBD channels. The segmentation is improving, taking on properties from both color and depth segmentation alone. (f) The resulting segmentation of the object using equal weights on the RGB channels and $\alpha_d = 2\alpha_r$. This is as good as we can do without further refinement by the user.

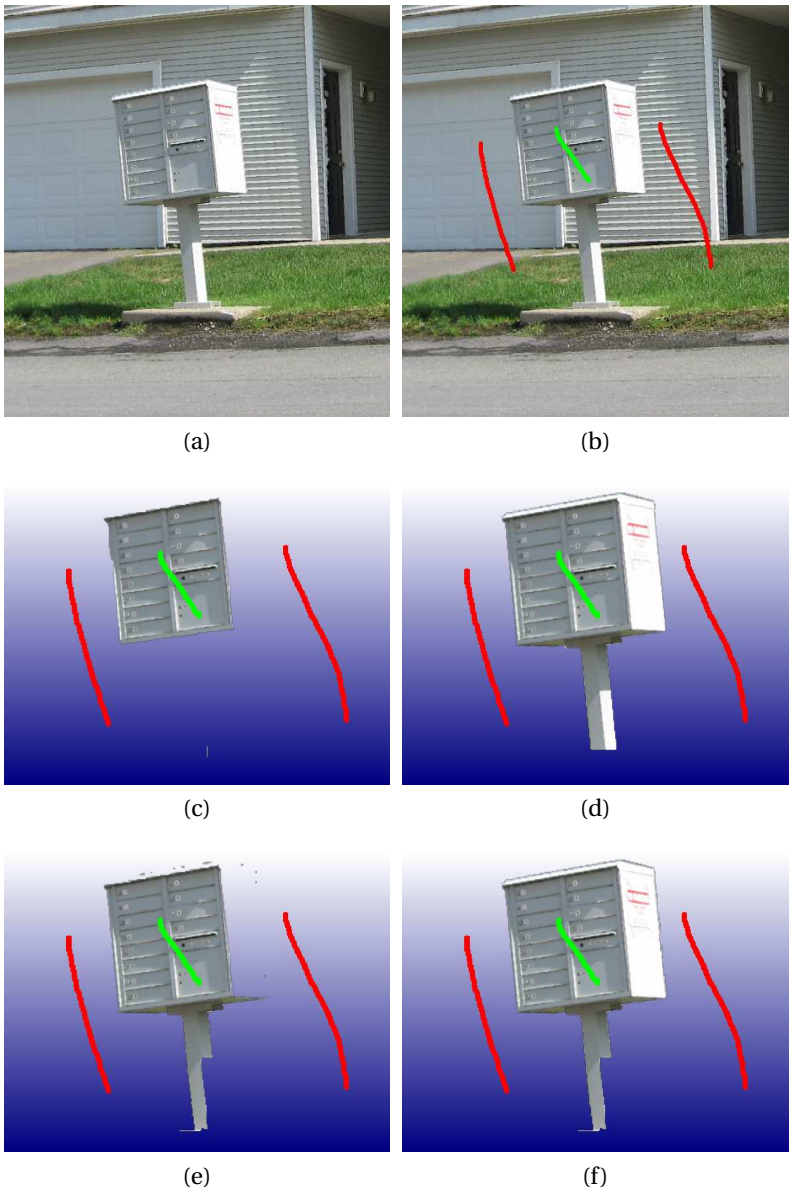


Figure 5.7: Color vs depth segmentation. (a) The original scene. (b) The user provided strokes on the image. (c) The resulting segmentation of the object using color information only. We see that only the front face of the top part of the mailbox has been segmented. (d) The resulting conservative segmentation of the object using depth information only. While the segmentation looks quite good, we are actually missing several inches of the bottom of the post. (e) The resulting segmentation of the object using equal weights on each of the normalized RGBD channels. The segmentation takes on properties from both color and depth segmentation alone. (f) The resulting segmentation of the object using equal weights on the RGB channels and $\alpha_d = 2\alpha_r$. No combination of channel weights seems to produce a better segmentation.

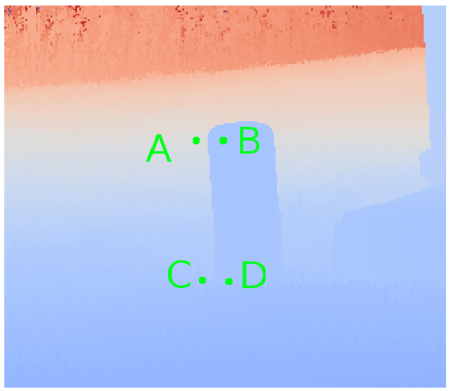


Figure 5.8: The depth image of the electric box scene. We see that points A (background) and B (object) have significantly different depths, making the distinction easy to determine both for a human and for the algorithm. However, points C (background) and D (object) have approximately the same depth, making the determination very difficult.

5.3 Contribution: Two Step LiDAR Segmentation

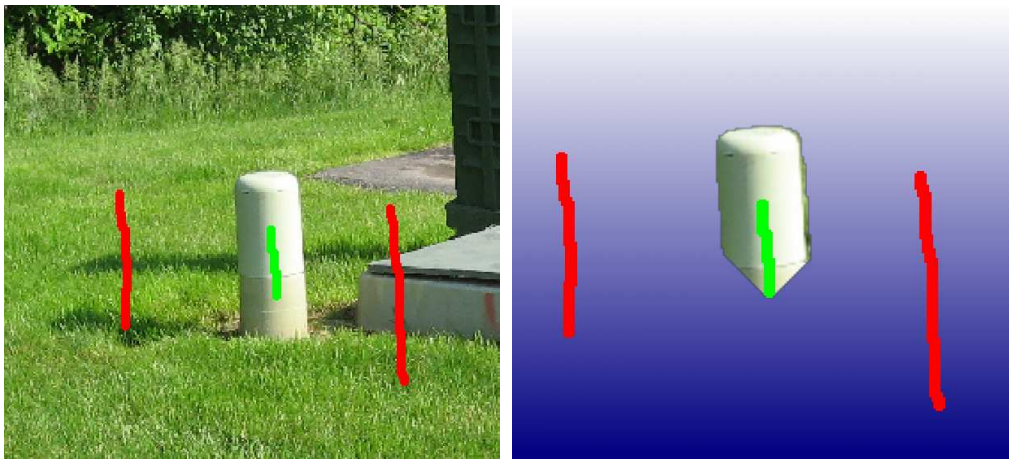
In this section we present a novel two step algorithm for segmenting an object in a LiDAR scan. First, we perform a segmentation on the depth image alone, which results in an under-segmentation of the object. As we described in the previous section, the boundary of the object far away from the ground attachment is easy for the graph cut to find correctly, but as the depth values converge to the same value as the ground, the distance function between two pixels near the ground boundary becomes very small, leading to large edge weights, which the algorithm does not easily cut. To remedy this, we follow this under-segmentation with a second segmentation step. Our key contribution is using the result of the initial depth-only segmentation as the foreground seeds for the second segmentation, as we will show in this section. Additionally, we describe a technique to additionally create new background seeds which ensure the boundary in the region far from the ground is preserved in the second step. We show that this two step technique can produce more accurate segmentations of objects in LiDAR scans than can color-only segmentations, depth-only segmentations alone, or even combined RGBD segmentations. Additionally, little to no user refinement is necessary using our technique.

5.3.1 Step 1: Depth-only Segmentation

We motivate our approach with a simple real-world example. Consider the scene shown in Figure 5.9. After the user marks foreground and background with simple strokes, using a depth-only segmentation produces the result in Figure 5.9b.

5.3.2 Step 2: Refining the LiDAR Segmentation Using Color Information

The result of the depth-only segmentation have given us a much larger set of "definitely foreground" pixels than those provided by the user's strokes. We take advantage of this new knowledge by using these pixels as the initialization of a second segmentation. This time, we directly use the normalized RGBD channels each with unit weight rather than requiring the user to interactively determine weights which produce a good result, as would be required with the sparse initial knowledge alone. The effective result is as if the user, rather than scribbling very casually, had instead



(a) (b)

Figure 5.9: An example initial depth segmentation. (a) The user specified foreground and background pixels in the form of rough strokes. (b) The resulting segmentation using depth information only.

painstakingly very carefully filled in a large piece of the object at a pixel by pixel level. This step of the procedure is shown in Figure 5.10.

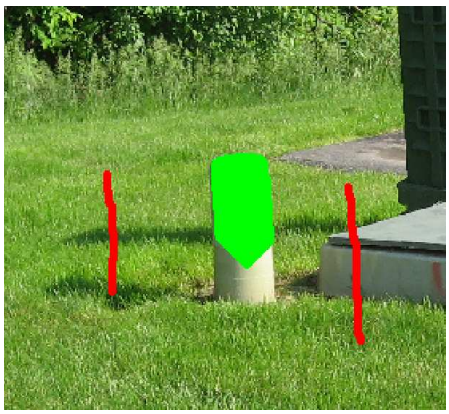


Figure 5.10: The new foreground pixels resulting from the depth segmentation.

5.3.2.1 Background Pixel Inference

If we simply use the foreground pixels from the first step (depth-only segmentation), there is nothing preventing the exact same color bleeding that we observed in a color-only segmentation. In Figure 5.11, we show this effect.

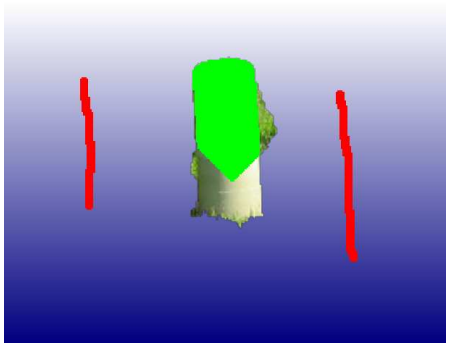


Figure 5.11: Even with the foreground pixels from the depth-only segmentation, using the color information results in bleeding over the edges that were correctly obtained in the depth-only segmentation.

We must indicate the region where we are confident that the depth segmentation was correct, as the depth provides enormous information about the outline of the object. To constrain the graph cut, we must add additional constraints in the form of new "definitely background" pixels. Our goal is to allow the color segmentation to be used only where the depth segmentation is uncertain. To determine the confident pixels, we propose the following procedure. By dilating the region that we have marked as foreground in the initial depth segmentation step and then extracting its boundary, we obtain a set of pixels some of which are background pixels immediately outside the object, while others should be part of the foreground. The boundary of the dilated region is shown in Figure 5.12.

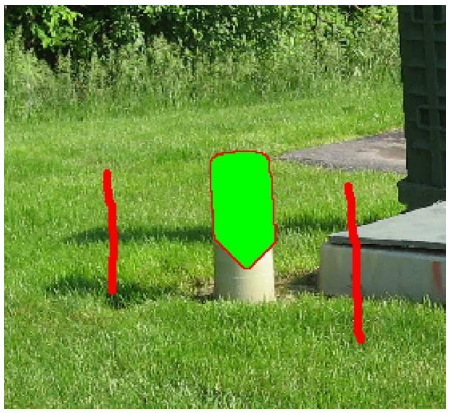


Figure 5.12: Boundary of initial depth segmentation foreground.

To identify which of these pixels are actually background pixels, we perform a test at every proposed background pixel. We compute the depth difference from each proposed background pixel to its neighboring foreground pixels. If the average depth is above a threshold, D_{max}, we are confident that the pixel is a background pixel. If the depth difference is not larger than the threshold, the pixel is discarded and not marked as a background seed. The resulting background pixels are shown in Figure 5.13.

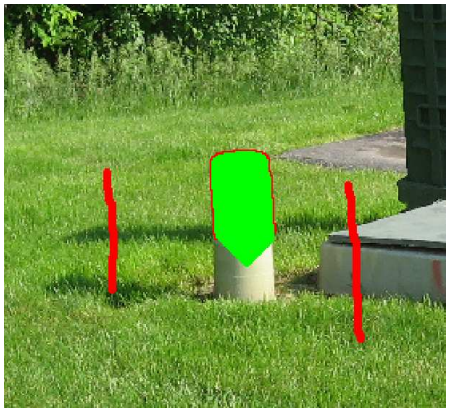


Figure 5.13: Background pixels marked after background inference.

We have heuristically set $D_{max} = .03m$. This value indicates that we believe if a pixel differs in depth from its neighbors by less than this value, it is part of the same object. Of course we cannot apply this idea directly to produce a segmentation — it only applies to pixels on the boundary of our initial depth segmentation. As the lowest values we have observed between the depth of a pixel in the foreground of our initial depth segmentation and its neighboring background pixels is $.4m$., this is a very safe choice, and not at all a sensitive parameter.

5.4 Experiments

In this section, we apply our proposed segmentation algorithm to several real-world data sets. In each example we show, the minimalistic user strokes, the result of the depth-only segmentation, the new seed pixels (foreground and background), and the final segmentation in 2D and 3D.

Figure 5.14 shows our algorithm applied to segment the large electric box from

the electric box data set. We see that the resulting segmentation is very sharp and has accurately separated the object from the background.

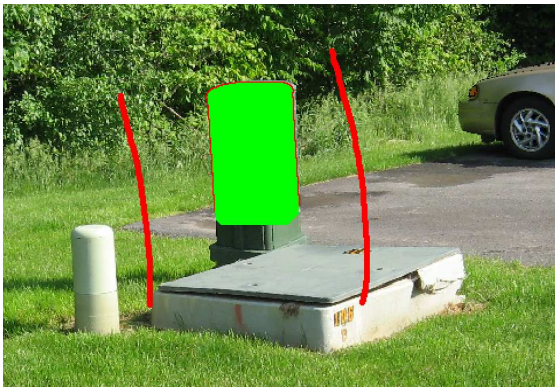
Figure 5.15 shows our algorithm applied to segment the mailbox from the scene. This is a very challenging color segmentation problem as the background (garage door and building siding) is very similar in color to the object. We see that even though the user stroke was only on the large top portion of the mailbox, most of the post was still labeled object in the initial depth segmentation. Our algorithm, without additional stroke refinement by the user, was able to accurately label the remaining portion of the mailbox post as foreground.

Figure 5.16 shows our algorithm applied to segment the trashcan from the scene. This data set is very challenging for a color-only segmentation algorithm as the black wheel is almost indiscernible from the shadow cast by the trashcan. We see that the depth-only segmentation was, as usual, able to accurately segment the top portion of the object. The second segmentation in our procedure was able to separate the bottom of the trashcan from the difficult shadow in the background. We note that a small group of points on the concrete were included in the foreground. As these points are very similar in both color and depth, it is hard for the algorithm to identify them as background.

