LIDAR DATA MANIPULATION: OBJECT DETECTION VERIFICATION, FULL-OBJECT SEGMENTATION IN LIDAR, AND LARGE HOLE FILLING IN LIDAR

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, COMPUTER, AND SYSTEMS ENGINEERING

Approved by the
Examining Committee:

Richard 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 which uses time-of-flight calculations on laser pulses to obtain 3D point samples from a 3D scene. 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 texture information. In this thesis we investigate several interesting uses of such data sets.

First, 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.

Second, we propose a two step technique to interactively segment objects from the background in LiDAR scans. Rooted in image graph-cut techniques, this method uses the confidence of the depth of the LiDAR points where it is unambiguous to perform a coarse conservative segmentation which is used to initialize a traditional color segmentation. The resulting segmentation demonstrates the positive qualities of both types of segmentations.

Third, we study how to fill a large hole in LiDAR data. These holes are present in every scan, as the laser cannot see through opaque objects, resuling in “shadows” being cast behind every object. There are two uses for filling these holes. The first is to make a scan viewable from a position other than the original acquisiton position. The second is to fill the the resulting hole in a scene after removing an object selected with our LiDAR object segmentation algorithm.

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 a “forward looking” extension which considers the cost of filling multiple candidate patches at each iteration, leading to an algorithm 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 prolific 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. As the tasks of researchers and practitioners of these various fields robots have become more and more complex, these 2D images can no longer provide the level of detail and amount of information that they require.

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, the 3D point in the world that generated each image pixel. In fact, there is an infinity of world points (a 3D ray) which could have produced every image pixel. We explain this problem in detail in Chapter 2.1.1. To overcome this ambiguity, researchers have investigated several ways of acquiring information about the 3D scene.

Several techniques and acquisition methods have been developed for obtaining this information (detailed in Chapter 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. These 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.

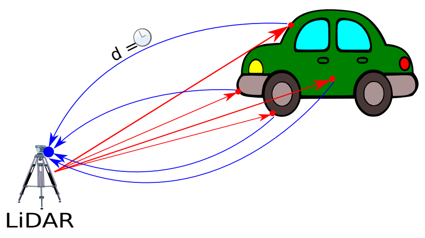


Figure 1.1: A sketch of a LiDAR scan.

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, as the full 3D structure is not known since the laser pulses cannot penetrate opaque objects to see their reverse side.

Artists and produces in Hollywood as well as other consumer media sources have recently become very interested in 3D data. 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. By collecting high resolution scans of actors’ faces, detailed 3D character animation and rendering is made possible. Motion capture suits allow producers to capture 3D movement of actors, allowing for realistic synthetic character motion. LiDAR has even been used by the music industry. Namely, the band Radio Head 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 navigating a suburban environment. A frame from the video is shown in Figure 1.2a. 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 1.2b.

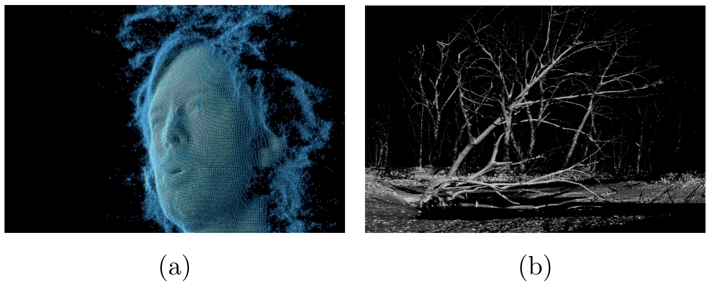


Figure 1.2: **Non-technical uses of LiDAR. A frame form Radio Head's music video "House of Cards" is shown in 1.2a and a piece of art made from a LiDAR scan is shown in 1.2b.**

While LiDAR has enabled neat effects in the arts, it has become arguably 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 artifacts. A popular example of this type of work is the Digital Michaelangelo project, in which researchers performed extremely high resolution scans of the Statue of David for historical preservation, as well as analysis of the tool marks left by the artist. Archaeologists frequently use LiDAR scans to allow computer algorithms to reassemble the pieces of broken ceramic pottery like a jigsaw puzzle.

LiDAR data has also become extremely interesting and important to the military. It is used for mission planning, reconnaissance and intelligence gathering, as well as creating extremely accurate simulations of battle field 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 leads to the capability of creating highly accurate local weather models and forecasts, which can use to precisely time covert missions.

The type of data we are interested in throughout this thesis is referred to as "terrestrial 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 things, such as topographic studies, urban planning, or military mission planning. A typical aerial scan is shown in Figure 1.3.

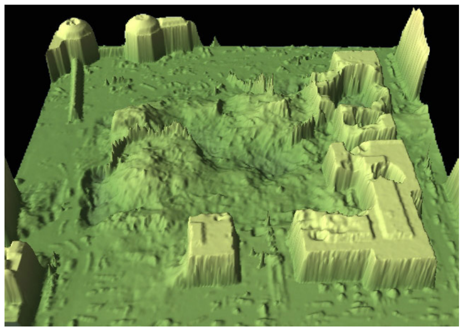


Figure 1.3: A terrain map acquired from an Aerial LiDAR scanner

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. The type of editing we are interested in consists of two main parts - selecting objects and completing, or filling, the part of the scan left by the hole of the removed object. The same technique used to fill holes left by removed objects can also be used to fill LiDAR shadows (discussed in detail in Chapter 2.1.4), 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 and extend in this thesis. This chapter provides an overview of the literature of all of the following chapters. The technical details of the previous work that is directly applied and extended in this thesis are deferred to the beginning of each chapter to 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 software tools that were necessary 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 LiDAR data sets from artificial 3D object models completely artificially. 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 to re-color, or resection, a LiDAR 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.

- 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.

- 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 much like 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 accurate full-object segmentation very easily. The main realization is that segmenting the depth image alone can provide an excellent segmentation in some regions of the object, but has trouble near smooth boundaries from the object to the background, such as the attachment point of the object to the ground. Our goal is to utilize these nice segmentation in the well behaved regions of the object, and then refine the segmentation 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. 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, provide alone. We also use a boundary test to mark pixels which we are now confident belong to the background. This second step refines the segmentation of the object in the region which was ambiguous, while preserving the sharp object boundary obtained by the depth segmentation. 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.

- In Chapter 6, LiDAR Inpainting, we study the problem of filling a large hole in LiDAR data. As the laser cannot see through opaque objects, a phenomenon referred to as a “LiDAR shadow” results behind every object in the scene. These holes are present in nearly every real-world scan. We present an algorithm to synthesize, or hallucinate, data in this hole which looks like a plausible representation of what could have been present in the scene. The major benefit of filling this type of hole is making a scan viewable from a position other than the original acquisition viewpoint, which is done in almost any data exploration attempt. In addition to filling holes produced by LiDAR shadows, we can similarly fill holes in the scene left by our object segmentation and removal technique which we discussed in Chapter 5. 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.

- In Chapter 7, Post-Candidacy Work, we discuss the work to be completed post-candidacy. The work proposed consists of two main parts. 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. Typically very 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 candidate patch using a simple and fast sum of squared differences method, and then perform intensive computations on the proposed best source patch 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. We discuss several new techniques as well as enhancements to existing patch comparison techniques that can be applied as acceptance tests, and explain the combination of several such tests.

  Second, using the idea of acceptance tests, we re-automate the now partially interactive process. That is, rather than immediately prompt the user to take an action after designating a patch as not acceptable, we can instead try to fill a different target patch to see if an acceptable source patch can be found at the new location instead. This can be performed at several target patches in parallel, leading to a slightly less greedy version of an exemplar based algorithm. We will show through several experiments that though only a few bad patches may be found out of hundreds in each hole filling, by eliminating or at least deferring these patches until the end of the operation the results can be greatly improved.

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. We start by describing concepts and algorithms in 2D (as applied to images), and then expand these explanations to describe their application to 3D point cloud data.

2.1 Data Acquisition

2.1.1 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, "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.

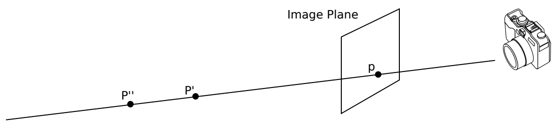


Figure 2.1: The problem with obtaining 3D information from a single image.