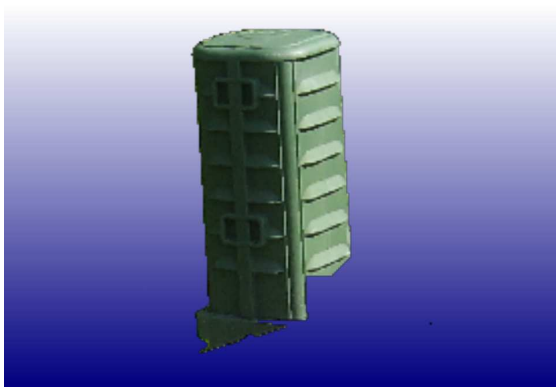
Figure 5.17 shows our algorithm applied to segment the small electric box from the electric box data set. Overall, the result is quite reasonable, but we note two issues. First, there are some grass pixels around the sides of the electric box that have been incorrectly labeled as foreground. This is not an issue with the segmentation procedure, but rather the input data alignment (the recoloring procedure we described in Section 3.2). Additionally, we see that several pixels in the grass in front of the object have been incorrectly labeled as foreground. We attribute this to the noisy-ness of both the depth and color values in highly textured regions such as grass. The effect is not intrusive when viewed in 2D, but in the resulting 3D segmentation the error is more visible.

(a) The initial image (RGBD).

(b) The initial strokes on the large electric box.



(c) The seeds computed from the depth-only segmentation.



(d) Our segmentation of the large electric box (2D).

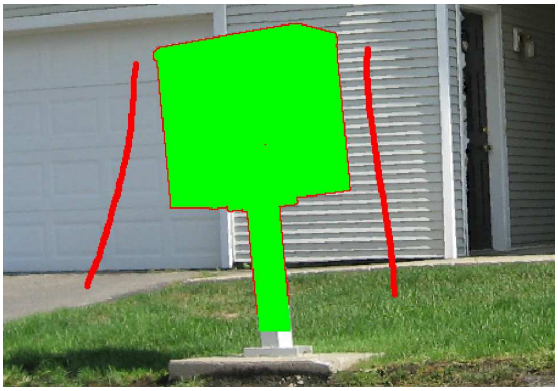
(e) The 3D scene.

(f) Our segmentation of the large electric box (3D).

Figure 5.14: The final segmenting of an electric box.

(a) The initial image (RGBD).

(b) The initial strokes on the mailbox.



(c) The seed pixels computed from the first segmentation.

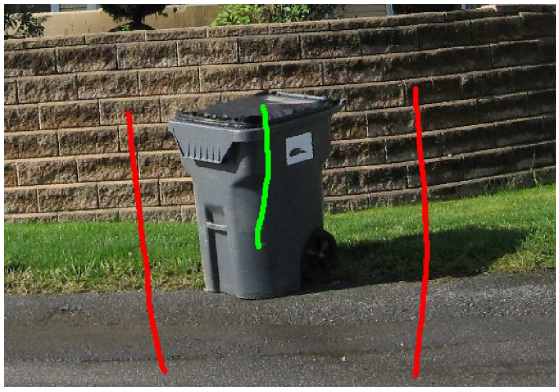
(d) Our segmentation of the mailbox (2D).

(e) The 3D scene.

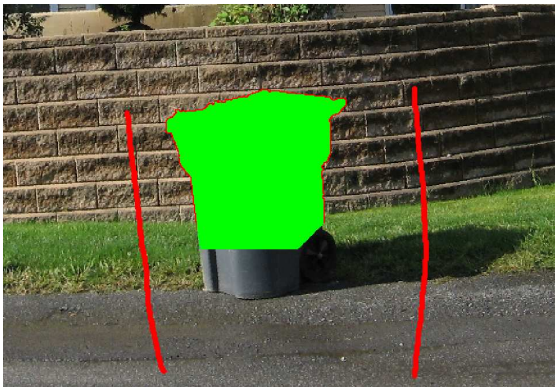
(f) Our segmentation of the mailbox (3D).

Figure 5.15: Our LiDAR segmentation algorithm on the mailbox dataset.

(a) The initial image (RGBD).



(b) The initial strokes on the trashcan.



(c) The seeds computed from the initial segmentation.

(d) Our segmentation of the trashcan in 2D.

(e) The 3D scene.

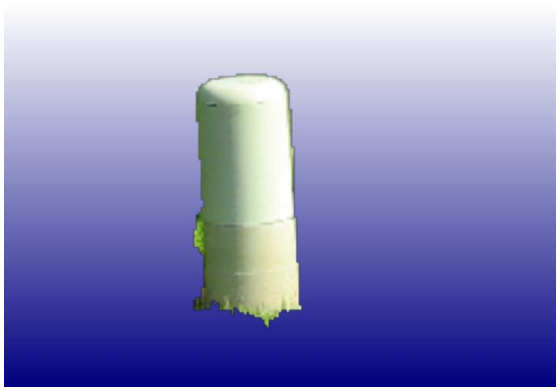
(f) Our segmentation of the trashcan in 3D.

Figure 5.16: Our LiDAR segmentation algorithm on the trashcan data set.

(a) The initial image (RGBD).

(b) The initial strokes on the small electric box.

(c) The computed seeds from the depth-only segmentation.



(d) Our segmentation of the small electric box (2D).

(e) The 3D scene.

(f) Our segmentation of the small electric box (3D).

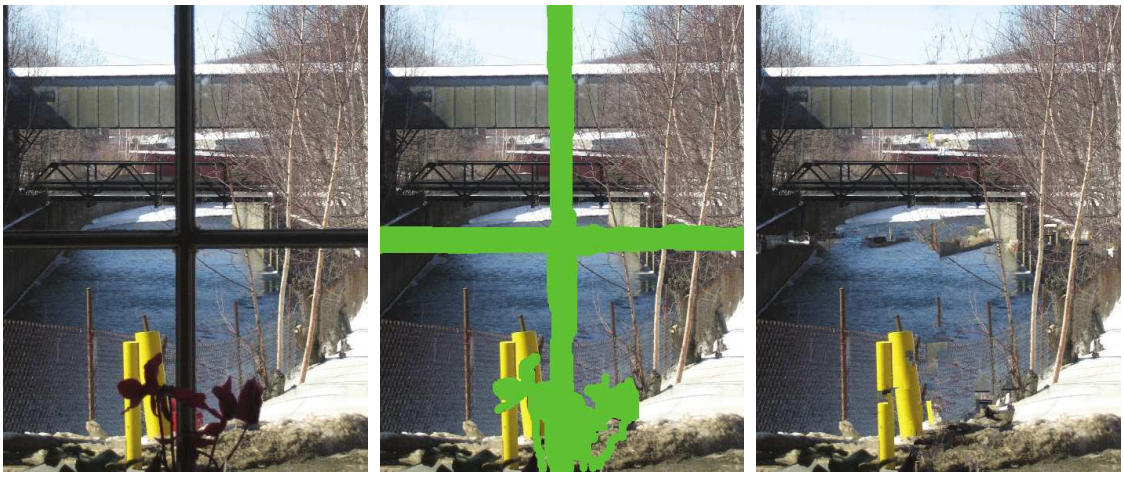
Figure 5.17: The final segmenting of an electric box.

5.5 Discussion

In this chapter, we have presented a novel two-step algorithm for segmenting objects in a LiDAR scan. We have presented several successful segmentations of real-world objects in challenging outdoor scenes. We have showed that minimal user stroking is required to achieve accurate boundaries in both depth and color. The resulting segmentation in 3D accurately portray the objects, and could serve as excellent input to the next step in the pipeline of an object recognition system, or object model database.

CHAPTER 6
LiDAR Inpainting

There has recently been a lot of work on the image inpainting problem in standard RGB images. The user indicates a region in an image, and an algorithm is applied to automatically fill in the region with plausible background texture. Image editing tools like Photoshop now routinely include methods to seamlessly remove objects from an image, as well as repair artifacts. An example of the goal of image inpainting is shown in Figure 6.1. The goal in this example is to remove the window frame from the image.



(a) (b) (c)

Figure 6.1: A demonstration of image inpainting. (a) The original image. (b) The region to inpaint is shown in bright green. The goal in this example is to remove the window frame from the image. (c) The inpainted image. If an observer was presented with only this image, they would likely not notice that it had been modified. (Images from [127])

In this chapter, we introduce an algorithm to "complete" LiDAR images by filling large holes, that might be created by occluded regions or the removal of segmented objects as described in the previous chapter. As a contrast to image inpainting, an inpainted LiDAR scan can be viewed from a different perspective, exposing background texture not present in the original scan. To motivate the goal of this chapter, we show a example result of our algorithm in Figure 6.2.

(a) (b)

Figure 6.2: A demonstration of LiDAR inpainting. (a) A 3D scene of a mailbox with a building in the background. (b) The mailbox has been removed and the scene behind it has been filled using our proposed LiDAR inpainting algorithm.

Our work draws from and extends research from two fields; patch-based image inpainting and image reconstruction from gradients. As in the previous chapter, we find that image-based techniques cannot be directly applied to work with LiDAR data. In this chapter, we propose new techniques to fill large holes in LiDAR data. We first detail the image-based methods which we build upon, then introduce our new algorithm and show several real-world examples of successful filling texture and structure in LiDAR scans.

6.1 Patch-Based Image Inpainting

Patch-based image inpainting is the process of filling a hole in an image by copying patches of pixels from elsewhere of the image into the hole. One iteration of this type of algorithm is shown in Figure 6.3.

Before proceeding, we must define some terms. The **hole**, Ω, is the portion of the image, I, that we wish to fill. The boundary of this hole we denote $\partial\Omega$. The **source region**, S, is the portion of the image which is known (is not part of the hole) at the beginning of the procedure, or has been already filled. A **target patch**, ψ_T, is any patch whose center is on the hole boundary. We denote the target patch centered at pixel i by

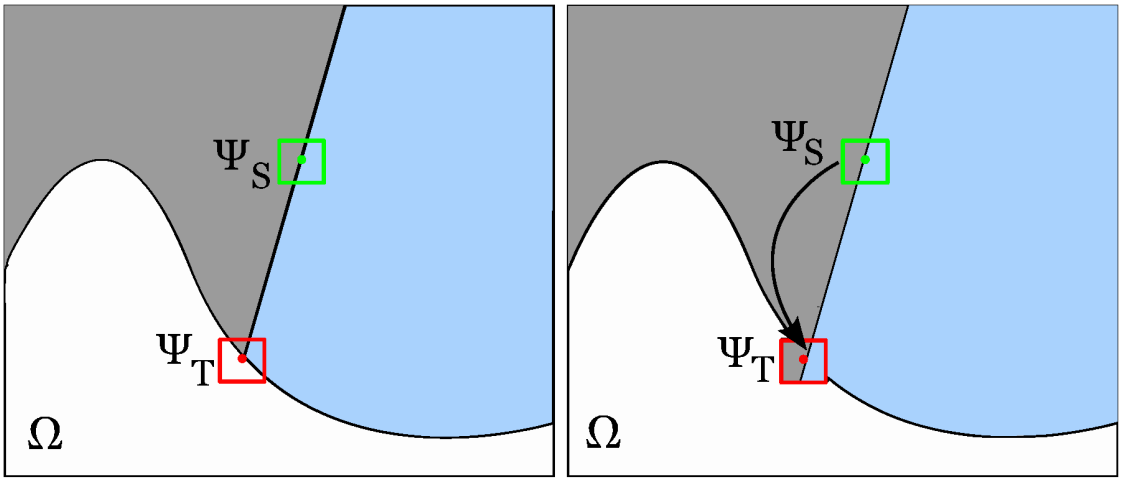


Figure 6.3: A conceptual demonstration of patch-based inpainting. (a) Image to be filled and potential source patches. (b) The target patch properly filled.

ψ_T^i. A **source patch**, ψ_S is a patch entirely composed of pixels from the source region. Again, the source patch centered at pixel i is denoted ψ_S^i. The size of the patches must be specified by the user. Typically we specify a "half-width", h rather than the side length because this guarantees the patches to have odd size, which ensures they have a well defined center pixel, which is important because our definition of a target patch is defined by its center pixel. The patch side length l is simply $l = 2h + 1$. These terms are illustrated in Figure 6.5.

Target patches have two distinct regions. The region that overlaps the hole we call the *hole region*, $\psi_T \cap \Omega$. The region that overlaps the source region we call the *valid region*, $\psi_T \cap S$. Although source patches have only one region (all pixels are "valid", in that there are no missing pixels due to the hole), when performing a comparison between a source and target patch, we address regions of the source patch corresponding to the regions of the target patch. For example, we can refer to the "hole region of the source patch", which indicates the region of the source patch that corresponds to the hole region of the target patch. As this terminology is used extensively, we have labeled these regions in Figure 6.5.

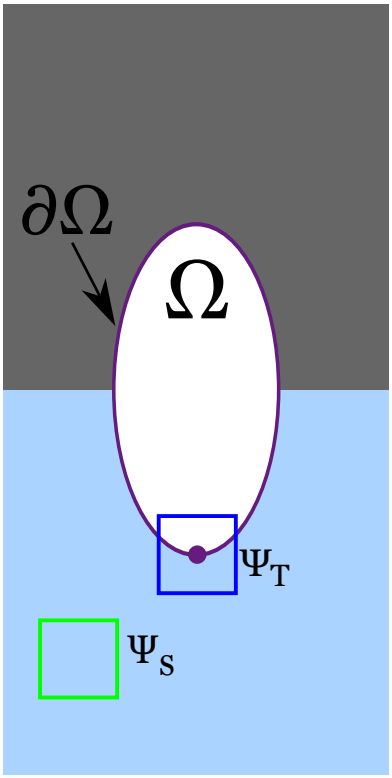
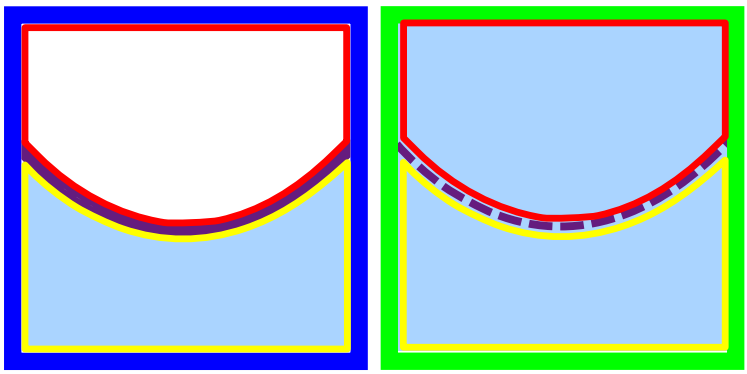


Figure 6.4: Important terminology for patch-based inpainting. The hole to be inpainted, Ω, is shown in white. A target patch ψ_T, is shown in dark blue. A source patch ψ_S, is shown in green.



(a) (b)

Figure 6.5: Important regions for patch-based inpainting. (a) The target patch from Figure 6.4 with its regions outlined. The hole region is outlined in red, while the valid region is outlined in yellow. The hole boundary is indicated with a purple line. (b) The source patch from Figure 6.4 with its regions corresponding to the target patch's regions outlined. The valid region of the source patch is outlined in yellow, while the hole region is outlined in red. The hole boundary is indicated with a dashed purple line to remind us that this boundary does not actually exist in the image, but rather divides the patch in the same fashion as the hole boundary divides the target patch.

Additionally, to ease the notation, when we refer to $\psi_T(q)$ and $\psi_S(q)$ in the same equation, q refers to a specific corresponding pixel in the target and source patches. This is shown in Figure 6.6.

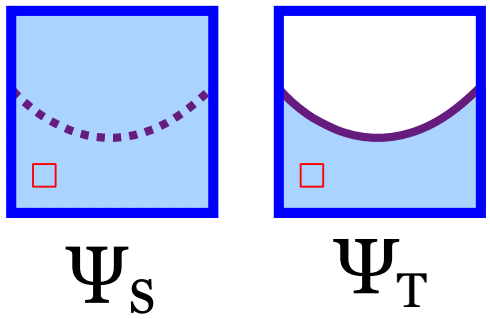


Figure 6.6: Corresponding pixels in a source and target patch. The red square indicates a pixel in the same position relative to its patch in both patches. The two red pixels are corresponding pixels in the two patches.

A typical patch-based inpainting algorithm proceeds as follows:

1. Determine which target patch fill.
2. Search for the "best" source patch to use to fill the selected target patch.
3. Copy the region of the source patch corresponding to the hole region of the the target patch into the target patch.
4. Update the hole, the hole boundary, and potentially add new source patches.
5. Repeat until the target region contains no pixels.

The two most important steps in this algorithm are choosing which target patch to fill and deciding which source patch to copy into the chosen target patch. We discuss the details of these steps in the following sections. Particularly, we discuss the technique described in a seminal work by Criminisi [37].

6.1.1 Selecting a Target Patch

Since most patch-based inpainting methods are greedy, selecting a good order in which to fill the hole is very important. There has been a lot of effort dedicated to determining a good order in which to fill target patches. In a seminal paper, Criminisi [37] noted the importance of filling patches that continue linear structures first. Their argument was that these linear structures are critical to successful human interpretation of the resulting image. If the linear structures are broken, it is almost always obvious that the image has been modified. Other researchers have gone as far as to allow the user to draw lines to force inpainting to first proceed along these regions (i.e. [146]). In this chapter, we do not require any such user guidance.

Criminisi described two separate ideas to be applied simultaneously to determine the priority of a patch. First, filling patches at the hole boundary is significantly easier than filling patches deep within the hole. That is, we have some context in the form of known pixels at target patches on the boundary. Furthermore, as the hole starts to be filled, we should have a high *confidence* (the term used by [37] to describe this phenomenon) that the pixels in the known region that were known at the outset of the algorithm are correct, and then gradually have less confidence in pixels that have been filled by the algorithm. Second, as continuing linear structures is critical, we should prefer to fill target patches that contain linear structures in the image. This part of the priority computation is known as the *data* term, as it is based on actual pixel values in the image, as we will see in this section.

At each iteration of the inpainting algorithm, a *priority* value, $P(\psi_T^i)$, is computed for each target patch. We then select and fill the target patch with the highest priority. This is expressed mathematically in Equation 6.1.

$$\psi_T = \arg\max_{i \in \partial\Omega} P(\psi_T^i) \tag{6.1}$$

The first idea described in [37] is to compute the confidence of a particular target patch. We think of the confidence of a target patch as a measure of the amount of reliable information contained in the patch. To compute the confidence value for each target patch, we use Equation 6.2.

$$C(\psi_T) = \frac{\sum_{q \in \psi_T \cap S} C(q)}{|\psi_T|} \tag{6.2}$$

where $|\psi_T|$ is the area (number of pixel) of the target patch. That is, the confidence of a target patch ψ_T is equal to the sum of the confidence of the pixels in its source region divided by the area of the patch. This naturally gives preference to patches with have a large source region, as well as those which are surrounded by highly confident pixels. Of course, since this is a recursive definition (the priority of a patch depends on the priority of each pixel in the patch), we must initialize the confidence values of each pixel prior to beginning the inpainting. We do this by setting the confidence of pixels outside of the hole to 1 (fully confident) and the confidence of pixels inside of the hole to 0 (completely unsure). In Figure 6.7, we show the confidence values of the hole pixels throughout the inpainting process.

(a) (b) (c) (d)

Figure 6.7: The confidence map as patches are filled. (a) An image with a hole to be inpainted shown in bright green. (b) The confidence of pixels after the first patch is filled is indicated by their brightness. White indicates the known region, while black indicates the hole. We see that since the patch that has been filled was surrounded by approximately half known pixels (confidence = 1) and half unknown pixels (confidence = 0) that its confidence is an intermediate value, near 0.5. (c) The confidence map after 70 iterations. (d) The confidence map at the end of the inpainting. We see that the confidence gets lower as we move towards the center of the hole.

Figure 6.7 was generated by using the confidence value alone as the filling priority. We show the resulting ordering in Figure 6.8.

(a) (b)

Figure 6.8: An example of the fill ordering produced by the confidence approach. (a) The image to be filled. The region to be filled is shown in bright green. (b) The order that the patches were filled is indicated by increasing brightness. We see that the patches are filled near the original hole boundary first before moving inward toward the center of the hole.

The second idea popularized in [37] is filling target patches containing linear structures first. To describe this procedure, we must introduce two new ideas. First, we denote the normal vector (the vector orthogonal to the tangent vector) of the hole boundary $\partial\Omega$ at pixel p as n_p. Second, we define the *isophote* directions in an image as the gradient vectors rotated by 90 degrees. These isophote directions indicating the direction of least increase rather than greatest increase, which is exactly the definition of the direction of a linear structure. We denote the isophote direction at a point p as $\nabla I_p^{\perp}$. A boundary normal and isophote direction are shown for a target patch centered at a pixel p in Figure 6.9.

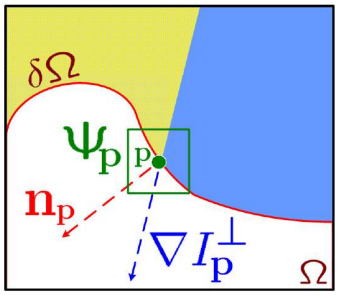


Figure 6.9: The isophote direction and the boundary normal at a point. (Figure from [36]).

To encourage patches containing these linear structures to be selected first, [37] introduced a *data term* which has higher values for target patches that lie on these linear structures, and where their isophote direction is aligned with the hole boundary normal direction. This condition is expressed in Figure 6.3.

$$D(\psi_T^i) = \frac{|\nabla I_i^{\perp} \cdot n_p|}{255} \tag{6.3}$$

In [37], the final priority of each target patch was simply taken to be the product of the confidence and data terms, as in Equation 6.4.

$$P(\psi_T) = C(\psi_T)D(\psi_T) \tag{6.4}$$

We note that in this formulation, the normalization term of 255 in the data term is actually unnecessary. Recall from Equation 6.1 that we will only use this priority function to as an objective function to be maximized. Substituting our priority function into 6.1 we obtain

$$\psi_T = \arg\max_{i \in \partial\Omega} \frac{\sum_{q \in \psi_T \cap S} C(q)}{|\psi_T|} \frac{|\nabla I_i^{\perp} \cdot n_p|}{255} \tag{6.5}$$

the solution to which is identical to that of

$$\psi_T = \arg\max_{i \in \partial\Omega} \frac{\sum_{q \in \psi_T \cap S} C(q)}{|\psi_T|} |\nabla I_i^{\perp} \cdot n_p| \tag{6.6}$$

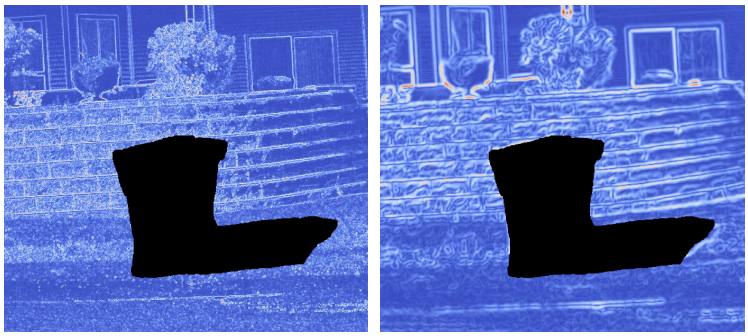
Figure 6.10 shows the fill order using the priority term including both the confidence and isophote direction contributions.

In real-world images with natural textures, the isophote directions are extremely noisy. Therefore they do not always present as strong as we would expect in some regions, and the fill order is not as predictable as we would like. For example, we would like to think that the boundary between the grass and the brick wall in this example would have very strong isophotes, but in fact the strongest isophote directions are those produced by the harsh shadows between rows of bricks, as well as those simply due to areas of high texture in the grass. Even at different scales (blurring the image before computing the isophote directions, the result is not what we would hope. Figure

(a) (b) (c)

Figure 6.10: An example of the fill ordering produced by the joint confidence and isophote direction preference ordering. (a) The original image with the image to be inpainted shown in bright green. (b) The magnitude of the isophote directions (blue is weak, red is strong). (c) The order that the patches were filled is indicated by increasing brightness. We see that unlike with the confidence term only, patches are not necessarily filled in a strict outside-in fashion where the isophote magnitudes are strong enough to give preference to the linear structures.

6.11 shows the magnitude of the isophotes for the original image and a blurred version of the image.



(a) (b)

Figure 6.11: An example of the effect of scale on the isophote magnitudes. (a) The isophote magnitudes computed from the original image. (b) The isophote magnitudes of a blurred version of the image.

Finally, we note that care must be taken when computing the gradient (and therefore isophote directions) of an image near a hole. In a standard image processing problem, the gradient of an image is computed by convolving edge detecting filters with the image. However, if we naively compute the gradient of an image with a hole in this

fashion, the gradient values computed at the hole boundary are undefined, as they rely on unknown values. These undefined values are problematic, as we need the isophote directions exactly at the hole boundary to compute the data term shown below. To address this issue, we immediately dilate the inpainting mask specified by the user. This allows us to compute the isophote directions in the typical fashion, as the input image is well defined everywhere we need it to be to accurately compute the gradient at this new hole boundary. In doing this we have made the problem slightly harder by making the hole slightly bigger, but in practice this is not problematic.

6.1.2 Finding the Best Source Patch

Once a target patch is selected, we must find the "best" source patch to copy into its location. To do this, we must formulate a patch difference function, $D(\psi_T, \psi_S)$, and minimize this function over all source patches available in the image, as shown in Equation 6.7.

$$\psi_S^{best} = \arg\min_{i \in I-\Omega} D(\psi_T, \psi_S^i) \tag{6.7}$$

To compute the difference between a source patch and a target patch, the most common technique is to compute the sum of squared differences of colors at corresponding valid pixels, as shown in Equation 6.8.

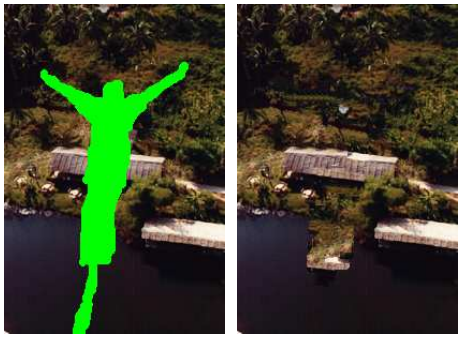
$$SSD(\psi_T, \psi_S) = \sum_{q \in \psi_T \cap (I-\Omega)} \big(\psi_T(q) - \psi_S(q)\big)^2 \tag{6.8}$$

This patch comparison function is very popular because of its simplicity and efficiency, allowing a very large number of source patches to be evaluated quickly. While this is the patch comparison function we use throughout this chapter, we discuss in detail in Chapter 7 why this does not always produce accurate matches.

6.1.3 Patch-Based Inpainting Example

Figure 6.12 shows an inpainting example on a real image. This result shows the typical quality of inpainting that the algorithm produces. This example took about 30 seconds on a Pentium 4 3GHz processor with a 206x308 image and a patch radius equal

to 5.



(a) (b)

Figure 6.12: Realistic demonstration of exemplar-based inpainting. (a) Image to be filled. The region to inpaint is shown in bright green. (b) The result of the inpainting.

Notice that although the resulting image looks feasible on its own, when we look at Figure 6.12a, the completion we would expect has the shoreline completed much more directly. We show the implications of this problem for our work to inpaint holes in LiDAR data in Section 6.6.

6.2 Reconstructing an Image from its Gradients

There has been much recent research interest in gradient-domain techniques for image processing. Pérez et al. [124] applied gradient-domain techniques to convincingly copy large regions of one image into an entirely different image. Bhat et al. [18] presented a generalized framework for gradient-domain image filtering that can produce several types of manipulated images. Such techniques rely on the problem of reconstructing an image from its gradients, which we summarize below.

We begin with an intensity image $I(x, y)$, a target region Ω inside the image, and the desired gradients $G_x(x, y)$ and $G_y(x, y)$ inside Ω. We wish to reconstruct new intensities $I^*(x, y)$ inside Ω subject to the constraint that I and I^* agree on the hole boundary $\partial\Omega$. That is, we want to solve

$$\min_{I^*(x,y)\in\Omega} \iint_{\Omega} (I^*_x(x,y) - G_x(x,y))^2 + (I^*_y(x,y) - G_y(x,y))^2 \; dx \; dy$$
$$s.t. \;\; I^*(x,y)|_{\partial\Omega} = I(x,y)|_{\partial\Omega} \tag{6.9}$$

Using the Euler-Lagrange equation from variational calculus, it can be shown that the solution to Equation 6.9 satisfies Equation 6.10 below:

$$\nabla^2 I^*(x,y) = \text{div}\,(G_x(x,y), G_y(x,y)) \;\; \forall (x,y) \in \Omega$$
$$s.t. \; I^*(x,y)|_{\partial\Omega} = I(x,y)|_{\partial\Omega} \tag{6.10}$$

where ∇^2 represents the image's Laplacian and div is the divergence of a 2D vector field. We discuss the solution to a discretized version of this equation in Section 6.4, which results in a simple system of linear equations. The approach is applied to color images by processing each channel independently.