2.1.2 Passive 3D Information Acquisition - Reconstruction from Images

This ambiguity is very well studied in the field of computer vision. In fact, many books have been written about understanding and overcoming this ambiguity. The concept of *stereo vision* shows that in many cases, with two cameras and a lot of hard work, the 3D information in a scene can be obtained. The heart of the problem is finding *correspondences* in the two images. That is, for a pixel in the first image, 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 straight forward mathematical procedure. However, identifying these correspondences unambiguously is extremely challenging. The fundamental problem is locating and matching small patches of pixels in the two images which is typically quite difficult (we detail 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 very hard and prone to error.

An extension of the stereo vision problem, *multi-view stereo* or *structure from motion*, 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). Using this type of data set is slightly easier because of the known ordering of the images (that the temporal consistency introduces), so the matching features can be assumed to be spatially near each other. By having a large set of images, it is possible to verify correspondences over multiple pairs of images, which makes the process much more robust than the two image case. In a very popular recent paper [3], Agarwal et al. demonstrates the state of the art of these techniques by attempting to reconstruct an entire city from a massive collection of tourist photographs.

2.1.3 Active 3D Information Acquisition

2.1.3.1 Structured Light Scanning

Structured light scanning is the process of projecting a known pattern into a scene, then taking an image of the resulting overlap of the pattern with the 3D geometry. The pattern is typically a grid or sequences of bars. The way that the pattern deforms when intersecting 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 ($< 10m$). A device that recently popularized a similar technique is the Kinect by Microsoft. The main advantage of this type of acquisition process is that it is very inexpensive - the Kinect sells for only \$100. However, as with most things, there is a significant trade off of cost and quality - the resulting point sets are extremely noisy as compared to the techniques described in the following sections. Additionally, these devices can be quite small, a common depth camera (shown in Figure 2.2), is only about 3 inches cubed.

Figure 2.2: Mesa SR4000 Depth Camera.

2.1.3.2 Light Detection and Ranging (LiDAR)

Light Detection and Ranging (LiDAR) is a data acquisition method which uses time-of-flight calculations on laser pulses to obtain 3D point samples from a 3D scene. 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 texture information. They capture a “2.5-D” image of a scene by sending out thousands of such pulses and using time-of-flight calculations to determine the distance to the first reflecting surface in the scene.

There are two broad categories of LiDAR scanners: “scanning” LiDAR and “flash” LiDAR. Scanning LiDAR scanners work by moving internal motors and mirrors. 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+m). 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. You will often see one or more of this type of scanner on vehicles involved with heavy automation/computational tasks, such as DARPA Grand Challenge vehicles, as shown in Figure 2.3.

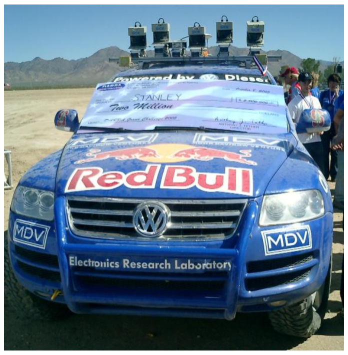


Figure 2.3: The winning vehicle of the DARPA Grand Challenge. Five LiDAR scanners are mounted across the top of the vehicle.

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 2.4.

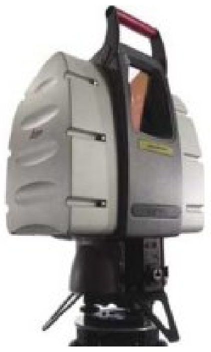


Figure 2.4: The Leica HDS3000 LiDAR scanner, used to acquire all of the data sets in this thesis.

A functional combination of time-of-flight cameras and LiDAR scanners are "Flash LiDAR" scanners. These scanners are physically small, have a long range (100 meters) and acquisition speed near video frame rates. However, they are quite noisy, as well as have a very large price tag.

A very useful property of LiDAR scanners is their ability to "see through" foliage. As shown in Figure 2.5, since each laser pulse travels on a completely 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 resulting points on the back of the scene are sparse, they can still be used to detect objects. This is contrast to an image of the scene, which would likely not show any detectable traces of the existence of the object.