6.3 A Framework for Depth Inpainting

In this section, we discuss an extension to patch-based inpainting to fill holes in depth images. We show why filling holes directly in the depth image is not appropriate, and propose a solution to these problems in Section 6.4.

Modern LiDAR scanners typically produce a grid of colored 3D points. That is, at each point, we know the depth (i.e., distance) from the scanner, as well as an RGB value associated with the point (usually obtained by a collocated camera). We can view the resulting dataset as a 4-channel RGBD image (Red, Green, Blue, Depth) over a 2D pixel grid. As a first attempt at inpainting holes in this type of data, we could simply extend the technique described in Section 6.1 to operate on these RGBD images. In this case, Ψ_S and Ψ_T are patches of 4D vectors defined over the 2D domain (x, y). The distance function $d(\Psi_S, \Psi_T)$ in Equation 6.7 is defined as the sum of squared differences of corresponding RGBD vectors at non-hole pixels. Before comparison, we normalize the channels in each RGBD image so that the standard deviations of the RGB channels sum to the standard deviation of the depth channel, using the same technique

discussed in Section 5.2.3.

Patch-based image inpainting relies on the idea that a patch that "looks like" the one we would expect to appear in the hole exists somewhere else in the image. In many RGB images, this is indeed the case. However, in depth images, this is typically not the case. For example, consider Figure 6.13, in which a hole interrupts a planar surface seen from above. Though several patches with the correct structure are available, no source patch exists that has the correct depth value at the interior of the hole.

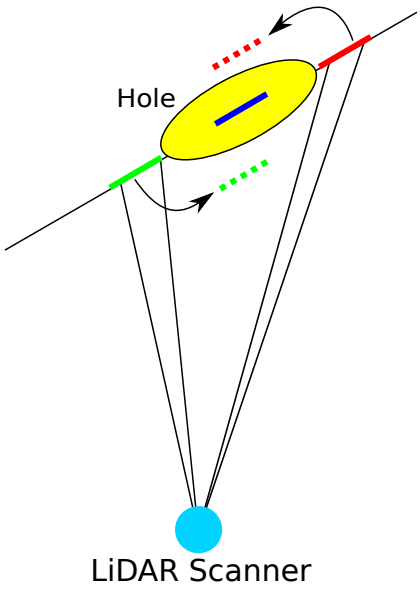
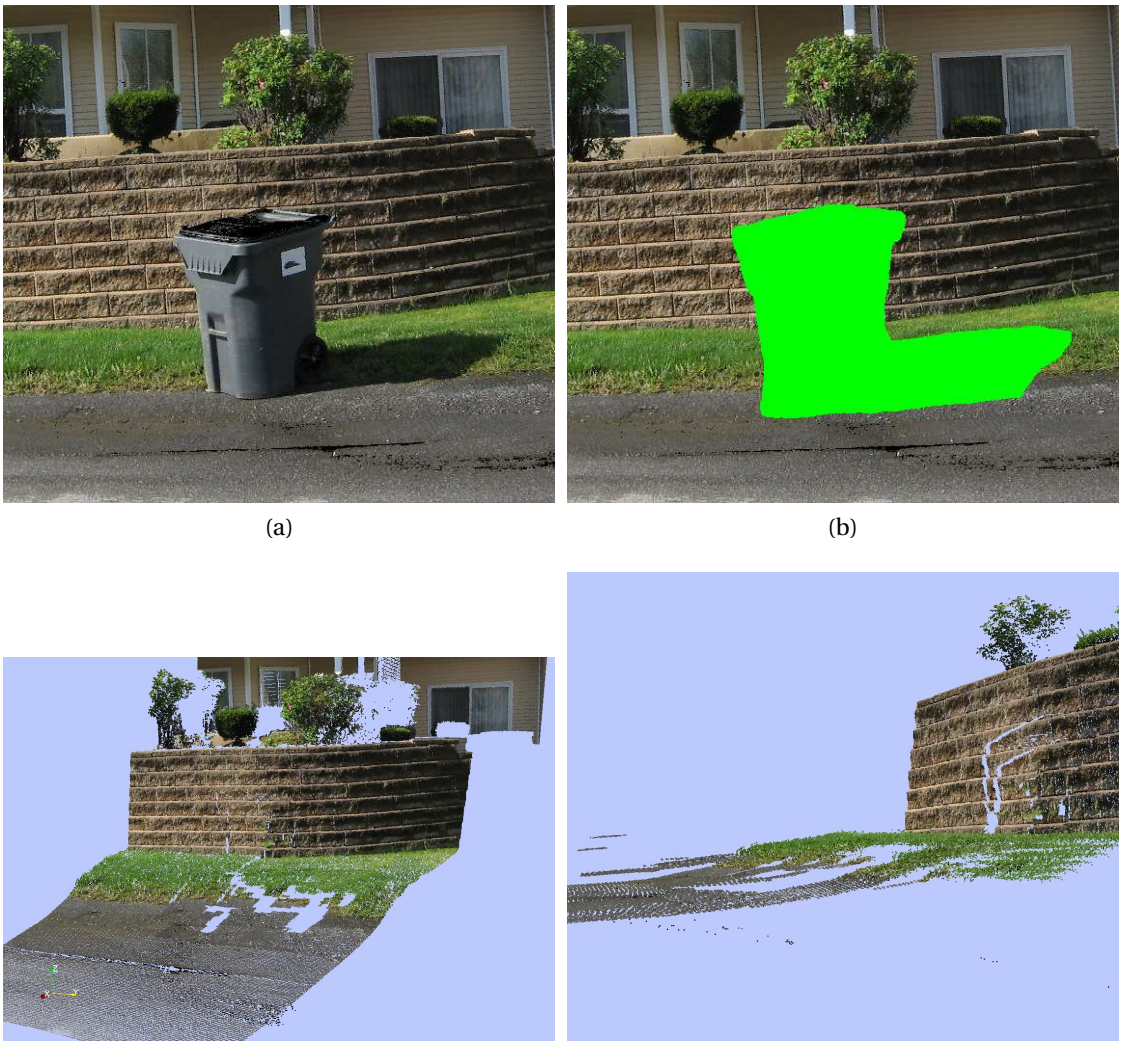


Figure 6.13: An illustration of why direct depth inpainting fails. We wish to fill the hole (yellow) by copying an existing depth patch to the location of the blue patch. Unfortunately, the closest patches in the depth image, though having the structure we need, do not occur at the appropriate depth. Using the green patch would result in 3D structure in front of the appropriate location (the green dashed patch), and using the red patch would result in 3D structure behind the appropriate location (the red dashed patch).

An example of this problem in a real data set is shown in Figure 6.14. We see that many of the patches that were copied were actually located at incorrect depths.

Another approach one might investigate is inpainting the RGB values as usual and then finding the smoothest possible depths to fill the hole. While this may work on very small holes or holes that appear in planar surfaces, in Figure 6.15 we show that the result is usually unacceptable for large holes with complex backgrounds. In Figure

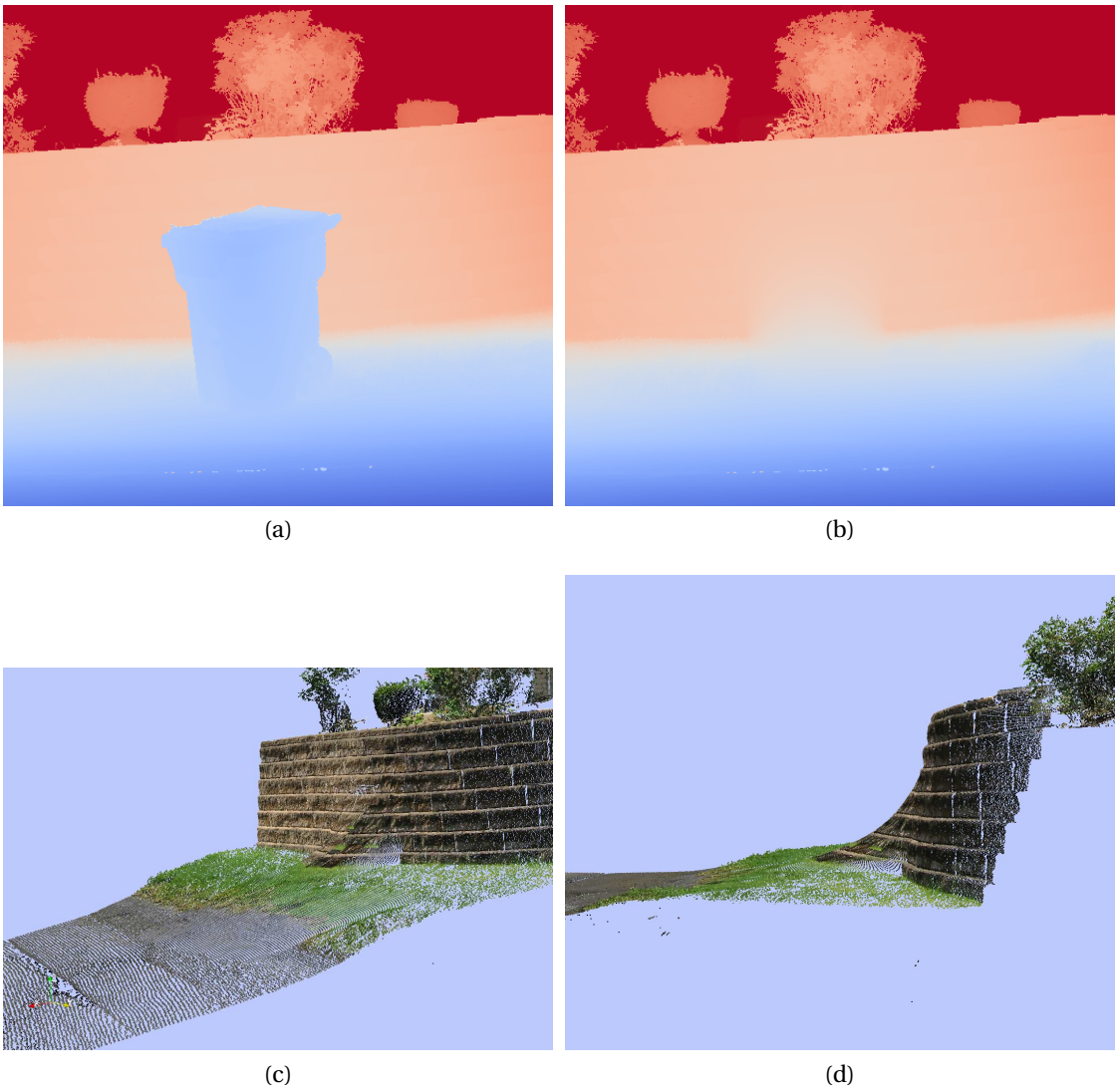


(a) (b)

(c) (d)

Figure 6.14: A demonstration of the result of directly inpainting a depth image. (a) The image associated with the original LiDAR scan. (b) The region to inpaint is indicated in bright green. (c) The structure after inpainting directly in the RGBD image. (d) A side view showing many patches at incorrect depths.

6.15b the hole in the depth image has been smoothly filled, resulting in very incorrect 3D structure, shown in Figure 6.15d.



(a) (b)

(c) (d)

Figure 6.15: A demonstration of smoothly filling a hole in the depth image. (a) The depth image corresponding to Figure 6.14a (blue = close to the scanner, red = far from the scanner). (b) The resulting depth image after removing the trashcan and filling the hole with a smooth surface. We note that the sharp edge at the boundary between the wall and the ground is not preserved. (c) The resulting 3D structure. (d) A side view of the 3D structure. It is clear that this result is unacceptable, since a surface that does not make sense in the scene has been created.

6.4 Inpainting 3D Structure Using Depth Gradients

To prevent the problem of copying depth patches which have very similar structure but different absolute depth values, we instead work in the depth image gradient domain. We observe that depth patches with the correct structure to complete the hole

are typically available in the known region of the image.

Similar to the RGBD images discussed in Section 6.3, we now construct a 5-channel image consisting of the RGB values, as well as the x and y components of the depth image gradient $D_x(x,y)$ and $D_y(x,y)$. We normalize the channels of this image so that the sum of the standard deviations of the R, G, and B channels equals the sum of the standard deviations of the D_x and D_y channels.

We perform the inpainting procedure in these 5-channel $RGBD_xD_y$ images, where the patch distance function now operates on 5D vectors defined over the 2D pixel domain. After inpainting the hole by cutting and pasting patches, we can color the new scene points directly using the first three channels of the resulting patches. However, an additional step is now required to obtain the depth values for the new points.

We have the desired depth gradient $(D_x(x,y), D_y(x,y))$ inside the hole, but what we need is the actual depth in the hole, $D(x,y)$. To perform this reconstruction of the depth image from its gradients, we apply the techniques described in Section 6.2. Here, $G_x(x,y)$ and $G_y(x,y)$ in Equation 6.9 are exactly our inpainted depth image gradients. We know the depth values immediately outside of the hole, so these serve as the boundary condition for the reconstruction problem. The system of equations that must be solved for $D^*(x,y)$, the reconstructed depth image, is shown in Equation 6.11.

$$\begin{aligned} \nabla^2 D^*(x,y) &= \frac{\partial D_x}{\partial x}(x,y) + \frac{\partial D_y}{\partial y}(x,y) \quad \forall (x,y) \in \Omega \\ s.t.\ \ D^*(x,y)|_{\partial\Omega} &= D(x,y)|_{\partial\Omega} \end{aligned} \tag{6.11}$$

Explicitly, a pixel whose 4-neighbors are fully inside the hole generates Equation 6.12.

$$D^*(x+1,y) + D^*(x-1,y) + D^*(x,y+1) + D^*(x,y-1) - 4D^*(x,y) = \frac{\partial G_x(x,y)}{\partial x} + \frac{\partial G_y(x,y)}{\partial y} \tag{6.12}$$

A pixel that has at least one 4-neighbor outside the hole generates an equation similar to Equation 6.13, which shows the case where the pixel $(x+1,y)$ is outside the hole:

$$D^*(x-1,y)+D^*(x,y+1)+D^*(x,y-1)-4D^*(x,y)=\frac{\partial G_x(x,y)}{\partial x}+\frac{\partial G_y(x,y)}{\partial y}-D(x+1,y) \tag{6.13}$$

Thus, if there are N pixels in the whole, Equations 6.12-6.13 can be written as a matrix equation $Ad = b$, where A is a known $N \times N$ matrix, b is a known $N \times 1$ vector, and d is an unknown $N \times 1$ vector containing the depths to be determined. Since A is extremely sparse, containing 5 or less non-zero entries per row, this system can be solved very efficiently, even for large N.