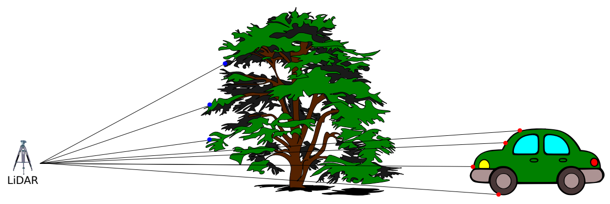


Figure 2.5: A demonstration of the foliage penetration capability of LiDAR. The blue dots show points that were acquired from the occluding object, while the red dots show points which reached the object of interest. A photograph from this same viewpoint may not have shown any, or extremely limited, traces of the car.

2.1.3.3 Data Format and Storage

In most 3D acquisition methods, the resulting data set is stored as an "unorganized point cloud." A typical point cloud is shown in Figure 2.6.

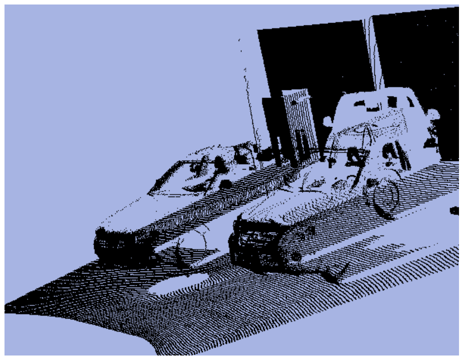


Figure 2.6: A typical point cloud. Two cars scanned with a building in the background.

Some data collection methods acquire points on a spherical grid. When this is the case, the points can be alternatively be viewed 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 point. These pixels are float-valued, so to display them the must be pseudo-colored, as shown in Figure 2.7.

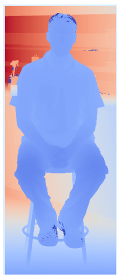


Figure 2.7: A depth image of a man sitting on a stool. Dark blue points are closest to the scanner, while dark red points are farthest from the scanner.

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.

Bulkiness

While scanners are often vehicle mounted, many are tripod mounted. All of the data in this thesis was acquired using the HDS3000. This scanner required a car-battery sized power source, a surveyors 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 2.8.

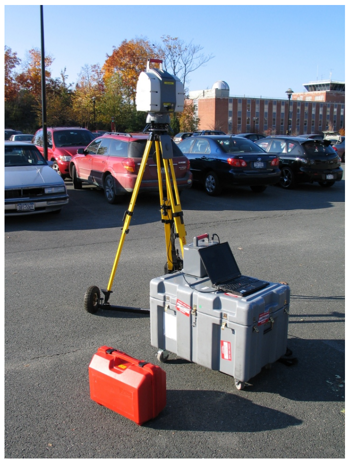


Figure 2.8: A typical scanning session with the Leica HDS3000

Point Density Variability and the Glancing Angle Problem

Most LiDAR scanners acquire data on a spherical or cylindrical grid. This grid is uniformly angularly spaced. Because the distance to the surfaces in the scene is unknown a-priori, the resulting points do not uniformly sample the scene surfaces. Additionally, even points that are on a surface perpendicular to the scanner will have different depths, as shown in Figure 2.9.

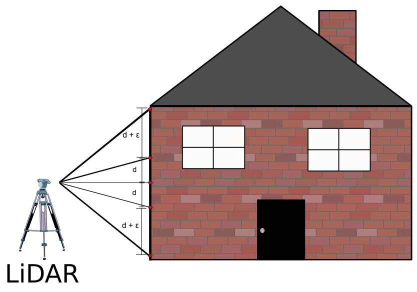


Figure 2.9: The variable point spacing of a LiDAR scan.

Thin Objects

An optical camera actually accumulates the contribution from many points in the world to produce each pixel. On the other hand, a LiDAR pulse approximates seeing an actual 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 2.10, we show a scene in which researches have performed a complex data interpolation in order to detect and fill in these missing power lines.

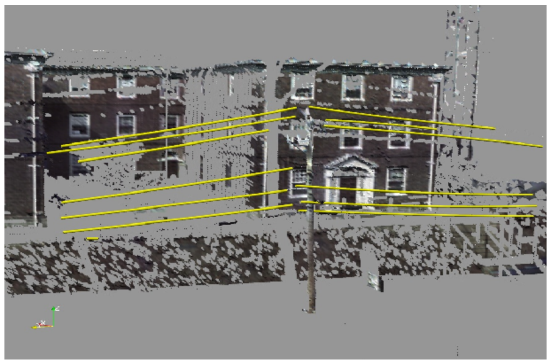


Figure 2.10: Interpolated power lines.

LiDAR Shadows

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 viewpoint from which it was acquired, it leaves a “hole” or “shadow” in the scene, as shown in Figure 2.11.

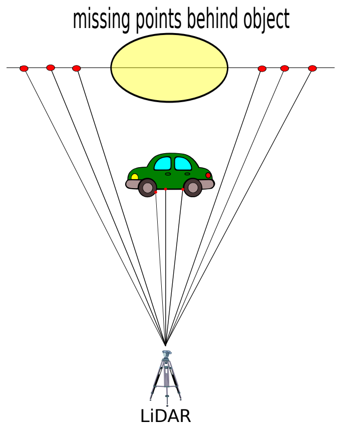


Figure 2.11: The “shadow” left behind a LiDAR scan.

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. However, if this area is not of particular interest, it is still distracting to see large holes in the scene when inspecting other regions. This problem is of particular note, as it is the focus of our contribution in Chapter 6.

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 2.12.

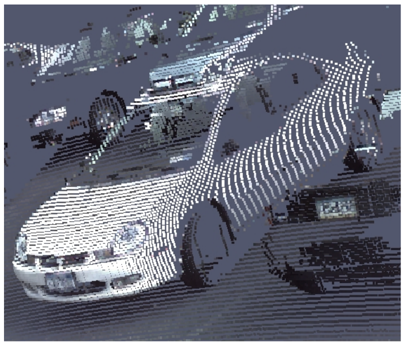


Figure 2.12: Missing points in the windows of a car.

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

2.2 Object Registration, Segmentation, and Detection

The problems of *object registration*, *object segmentation*, and *object detection* are very related. In fact, their very definitions are fuzzy, and in practice, often interchangeable. Solutions to these problems often overlap significantly, and many real-world problems are typically solved by creating complete systems by chaining together solutions to several of the problems we describe in this section. In this section we define these problems and detail previous work in each of these areas.

2.2.1 Registration

Registering, or aligning, two or more items is a common problem across a variety of fields. The problem can be stated as "given two items, find the best transformation to align one with the other." These "items" to be registered vary with the application domain, and can include images, 3D meshes, point clouds, or anything else with a well defined representation. This seemingly benign definition of the problem has two major caveats. First, the definition of "best" is wildly disputed and very domain specific. Second, a "transformation" 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 2.13 shows two problems that, although appearing quite similar, are actually extremely different in complexity.

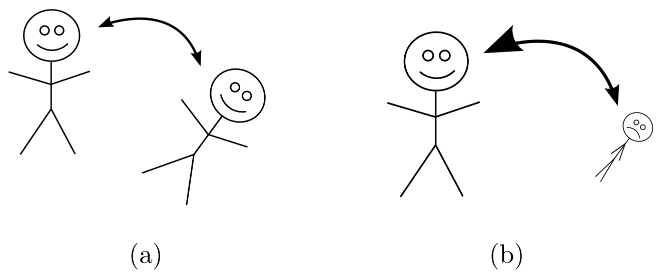
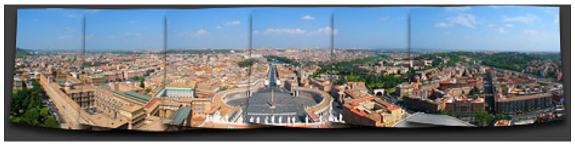


Figure 2.13: Two image registration problems. a) A simple image registration problem. A 2-DOF solution can exactly align the two stick figures. b) A difficult image registration problem - a complex deformable transformation or articulated model would be necessary to align the two stick figures.

In fact, the two “items” do not even have to be of the same type, or even in the same space. That is, we can attempt to register an image to an image (both 2D), a point cloud to a point cloud (both 3D), or even an image (2D) to a point cloud (3D).