Once we have solved for the depths in the hole, we construct the new 3D points by placing points along the original LiDAR rays at the distances prescribed by the new depth image. We describe the entire procedure algorithmically in Algorithm 1.

Algorithm 1 FillLargeHoles(LiDAR scan)

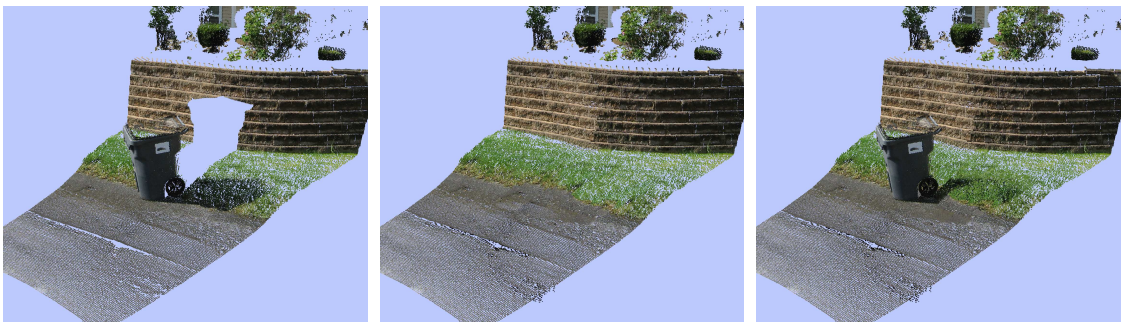
```
Construct the RGBD_xD_y image from the source data
Normalize the RGBD_xD_y image
Inpaint the RGBD_xD_y image:
while Hole pixels remain do
    TargetPatch ← SelectTargetPatch()
    SourcePatch ← FindMatch(TargetPatch)
    Copy SourcePatch into TargetPatch
    Update the hole and hole boundary
end while
Extract the inpainted D_x and D_y inside the hole
Reconstruct the hole depths D* by solving Equations 6.12-6.13
Create the new 3D points at the prescribed depths along the original LiDAR rays
```

6.5 Experiments

In this section, we demonstrate our algorithm on several real-world data sets. The results are convincing, even with complicated backgrounds. The resulting hole completions we obtain appear as if the points had been acquired by the actual LiDAR scanner, making the completions look very natural.

In Figure 6.16, we demonstrate our algorithm on a LiDAR scan of a trashcan in front of a brick wall. The hole left by the removal of the trashcan spans three textures

including concrete, grass, and brick. The algorithm is able to successfully fill this large hole with convincing color and structure.



(a) (b) (c)

Figure 6.16: (a) A LiDAR scan of a trashcan in front of a background consisting of concrete, grass, and a brick wall. (b) The inpainted 3D structure behind the trashcan. (c) A composite of the trashcan with the structure behind it.

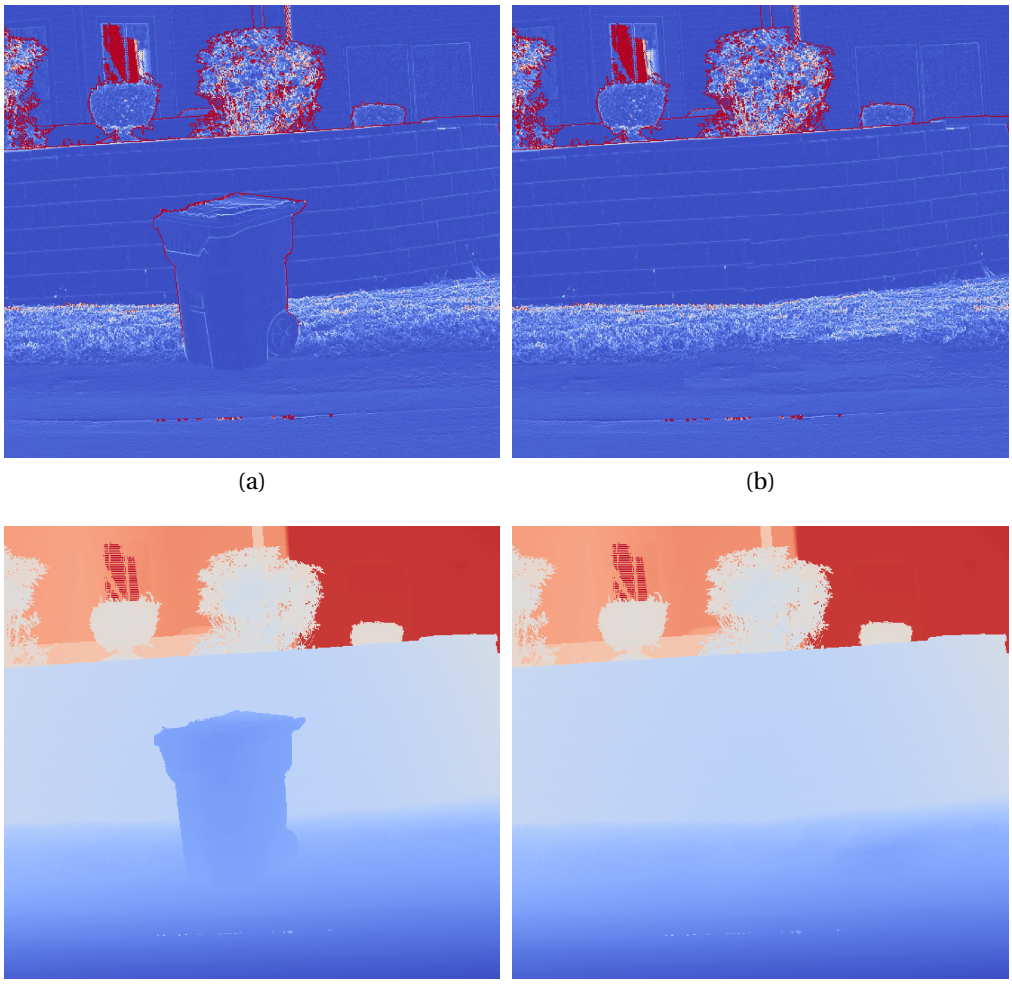
Figure 6.17(a)-(b) shows the depth gradient image before and after inpainting. The inpainted gradient image looks like the gradient field we would expect if the trashcan had not been present in the scene. Figure 6.17(c)-(d) shows the depth image before and after reconstruction from the inpainted depth gradient image, indicating that the results are realistic.

In Figure 6.18, we demonstrate the algorithm with a less regular background. In this LiDAR scan, several electrical boxes are present in a grassy field, with a very complicated background of bushes and trees. Again, we show that the algorithm was able to fill in convincing color and structure in both the smooth ground region as well as the noisy region of trees.

In Figure 6.19, we show a LiDAR scan of a mailbox with a building in the background. The mailbox occludes multiple linear structures, and a harsh shadow is present on the building. Despite these challenges, the algorithm is able to produce a satisfying result.

A summary of the data sets shown throughout this paper, including image size, hole size, and timing of the entire LiDAR inpainting process, is provided in Table 6.1. The experiments were performed on a computer with an Intel Core 2 Duo 3 GHz CPU.

About 75% of the time is spent in the inpainting process, while the remaining

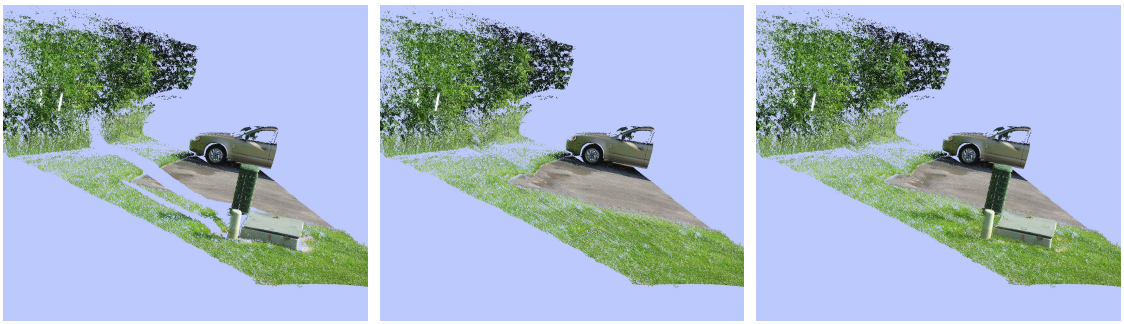


(a) (b)

(c) (d)

Figure 6.17: A demonstration of our depth gradient inpainting approach. (a) The magnitude of the original gradient (blue = low gradient magnitude, red = high gradient magnitude). (b) The magnitude of the inpainted depth gradient. (c) The original depth image (blue = close to the scanner, red = far from the scanner). (d) The depth image reconstructed from the inpainted depth gradient. We note that the structure of the corner between the wall and the ground was successfully preserved.

time is spent reconstructing the depth image from its gradients.



(a) (b) (c)

Figure 6.18: (a) A LiDAR scan of several electric boxes in a grassy field, with a complex background consisting of bushes and trees. (b) The inpainted scene structure behind the electric boxes. (c) A composite of the electric boxes with the inpainted scene behind them.

(a) (b) (c)

Figure 6.19: (a) A LiDAR scan of a mailbox with a building in the background. The LiDAR shadow interrupts multiple linear structures in the background. (b) The inpainted scene structure in the LiDAR shadow. (c) A composite of the mailbox with the background filled behind it.

Table 6.1: A summary of the data sets shown throughout this paper.

Data set	Image size	Hole size (pix)	Total time
Mailbox	459 × 489	30171	1m5s
Electric boxes	688 × 478	45434	2m19s
Trashcan	572 × 517	42734	1m59s
Air conditioners	400 × 496	13709	23s

6.6 Discussion

We presented an algorithm to fill large holes in LiDAR data. We inpaint the data in the depth gradient domain, then reconstruct geometry in the original scanner coordinate system. The experiments demonstrate that the method can plausibly fill large holes, making the data easily viewable from multiple viewpoints without perceptual artifacts.

We note that our LiDAR inpainting technique is more sensitive to poor patch choices than a standard image inpainting problem. For example, in Figure 6.20 we show a case where the image completion would have been deemed perfectly acceptable, but the resulting 3D structure exhibits strange behavior.

(a) (b) (c)

Figure 6.20: A demonstration of sensitivity to error in the inpainting. (a) An image of the inpainted colors of a scene. The red rectangle indicates a region in which a small error has occurred in the inpainting. (b) A zoomed-in version of the red rectangle from (a), showing that several rows of pixels were filled with grass when they should have been filled with brick to correctly continue the wall/ground boundary. (c) A 3D view of the resulting error in the reconstructed LiDAR points.

In this case, the pixels of grass above the yellow line in Figure 6.20b should have actually been filled with brick. In the image alone, the human visual system can hardly perceive this error. However, in the reconstructed 3D scene, the error manifests as a warp in the wall.

Also, as with image inpainting, there are cases where we cannot expect the algorithm to compute a reasonable completion. For example, Figure 6.21 shows a LiDAR scan of a corner of a building, with air conditioning units on the ground. When we attempt to inpaint these air conditioners, we have to construct the intersection of two

walls and the ground that does not appear elsewhere in the scene. The way that these multiple linear structures should be joined inside the hole is ambiguous. This problem might be mitigated by allowing the user to draw guidelines inside the hole to indicate the way linear structures should be merged, as in [146]. Another potential difficulty is having very limited structure to either side of the hole, so patches are repeated multiple times. For example, in Figure 6.21b we can see that there is only a very small piece of the image to the left of the hole at the wall/ground boundary. The results in many of these hard cases could potentially be improved by substituting the greedy inpainting algorithm we used here with a globally optimal technique (e.g. [96]), at the cost of slower performance.

Finally, we could apply a similar technique to achieve a different goal. Rather than filling large holes in LiDAR data by copying the gradients from elsewhere, we could correct sampling inconsistencies in LiDAR scans introduced in places where the laser is nearly parallel to scene surfaces or passes through spotty occlusion like foliage. By redistributing the depth gradient values in particular regions of the scan, we might be able to resample the 3D geometry using the same technique presented here.

(a) (b)

(c) (d)

Figure 6.21: A data set for which we do not expect a good result. There is no information to guide the algorithm to fill the corner that results from the intersect of the two walls and the ground. (a) An image of the LiDAR scan.(b) The region to inpaint. (c) The holes appear to be filled correctly when hidden by the objects. (d) Visible artifacts are present in the resulting scene, including a warped corner and wall.

CHAPTER 7
Post-Candidacy Work

Throughout the work on LiDAR inpainting in our research thus far, we have used a greedy, patch-based method to fill holes in images. While some global methods (notably [96]) have been proposed, they are currently much too slow in practice. A major disadvantage of the greedy method is that if a single bad patch is completed, the entire remaining inpainting result is often rendered unacceptable. In our experiments, we have seen that even a single bad patch can severely corrupt the result of a greedy image completion algorithm, as shown in Figure 7.1. In this example, we see that a patch of grass was incorrectly copied into the brick wall. The algorithm carries on filling patches, but it now believes that grass actually belongs in this region, so many more errors necessarily occur.

(a) (b)

Figure 7.1: An example of the result of a bad patch being copied. (a) In the blue circle, we see that a patch has been copied that does not continue the color and texture of the image into the hole properly. (b) The final result of the inpainting. This single incorrect patch causes a severe visual artifact.

In many cases, the human visual system is capable of viewing the completed image and not noticing that anything is "too wrong." Even in Figure 7.1, at first glance the image does not seem severely distorted. However, when applying these inpainting techniques to inpaint 3D structure (as we have done in Chapter 6), it is much more important to obtain a result that not only is artifact-free, but actually approximates a sensible result in the hole. Because of this, it is extremely important to never allow a

bad patch to be copied.

During our study of the existing literature of inpainting techniques, we have noticed that researchers alway trust that the patch found during their search step of the algorithm is correct. In every algorithm we have encountered, at the end of the patch matching process the "best" patch according to the search metric is automatically used to inpaint the image. We propose a direction for post-candidacy work to develop a search-method independent check of whether the patch that was found to be the "best" is actually a good candidate. This concept is very similar to the idea of our work in Chapter 4.