2.2.1.1 Image Registration

In image processing, the registration problem is defined as finding a transformation (typically an affine transformation is required) which aligns two images so that they best overlap. The result of this alignment can be used for many purposes. For example, aligned images can be used to create panoramas, as shown in Figure 2.14.


(a) Original images aligned and overlaid.

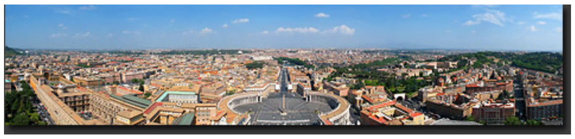
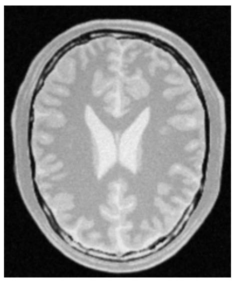

(b) The resulting panorama after blending.

Figure 2.14: Creation of an image panorama.

In medical image processing, there are several reasons to align images. By overlaying images from multiple modalities of the same organ, clinicians can obtain a better, more complete understanding of a situation. By aligning images of the same modality of a patient acquired over a period of time, changes can be detected which could lead to an understanding of the patient's condition. Finally, by aligning images from multiple patients, clinicians can study statistical averages to learn and better understand baselines from which to compare new patients. An example of the registration problem on two modalities of brain images is shown in Figure 2.15. By accurately overlaying these two images, a more complete understanding of the structures of the patients brain is obtained.


(a)

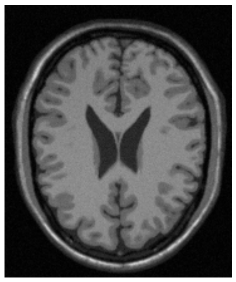
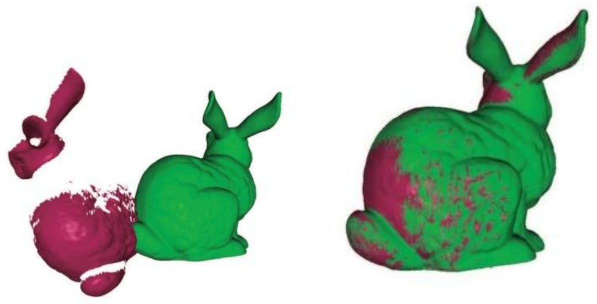

(b)

Figure 2.15: Medical image registration. a) A proton density MRI image of a brain. b) A T1 MRI image of a brain.

2.2.1.2 Point Cloud and Mesh Registration

In 3D, the problem is quite similar. We wish to take two point clouds and find a transformation which aligns one to the other. An example of this is shown in Figure 2.16.



(a) Two partial point clouds of the Stanford Bunny. (b) The resulting registration of the point clouds.

Figure 2.16: Point cloud registration.

Registering point clouds can serve several purposes. For example, if two point clouds of the same scene are available and both are partially incomplete, aligning the available data can “fill in” the missing parts of one cloud using the points in the other. Additionally, as we will see in Chapter 2.2.3, registering a point cloud or mesh of an object to a subset of a data set (e.g. if one object was the bunny, while the other was the scan of a scene containing a bunny), is exactly the “object detection” problem.

There is an extensive body of work on 3D object registration. A very popular technique to register point clouds is Iterative Closest Points (ICP) [170]. The idea is, for each point, to find its closest point in the other data set. Once these correspondences are computed, the best transformation between the point sets is found. Then, after transforming one of the point sets by this aligning transformation, the correspondences are re-estimated. This procedure is repeated until convergence. The good news is that this algorithm is guaranteed to converge to a local minimum. The bad news is that in practice there are a *huge* number of local minima, so the initial transformation from the feature correspondences must be extremely close to the actual best alignment, or ICP will not improve the solution. There have been many improvements to ICP. The original ICP algorithm performs extremely poorly in the presence of outliers. Phillips and Tomasi addressed this problem by introducing a fractional point set distance, which accounts for outliers in the correspondence identification step [125]. 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 has been widely used [36]. Finally, a paper by Rusinkiewicz and Levoy [131] provides a rigorous performance evaluation of several ICP methods. In a publication by Mitra et al.[116], ICP is generalized to, instead of consider the problem as a point-set to point-set registration problem, instead treat the problem as aligning a point set with the surface which the target point set represents. Fitzgibbon propose an energy function which can be directly minimized and has a large basin of attraction, which is of great practical value [65]. Specht et al. experimentally compare the ideas of registering range images directly versus registering their corresponding meshes [140]. They conclude that the performance is data set dependent, but propose future work on combining the two concepts. A discussion of a very separate class of registration algorithms based on computing and aligning *features* is deferred to Section 2.2.3.