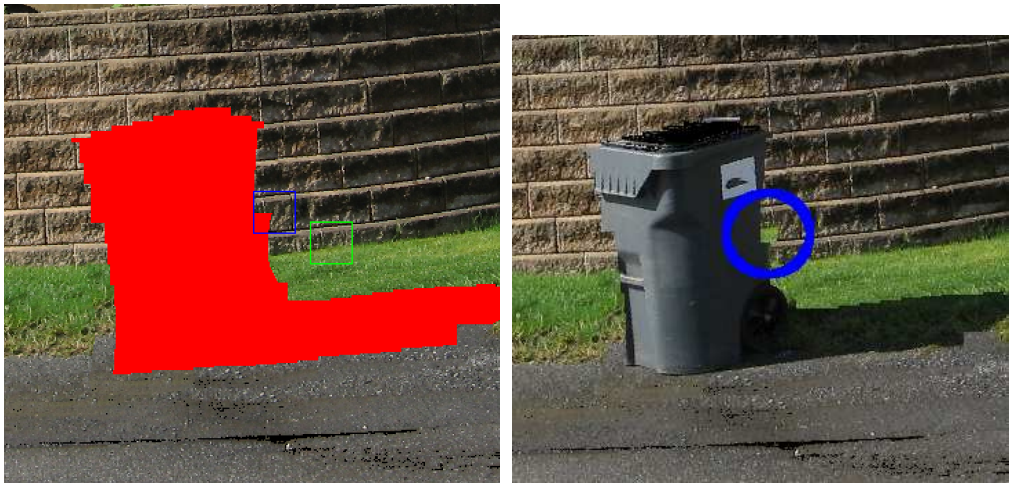
In this work, our goal is to detect when a bad patch is about to be copied, and take action to prevent this from happening. In this chapter we propose initial ideas for detecting bad matches, as well as explore possibilities for handling these errors when they occur. In initial experiments, even with a very small number (5-10 out of hundreds) of bad patch preventions, a dramatic increase in the quality of the final inpainted images was observed.

7.1 Patch Acceptance Criteria

We have developed several patch *acceptance criteria*, and intend to perform experiments to determine which are the most useful. We motivate this problem in Figure 7.2. We see that after a single incorrect patch is copied, the greedy nature of the algorithm propagates the error throughout the remainder of the inpainting, making the result unacceptable.

In Figure 7.3, we show a close-up of the target patch and the best matching source patch (according to the sum-of-squared differences metric in the corresponding valid regions). To a human, it is clear that these patches are obviously not good matches. We see that grass pixels will be copied into the hole where brick pixels should have been copied. Additionally, by inspecting the image it seems that we can find a large set of patches that would result in much better completions. However, due to the very high texture in these regions, the simple SSD metric is not able to differentiate this bad match from the better matches appropriately.

We have manually extracted a source patch that would lead to a much more ac-



(a) (b)

Figure 7.2: An example of patch that matches very poorly, but still ranks the best. (a) After several iterations of inpainting, we have arrived at this situation. The target patch is outlined in blue, and the best source patch is outlined in green. (b) The result of filling the target patch with this incorrect source patch.

(a) (b)

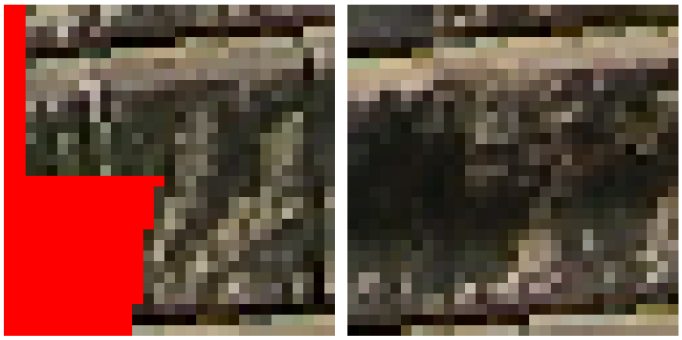
Figure 7.3: A close up of the incorrect matching patches. (a) The target patch from Figure 7.2 (red = hole). (b) The source patch from Figure 7.2. These patches have a SSD score of 7101.1.

ceptable completion, shown in Figure 7.4.

Besides statistical fluctuations (texture) leading to bad matches according to the SSD metric, there is a more systematic failure. Namely, of a very smooth patch is present and contains pixels whose mean are similar to a patch, the problem we described above happens very easily.

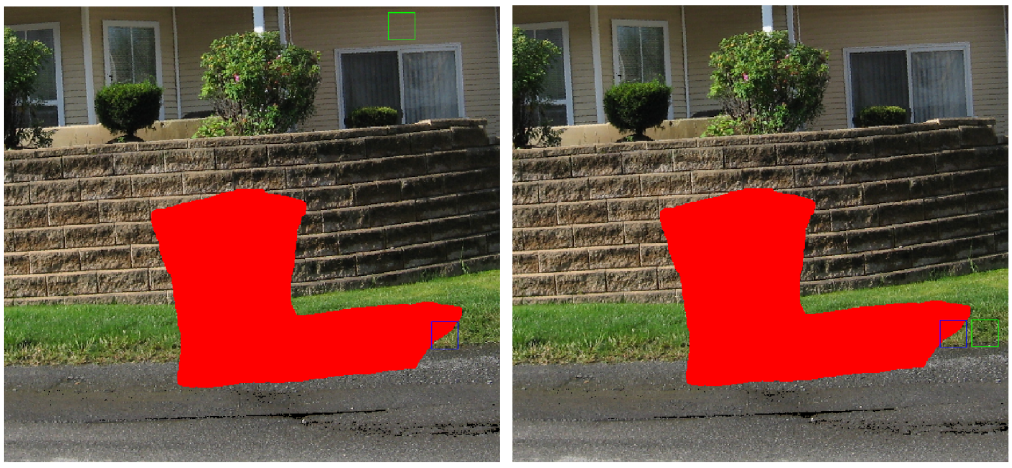
Consider the patches shown in Figure 7.5.

In Figure 7.6 we show close-ups of the patches from Figure 7.5 that would lead to



(a) (b)

Figure 7.4: A better source patch. (a) The same target patch from Figure 7.2 (red = hole). (b) A manually selected source patch which leads to a much more acceptable completion. The SSD between these patches is 8710.0, higher than that of the best, but incorrect, patch pair.



(a) (b)

Figure 7.5: A demonstration of an incorrect match involving a smooth patch. (a) The target patch (blue) and best matching source patch according to the SSD metric (bright green) are shown. The SSD score was 2964.57. (b) A much better choice of source patch is shown in bright green. The SSD score in this case was 5464.61, much higher than in (a).

a much more acceptable completion.

This phenomenon can be explained theoretically by the following. Consider two independent, normally distributed random variables $X = N\left(\mu_x, \sigma_x^2\right)$ and $Y = N\left(\mu_y, \sigma_y^2\right)$. The difference of these variables, $X - Y$ is distributed as $Z = N\left(\mu_x - \mu_y, \sigma_x^2 + \sigma_y^2\right)$. That is, the variance of the resulting variable is the sum of the variances of the original vari-

(a) (b) (c)

Figure 7.6: A close up of a very bad visual match that has a low SSD. (a) The target patch from Figure 7.5 (red = hole). (b) The best patch according to the SSD metric (a score of 2964.57) 7.6c A manually selected source patch, which is visually much more acceptable. The SSD between this patch and the target patch is 5464.61.

ables. Now consider three image patches, A, B, and C. A and B are from the same area of an image, so they have the same texture. We can describe a pixel in this region to have mean u_T and variance σ_T^2. Now if we subtract two pixels from this region, we obtain an error distributed as $N(0, 2\sigma_T^2)$. Now consider patch C to consist of pixels all with value c. The difference between a pixel from A and C is distributed as $N\left(u_T - c, \sigma_T^2\right)$. If c is near u_T, then this difference is statistically smaller (variance σ_T^2 versus $2\sigma_T^2$) than the difference of two pixels from the same textured region! This surprising and counter-intuitive result wreaks havoc on standard patch comparison functions similar to SSD.

In our experiments during the work in this thesis, this type of issue occurs often enough that standard greedy patch-based inpainting is highly prone to failure. However, we have noticed that a bad match only happens with a very small number of target patches (5 - 10) on large holes (200-300 target patches) in highly textured images. In post-candidacy work, we will address this problem by introducing methods to prevent this behavior.

We note that this phenomenon was previously observed in [79]. The authors mention in passing that the variance difference in the patches can be used to aid this problem, but do not prescribe a way to build this idea into the patch matching cost function. Our experiments thus far have shown that performing computations more

intensive than a SSD calculation at every source patch is too computationally expensive, which has motivated our idea of using acceptance criteria only on the several best matches, which we detail in the next section.

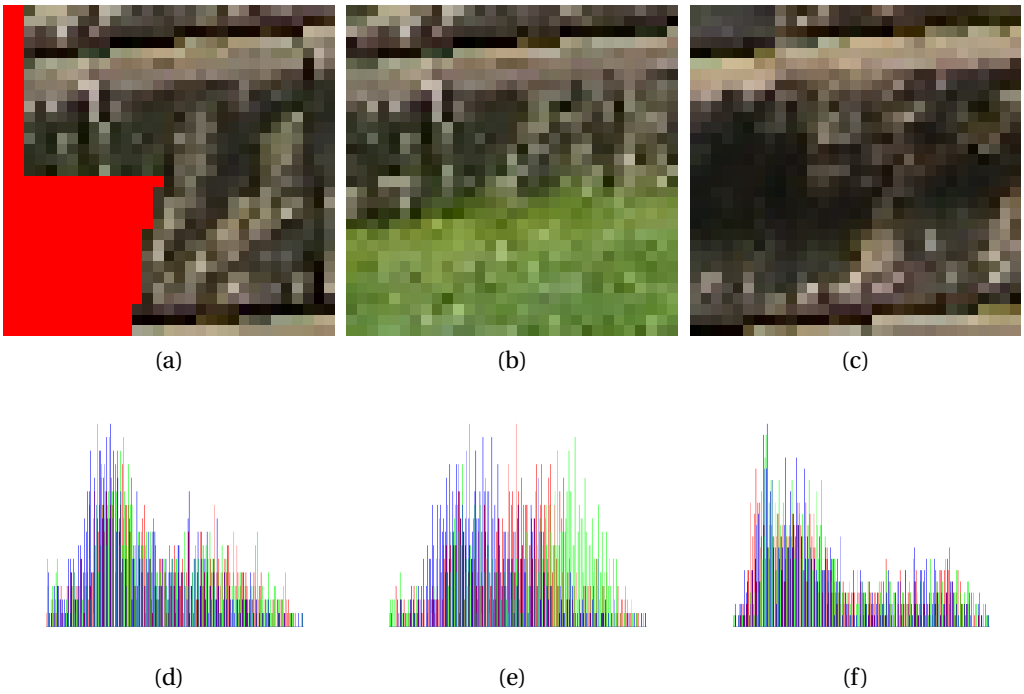
As we noted in the previous section, direct pixel-wise comparisons of patches are very unreliable estimates of the similarity of two patches. Unfortunately, more descriptive metrics are significantly more computationally intensive, and therefore cannot be computed at every source patch. Our main question is, once a small subset of promising patches has been identified using a fast comparison algorithm (such as SSD), can we construct a more complex algorithm that best predicts which of these patches is likely to produce the best completion?

In this section, we propose a modified patch search technique. First, we searches for a small set of candidate source patches using the standard SSD metric. These top patches are then strenuously evaluated using several criteria. If any of these criteria are not met, then the user must take action to correct the bad match.

A promising direction we have identified for this strenuous comparison of patches is to compare the histograms of the target patch and candidate source patches. Consider the histograms in Figure 7.7. Figure 7.7a, the target patch histogram from our previous example, is quite dissimilar from Figure 7.7b, the best patch according to SSD score. However, Figure 7.7c, the histogram of the patch selected by the user, is a much better match.

In the smooth patch example, the histograms also prove very discriminative, as shown in Figure 7.8.

While directly searching for source patches using the histograms is much too slow, it is also not descriptive enough on its own, as all spatial information within the patch is discarded. That is, a patch that is black on the left and white on the right has exactly the same histogram as a patch that is white on the left and black on the right. However, since our first step, the SSD comparison, did use this spatial information, the histogram is now a good way to differentiate bad matches from good matches. Many techniques to compare histograms have been proposed, such as a direct bin-to-bin comparison, the EarthMovers distance [131] , the histogram intersection score [147], and the diffusion distance [108]. We will investigate how those and other metrics help



(a) (b) (c)

(d) (e) (f)

Figure 7.7: Histograms of the target, best source, and manually selected source patches. (a) The histogram of the valid region of the target patch from Figure 7.7b. (c) The histogram of the valid region of the best source patch according to the SSD metric. (c) The histogram of the valid region of the a better patch selected manually.

us to identify a bad match.

As further enhancement, rather than naively collect pixels into uniform histogram bins, we can use color quantization quantization techniques to make “smarter” histograms which can then also be compared in the proposed regions. This could be done by using image clustering and smoothing algorithms such as [76, 157].

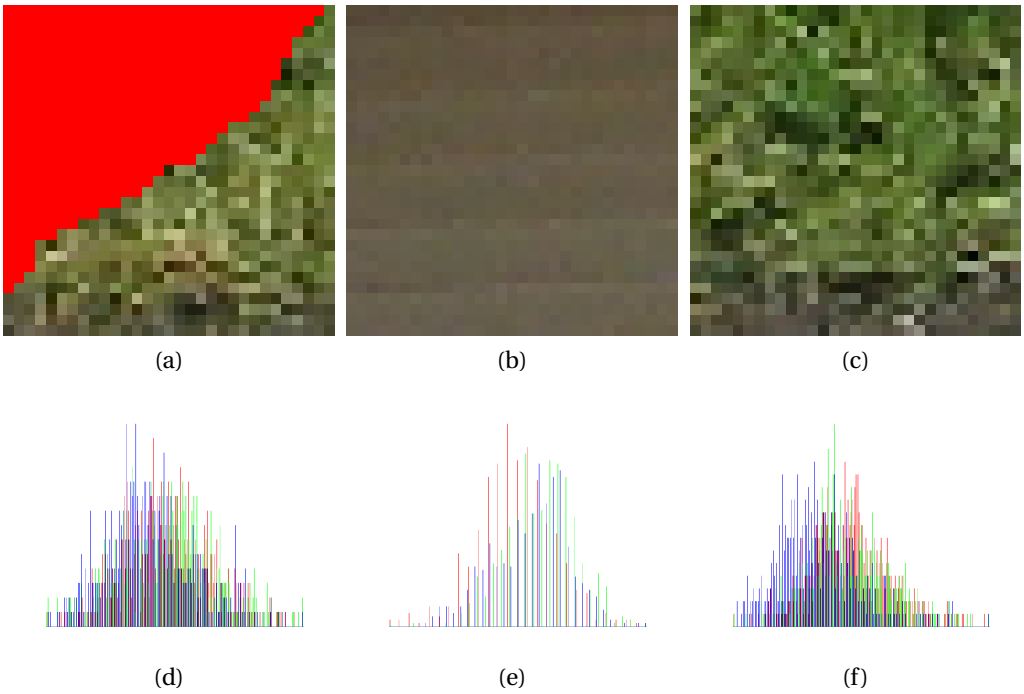


Figure 7.8: Histograms of the target, best source, and manually selected source patches in the smooth patch example. (d) The histogram of the valid region of the target patch from Figure 7.8. (e) The histogram of the valid region of the best source patch according to the SSD metric. (f) The histogram of the valid region of the a better patch selected manually.

7.2 Handling Detected Bad Patches

If it is not, the user should somehow choose a better patch. We detail proposed techniques for both identifying bad patches as well as different methods of override patch selection by the user in the remainder of this chapter. It is our opinion that a user would much rather have a non-fully-automatic algorithm that completes images successfully a larger portion of the time than a fully-automatic algorithm that sometimes fails and does not provide any recourse. In fact, in many real-world situations where professional artists and editors are using software to solve problems such as this (as described in [127]), they strongly prefer to have significant control of the result, rather than use a "black-box" algorithm.

Our goal is to alert the user of possible bad patch matches. Therefore, we can apply several acceptance tests simultaneously. We then perform a boolean AND operation on all of acceptance tests so that if any one of them fails, it is interpreted as an

indication that something could be wrong.

The acceptance test framework also allows us to handle special cases. For example, if the hole is very small (for example, covers less than 10% of the patch region), then it should be accepted automatically, because any statistical comparison will be much too sensitive when only a few pixels are involved. This situation frequently occurs near the end of an inpainting operation.

When the any of our acceptance tests fail, we take action. We are developing an interface to allow the user to do several things. First, the best source patch that had been selected is presented to the user, along with the tentative result of the patch inpainting, so they can visually determine if it is indeed a bad selection. If it is not (the acceptance tests produced a false negative), then the user can simply indicate that this patch is ok and allow the algorithm to proceed. If the patch is *not* visually acceptable, the user is then presented with several hundred of the top patches which can be quickly scanned to find an acceptable patch. Along with this work, we will investigate clustering the top several patches, so that the user can interactively narrow down the search to finding a good patch. In our experiments, there are generally hundreds of similarly bad patches grouped together (with similar SSD scores), and these repeated bad patches only serve as clutter. Finally, if a good patch is still not able to be located, the user can manually position a patch over any part of the source region to manually select which patch should be copied. This allows for a reasonable completion even in extremely hard data sets that would typically fail outright. Additionally, this process could be automated by identifying many, rather than just one, potential target patches at each iteration. Then, we fill the first one that passes all of our acceptance criteria. The resulting situation is shown in Figure 7.9.

If none of these candidate target patches has an acceptable match according to our criteria, we then ask the user to select a patch identically as before. Effectively, this allows us to defer filling difficult target patches until later in the algorithm. By filling these difficult patches only when no “easy” patches remain, we have reduced the number of iterations that the algorithm is allowed to propagate an error, greatly improving the inpainting result.

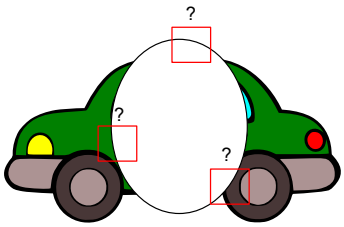


Figure 7.9: An sketch of inspecting multiple candidate target patches simultaneously.

BIBLIOGRAPHY

[1] A. Abdelhafiz, B. Riedel, and W. Niemeier. Towards a 3D true colored space by the fusion of laser scanner point cloud and digital photos. In *Proceedings of the ISPRS Working Group V/4 Workshop*, 2005.

[2] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. S. SLIC Superpixels. Technical Report June, Écolo Polytechnique Fédrale de Lausanne, 2010.

[3] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building Rome in a day. In *IEEE International Conference on Computer Vision*, volume 54, pages 72–79, September 2009.

[4] S. Agarwal, Y. Furukawa, N. Snavely, B. Curless, and S. M. Seitz. Reconstructing Rome. *Computer*, 43(6):40–47, 2010.

[5] A. Agathos and R. B. Fisher. Colour texture fusion of multiple range images. In *International Conference on 3-D Digital Imaging and Modeling*, pages 139–146, 2003.

[6] K. Alahari, P. Kohli, and P. H. S. Torr. Reduce, reuse & recycle: Efficiently solving multi-label MRFs. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008*, volume 1, pages 1–8. Ieee, June 2008.

[7] Y. Alshawabkeh, N. Haala, and D. Fritsch. Range Image Segmentation Using the Numerical Description of the Mean Curvature Values. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37(B5): 533–538, 2008.

[8] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng. Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 169–176 v. 2. Ieee, 2005.

[9] A. Bab-hadiashar and N. Gheissari. Range image segmentation using surface selection criterion. *IEEE Transactions on Image Processing*, 15(7):2006–2018, 2006.

[10] A. Bab-hadiashar and D. Suter. Robust Range Segmentation. In *International Conference on Pattern Recognition*, pages 969–971, 1998.

[11] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch : A Randomized Correspondence Algorithm for Structural Image Editing. *ACM Transactions on Graphics*, 28(3):24:1–24:11, 2009.

[12] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein. The Generalized PatchMatch Correspondence Algorithm. In *European Conference on Computer Vision*, pages 29–43, 2010.

[13] J. Becker, C. Stewart, and R. J. Radke. LiDAR inpainting from a single image. In *IEEE International Conference on Computer Vision Workshops*, pages 1441–1448. Ieee, September 2009.

[14] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[15] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Conference on Computer Graphics and Interactive Techniques*, pages 417–424, New York, New York, USA, 2000. ACM Press.

[16] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. Simultaneous Structure and Texture Image Inpainting. *Transactions on Image Processing*, 12(8):882–889, 2003.

[17] P. Besl and H. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[18] P. Bhat, C. L. Zitnick, M. Cohen, and B. Curless. GradientShop : A Gradient-Domain Optimization Framework for Image and Video Filtering. *ACM Transactions on Graphics*, 29(2), 2010.

[19] A. V. Bhavsar and A. Rajagopalan. Inpainting Large Missing Regions in Range Images. In *2010 20th International Conference on Pattern Recognition*, pages 3464–3467. Ieee, August 2010.

[20] J. M. Biosca and J. L. Lerma. Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. *Journal of Photogrammetry and Remote Sensing*, 63(1):84 – 98, 2008.

[21] R. Bornard, E. Lecan, L. Laborelli, and J.-H. Chenot. Missing Data Correction in Still Images and Image Sequences. In *ACM International Conference on Multimedia*, number December, pages 355–361, 2002.

[22] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23 (11):1222–1239, 2001.

[23] Y. Boykov and G. Funka-Lea. Graph Cuts and Efficient N-D Image Segmentation. *International Journal of Computer Vision*, 70(2):109–131, November 2006.

[24] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–37, September 2004.

[25] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *IEEE Computer Society International Conference on Computer Vision*, number July, pages 105–112 v.1, 2001.

[26] M. J. Bravo and H. Farid. Search for a category target in clutter. *Perception*, 33 (6):643–652, 2004.

[27] N. Brusco, M. Andreetto, A. Giorgi, and G. M. Cortelazzo. 3D registration by textured spin-images. In *International Conference on 3-D Digital Imaging and Modeling*, pages 262–269, 2005.

[28] R. Campbell and P. Flynn. Eigenshapes for 3D object recognition in range data. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 505–510, 1999.

[29] O. Carmichael and M. Hebert. 3D cueing: a data filter for object recognition. In *IEEE International Conference on Robotics and Automation*, pages 944–950, 1999.

[30] O. Carmichael, D. Huber, and M. Hebert. Large data sets and confusing scenes in 3-D surface matching and recognition. In *International Conference on 3-D Digital Imaging and Modeling*, pages 358–367. IEEE Comput. Soc, 1999.

[31] T. Chan and J. Shen. Variational Restoration Of Nonflat Image Features: Models And Algorithms. *SIAM Journal on Applied Mathematics*, 61:1338–1361, 2000.

[32] T. Chan and J. Shen. Non-Texture Inpainting by Curvature-Driven Diffusions (CDD). *Journal of Visual Communication and Image Representation*, 12(4):436–449, 2001.

[33] Y. Chen and G. Medioni. Object Modeling by Registration of Multiple Range Images. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 2724–2729, 1991.

[34] T. Chevalier, P. Andersson, C. Grönwall, and G. Tolt. Methods for ground target detection and recognition in 3-D laser data. Technical Report December, 2006.

[35] L. D. Cohen. On Active Contour Models and Balloons. *CVGIP: Image Understanding*, 53(2):211–218, 1991.

[36] A. Criminisi. Object Removal by Exemplar-Based Inpainting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 721–728, 2003.

[37] A. Criminisi. Region Filling and Object Removal by Exemplar-Based Image Inpainting. *IEEE Transactions on Image Processing*, 13(9):1200–1212, 2004.

[38] Y. Cui, S. Schuon, D. Chan, S. Thrun, and C. Theobalt. 3D Shape Scanning with a Time-of-Flight Camera. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1173–1180, 2010.

[39] C. Dal Mutto, P. Zanuttigh, and G. M. Cortelazzo. Scene Segmentation by Color and Depth Information and its Applications. Technical report, 2010.

[40] P. Dias, V. Sequeira, F. Vaz, and J. Goncalves. Registration and Fusion of Intensity and Range Data for 3D Modelling of Real World Scenes. In *International Conference on 3-D Digital Imaging and Modeling*, pages 418–425, 2003.

[41] D. Doria. A Synthetic LiDAR Scanner for VTK. Technical report, 2009. URL `http://www.insight-journal.org/browse/publication/695`.

[42] D. Doria. A Synthetic LiDAR Scanner for VTK. *Kitware Source Magazine*, 2010.

[43] D. Doria. A Greedy Patch-based Image Inpainting Framework. *Kitware Source Magazine*, 2011.

[44] D. Doria. Poisson Editing in ITK. *Kitware Source Magazine*, 2011.

[45] D. L. Doria. A Point Set Processing Toolkit for VTK. Technical report, 2009. URL `http://www.midasjournal.org/browse/publication/708`.

[46] D. L. Doria. Region Growing Using a Criterion on the Region Boundary. Technical report, 2009. URL `http://www.insight-journal.org/browse/publication/688`.

[47] D. L. Doria. A Conditional Mesh Front Iterator for VTK. Technical report, 2010. URL `http://www.insight-journal.org/browse/publication/725`.

[48] D. L. Doria. A K-Means++ Clustering Implementation for VTK. Technical report, 2010. URL `http://www.insight-journal.org/browse/publication/766`.

[49] D. L. Doria. A Mean Shift Clustering Implementation for VTK. Technical report, 2010. URL `http://www.insight-journal.org/browse/publication/765`.

[50] D. L. Doria. Point Set Surface Reconstruction for VTK. Technical report, 2010. URL `http://www.midasjournal.org/browse/publication/713`.

[51] D. L. Doria. RANSAC Plane Fitting for VTK. Technical report, 2010. URL `http://www.midasjournal.org/browse/publication/709`.

[52] D. L. Doria. Stratified Mesh Sampling for VTK. Technical report, 2010. URL `http://www.midasjournal.org/browse/publication/719`.

[53] D. L. Doria. Graph Cuts Based Super Pixel Segmentation for VTK. Technical report, 2010. URL `http://www.midasjournal.org/browse/publication/720`.

[54] D. L. Doria. Clustering Segmentation for VTK. Technical report, 2011. URL `http://www.midasjournal.org/browse/publication/833`.

[55] D. L. Doria. Interactive Correspondence Selection. Technical report, 2011. URL http://www.insight-journal.org/browse/publication/836.

[56] D. L. Doria. Criminisi Inpainting. Technical report, 2011. URL http://www.insight-journal.org/browse/publication/787.

[57] D. L. Doria. Loopy Belief Propagation on MRFs in ITK. Technical report, 2011. URL http://www.insight-journal.org/browse/publication/789.

[58] D. L. Doria. Small Hole Filling in ITK. Technical report, 2011. URL http://www.insight-journal.org/browse/publication/835.

[59] D. L. Doria and D. Borrmann. Hough Transform Plane Detector. Technical report, 2011. URL http://www.midasjournal.org/browse/publication/820.

[60] D. L. Doria and S. Chen. Interactive Image Graph Cut Segmentation. Technical report, 2010. URL http://www.insight-journal.org/browse/publication/777.

[61] D. L. Doria and A. Gelas. Poisson Surface Reconstruction for VTK. Technical report, 2010. URL http://www.midasjournal.org/browse/publication/718.

[62] D. L. Doria and R. J. Radke. Consistency and Confidence : A Dual Metric for Verifying 3D Object Detections in Multiple LiDAR Scans. In *International Conference on 3-D Digital Imaging and Modeling*, pages 1481–1488, 2009.

[63] I. Drori, D. Cohen-Or, and H. Yeshurun. Fragment-based image completion. *ACM Transactions on Graphics*, 22(3):303–312, July 2003.

[64] A. A. Efros and W. T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *Conference on Computer Graphics and Interactive Techniques*, pages 341–346, 2001.

[65] A. A. Efros and T. K. Leung. Texture Synthesis by Non-parametric Sampling. In *Proceedings of the 7th IEEE International Conference on Computer Vision, 1999*, number September, pages 1033–1038 v.2, 1999.

[66] I. Essa and N. Kwatra. Texture Optimization for Example-based Synthesis. *ACM Transactions on Graphics*, 24(3):795–802, 2005.

[67] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.

[68] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[69] A. W. Fitzgibbon. Robust Registration of 2D and 3D Point Sets. *Image and Vision Computing*, 21(13-14):1145–1153, 2001.

[70] D. Ford, L and Fulkerson. *Flows in networks*. Princeton University Press.

[71] G. Forlani, C. Nardinocchi, M. Scaioni, and P. Zingaretti. Complete classification of raw LIDAR data and 3D reconstruction of buildings. *Pattern Analysis and Applications*, 8(4):357–374, January 2006.

[72] A. Formella and C. Gill. Ray Tracing : A Quantitative Analysis and a New Practical Algorithm. *The Visual Computer*, 11(9):465–476, 1995.

[73] A. Frome, D. Huber, R. Kolluri, and T. B. Recognizing Objects in Range Data Using Regional Point Descriptors. In *European Conference on Computer Vision*, volume 1, pages 224–237, 2004.

[74] C. Frueh, R. Sammon, and A. Zakhor. Automated Texture Mapping of 3D City Models With Oblique Aerial Imagery. In *Proceedings of the 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004*, pages 396–403, 2004.

[75] C. Früh and A. Zakhor. Data Processing Algorithms for Generating Textured 3D Building Façade Meshes From Laser Scans and Camera Images. *International Journal of Computer Vision*, 61(2):159–184, 2005.

[76] B. Fulkerson, A. Vedaldi, and S. Soatto. Class Segmentation and Object Localization with Superpixel Neighborhoods. In *IEEE International Conference on Computer Vision*, pages 670–677, 2009.

[77] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, August 2010.

[78] A. Golovinskiy and T. Funkhouser. Min-cut based segmentation of point clouds. In *IEEE International Conference on Computer Vision*, pages 39–46. Ieee, September 2009.

[79] P. Goyal and S. Diwakar. Fast and Enhanced Algorithm for Exemplar Based Image Inpainting. In *Pacific-Rim Symposium on Image and Video Technology*, pages 325–330, 2010.

[80] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*, volume 23. Cambridge Univ Press, 2 edition, 2004. ISBN 0-521-45051-8.

[81] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, 1987.

[82] B. Horn. Extended Gaussian Images. *Proceedings of the IEEE*, 72(12):1671–1686, 1984.

[83] D. Huber. Fully automatic registration of multiple 3D data sets. *Image and Vision Computing*, 21(7):637–650, July 2003.

[84] D. Huber. Parts-based 3D object classification. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages II–82, 2004.

[85] A. Jagannathan, E. L. Miller, and S. Member. Three-Dimensional Surface Mesh Segmentation Using Curvedness-Based Region Growing Approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(12):2195–2204, 2007.

[86] A. Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997.

[87] A. E. Johnson and M. Hebert. Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433–449, 1999.

[88] D. B. Judd. Hue Saturation and Lightness of Surface Colors with Chromatic Illumination. *JOSA*, 30(1):2–32, 1940.

[89] E. Kalogerakis, A. Hertzmann, and K. Singh. Learning 3D mesh segmentation and labeling. *ACM Transactions on Graphics*, 29(4):102:1–102:12, July 2010.

[90] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[91] N. Kawai, T. Sato, and N. Yokoya. Image Inpainting Considering Brightness Change and Spatial Locality of Textures and Its Evaluation. *Computer Science*, 5414:271–282, 2009.

[92] K. Kevin, R. Koch, and C.-a.-u. Kiel. Perspectively Invariant Normal Features. In *International Converence on Computer Vision*, 2007.

[93] K. Klasing and D. Wollherr. Realtime segmentation of range data using continuous nearest neighbors. In *IEEE International Conference on Robotics and Automation, 2009*, pages 2431–2436, 2009.

[94] K. Klasing, D. Wollherr, and M. Buss. A clustering method for efficient segmentation of 3D laser data. *2008 IEEE International Conference on Robotics and Automation*, pages 4043–4048, May 2008.

[95] V. Kolmogorov and C. Rother. Comparison of Energy Minimization Algorithms for Highly Connected Graphs. *Computer Vision-ECCV 2006*, 3952:1–15, 2006.

[96] N. Komodakis. Image Completion Using Global Optimization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 442–452. Ieee, 2006.

[97] N. Komodakis and G. Tziritas. Image completion using efficient belief propagation via priority scheduling and dynamic pruning. *IEEE Transactions on Image Processing*, 16(11):2649–2661, November 2007.

[98] M. Kortgen, G.-J. Park, M. Novotni, and R. Klein. 3D Shape Matching with 3D Shape Contexts. In *Central European Seminar on Computer Graphics*, 2003.

[99] T. Kurita. An Efficient Agglomerative Clustering Algorithm for Region Growing. In *MVA 1994 IAPR Workshop on Machine Vision Applications*, pages 210–213, 1994.

[100] R. Lange. *Time-of-flight distance measurement with with custom custom solid-state solid-state image sensors in CMOS / CCD-technology Robert 3D Time-of-Flight Distance Measurement with Custom Solid-State Image Sensors in CMOS/CCD-technology*. PhD thesis, Universitätsbibliothek.

[101] I. Lee and T. Schenk. Perceptual organization of 3D surface points. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34(3A):193–198, 2002.

[102] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless Image Stitching in the Gradient Domain. *Computer Science*, 3024:377–389, 2004.

[103] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, and S. J. Dickinson. TurboPixels : Fast Superpixels Using Geometric Flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2290–2297, 2009.

[104] M. Levoy. QSplat : A Multiresolution Point Rendering System for Large Meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, 2000.

[105] M. Levoy, S. Rusinkiewicz, M. Ginzton, J. Ginsberg, K. Pulli, D. Koller, S. Anderson, J. Shade, B. Curless, L. Pereira, J. Davis, and D. Fulk. The Digital Michelangelo Project : 3D Scanning of Large Statues. In *ACM SIGGRAPH*, 2000.

[106] Y. Li, J. Sun, C.-k. Tang, and H.-y. Shum. Lazy Snapping. *ACM Transactions on Graphics*, 23(3):303–308, 2004.

[107] B. Liefers and S. R. T. Tan. Gradient Space Manipulation and Shadow Removal. Technical report, Utrecht University, 2011.

[108] H. Ling and K. Okada. Diffusion Distance for Histogram Comparison. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 246–253. Ieee, 2006.

[109] L. Linsen. Point Cloud Representation. Technical report, 2001.

[110] L. Liu, I. Stamos, N. York, G. Yu, and G. Wolberg. Multiview Geometry for Texture Mapping 2D Images Onto 3D Range Data. *Computer Vision and Pattern Recognition*, 2:2293–2300, 2006.

[111] R. Liu and H. Zhang. Segmentation of 3D Meshes through Spectral Clustering. In *Pacific Conference on Computer Graphics and Applications*, pages 298–305, 2004.

[112] D. G. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the 7th IEEE International Conference on Computer Vision, 1999*, volume 2, pages 1150–1157 v.2, 1999.

[113] D. J. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, 1990.

[114] A. Makadia, A. Patterson, and K. Daniilidis. Fully Automatic Registration of 3D Point Clouds. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1297–1304. Ieee, 2006.

[115] A. Mastin, J. Kepner, and J. Fisher III. Automatic Registration of LIDAR and Optical Images of Urban Scenes. In *IEEE Conference on Computer Vision and Pattern Recognition, 2009*, pages 2639–2646, 2009.

[116] B. Matei, Y. Shan, S. Member, R. Kumar, D. Huber, and M. Hebert. Rapid Object Indexing Using Locality Sensitive Hashing and Joint 3D-Signature Space Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7): 1111–1126, 2006.

[117] A. S. Mian, M. Bennamoun, and R. Owens. Three-Dimensional Model-Based Object Recognition and Segmentation in Cluttered Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1584–1601, 2006.

[118] N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas. Registration of Point Cloud Data from a Geometric Optimization Perspective. In *Eurographics Symposium on Geomtetry Processing*, 2004.

[119] D. Nehab and P. Shilane. Stratified Point Sampling of 3D Models. In *Eurographics Symposium on Point-Based Graphics*, pages 49–56, 2004.

[120] M. M. Oliveira, B. Bowen, R. McKenna, and Y.-S. Chang. Fast Digital Image Inpainting. In *International Conference on Visualization, Imaging and Image Processing*, number Viip, 2001.

[121] N. R. Pal and S. K. Pal. A Review on Image Segmentation Techniques. *Pattern Recognition*, 26(9):1277–1294, 1993.

[122] S. Park, X. Guo, H. Shin, and H. Qin. Shape and Appearance Repair for Incomplete Point Surfaces. In *International Conference on Computer Vision*, pages 1260–1267, 2005.

[123] A. Patterson, P. Mordohai, and K. Daniilidis. Object Detection from Large-Scale 3D Datasets using Bottom-up and Top-down Descriptors. *European Conference on Computer Vision*, pages 553–566, 2008.

[124] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003.

[125] J. M. Phillips and C. Tomasi. Outlier Robust ICP for Minimizing Fractional RMSD. In *International Conference on 3-D Digital Imaging and Modeling*, pages 427–434, 2007.

[126] K. Pulli and M. Pietikainen. Range Image Segmentation Based on Decomposition of Surface Normals. In *Scandinavian Conference on Image Analysis*, 1993.

[127] R. J. Radke. *Computer Vision For Visual Effects*. Cambridge University Press, 2012.

[128] P. S. Ramsing and S. Ruikar. Improved Algorithm for Exemplar Based Image Inpainting. In *Proceedings of the International Conference on Information Science and Applications ICISA*, number February, pages 254–257, 2010.

[129] C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno. A low cost 3D scanner based on structured light. *Computer Graphics Forum*, 20(3):299–308, September 2001.

[130] C. Rother, V. Kolmogorov, and A. Blake. GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. In *ACM SIGGRAPH*, volume 23, pages 309–314, 2004.

[131] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[132] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *International Conference on 3-D Digital Imaging and Modeling*, pages 145–152. IEEE Comput. Soc, 2001.

[133] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3D Point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, November 2008.

[134] R. B. Rusu, N. Blodow, and M. Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009.

[135] S. Salamanca, P. Merch, A. Adan, C. Cerrada, and E. Pérez. Filling Holes in 3D Meshes using Image Restoration Algorithms. In *International Symposium on 3D Data Processing, Visualization, and Transmission*, 2008.

[136] H. Samet. Implementing ray tracing with octrees and neighbor finding. *Computers & Graphics*, 13(4):445–460, 1989.

[137] S. M. Seitz, J. Diebel, D. Scharstein, and R. Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 519–528, 2001.

[138] A. Sharf, M. Alexa, and D. Cohen-Or. Context-based surface completion. *Association for Computing Machinery Transactions on Graphics*, 23(3):878–887, August 2004.

[139] J. Shen, X. Jin, C. Zhou, and C. Wang. Gradient based image completion by solving the Poisson equation. *Computers & Graphics*, 31(1):119–126, January 2007.

[140] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[141] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing Visual Data Using Bidirectional Similarity. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008*, number ii, 2008.

[142] E. R. Smith, B. J. King, C. V. Stewart, and R. J. Radke. Registration of Combined Range-Intensity Scans : Initialization Through Verification. *Computer Vision and Image Understanding*, 110(2):226–244, 2007.

[143] E. R. Smith, R. J. Radke, and C. V. Stewart. Physical Scale Keypoints: Matching and Registration for Combined Intensity/Range Images. *International Journal of Computer Vision*, 97(1):2–17, June 2011.

[144] A. R. Specht, A. D. Sappa, and M. Devy. Edge registration versus triangular mesh registration, a comparative study. *Signal Processing: Image Communication*, 20 (9-10):853–868, October 2005.

[145] P. Stavrou, P. Mavridis, G. Papaioannou, and G. Passalis. 3D Object Repair Using 2D Algorithms. *International Conference on Computational Science*, 3992:271–278, 2006.

[146] J. Sun and J. Jia. Image Completion with Structure Propagation. *ACM Transactions on Graphics*, 24(3):861–868, 2005.

[147] M. Swain and D. Ballard. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[148] A. Telea. An image inpainting technique based on nonparametric kernel estimation. *Journal of Graphics Tools*, 9(1):23–24, May 2004.

[149] E. Trucco and R. B. Fisher. Experiments in Curvature-Based Segmentation of Range Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17 (2):177–182, 1995.

[150] A. N. Vasile and R. M. Marino. Pose-Independent Automatic Target Detection and Recognition Using 3D Laser Radar Imagery. *Lincoln Laboratory Journal*, 15 (1):61–78, 2005.

[151] J. Verdera, V. Caselles, M. Bertalmio, and G. Sapiro. Inpainting surface holes. In *Proceedings of the 2003 International Conference on Image Processing*, pages II–903–6. Ieee, 2003.

[152] J. Wang and M. Oliveira. Filling holes on locally smooth surfaces reconstructed from point clouds. *Image and Vision Computing*, 25(1):103–113, January 2007.

[153] J. Wang and M. Oliveira. A hole-filling strategy for reconstruction of smooth surfaces in range images. In *Brazilian Symposium on Computer Graphics and Image Processing*, volume 16th, pages 11–18, 2003.

[154] L. Wang, J. Hailin, R. Yang, and M. Gong. Stereoscopic Inpainting : Joint Color and Depth Completion from Stereo Images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[155] Y. Wexler, E. Shechtman, and M. Irani. Space-Time Completion of Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):463–476, 2007.

[156] C. Wu, B. Clipp, X. Li, J.-m. Frahm, and M. Pollefeys. 3D Model Matching with Viewpoint-Invariant Patches (VIP). In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[157] L. Xu, C. Lu, Y. Xu, and J. Jia. Image Smoothing via L0 Gradient Minimization. *ACM Transactions on Graphics*, 30(6):1–12, 2011.

[158] N. Xu, R. Bansal, and N. Ahuja. Object Segmentation Using Graph Cuts Based Active Contours. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 46–53 v.2, 2003.

[159] Z. Xu and J. Sun. Image inpainting by patch propagation using patch sparsity. *IEEE Transactions on Image Processing*, 19(5):1153–1165, May 2010.

[160] M. Y. Yang. Plane Detection in Point Cloud Data. Technical Report 1, University of Bonn, 2010.

[161] T. C. Yapo, C. V. Stewart, and R. J. Radke. A Probabilistic Representation of LiDAR Range Data for Efficient 3D Object Detection. In *Computer Vision and Pattern Recognition Workshops*, 2008.

[162] G. Yu, M. Grossberg, G. Wolberg, and I. Stamos. Think Globally , Cluster Locally : A Unified Framework for Range Segmentation. In *International Symposium on 3D Data Processing, Visualization and Transmission*, 2008.

[163] Y. Zeng, W. Chen, D. Samaras, and Q. Peng. Topology Cuts : A Novel Min-Cut / Max-Flow Algorithm for Topology Preserving Segmentation in N-D Images. *Computer Vision and Image Understanding*, 112(1):81–90, 2008.

[164] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994.