2.2.2 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." This problem can be described as "given the known approximate location of an object, 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 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 not labeling any 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 are 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. It can be used to aid in constructing model databases by quickly extracting objects of interest. Additionally, in the field of robotics, for a robot to perform identification tasks it must first separate all of the members of this object to give to the recognition algorithm as a whole.

The second type of object segmentation is “object part segmentation.” Here we are interested in extracting coherent collections of members which belong to a “part” of an object. 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. For example, we wish to divide a human into “arms”, “legs”, “head”, “torso”, etc. 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. This type of segmentation is also useful for object recognition, because if we are convinced we have found two legs, two arms, a torso, and a head, then we might conclude that we have found a human.

In Figure 2.17 we show an example of the difference between whole-object and parts-based segmentation.

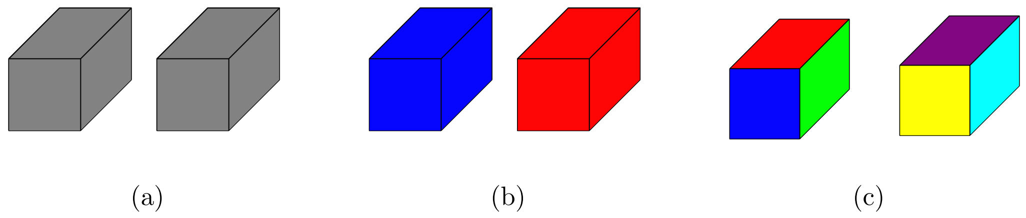
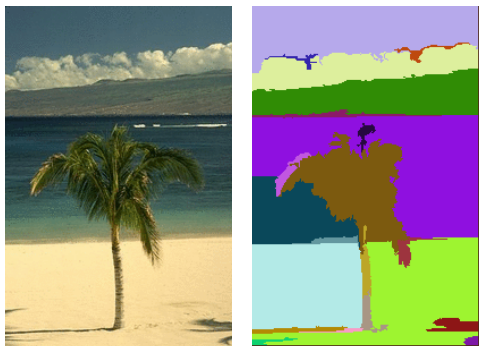


Figure 2.17: **Two types of 3D segmentation. a) A scene consisting of two cubes. b) Whole object segmentation. Each object is shown in a different color. c) Object part segmentation. Each part of each object is shown in a different color.**

2.2.2.1 Image Segmentation

The problem of image segmentation is the problem of dividing an image into parts, or regions, with different properties. 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 segmentation is shown in Figure 2.18.



(a) An image. (b) The segmented image.

Figure 2.18: An example of a segmented image.

Other times, 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 goal of both of these types of segmentation at the end of the day is 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 proved 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 [96] 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 [70]. 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, 101]. 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 [163]. 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 pixels 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 hopes 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 [38, 88] have been followed by significant study and improvement, including recent hybrid techniques [164].

A class of very powerful methods that have recently become very popular in image processing are graph-based 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.

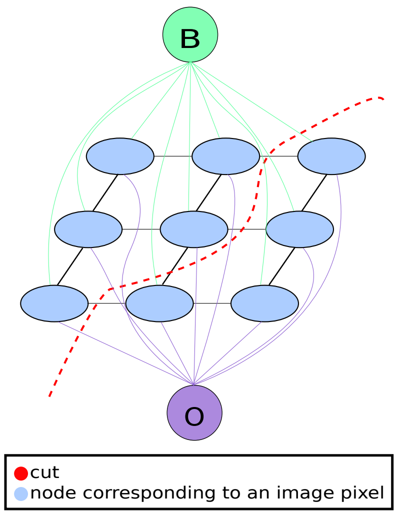


Figure 2.19: A graph cut image segmentation

In a publication by Boykov and Jolly [29], the idea of performing segmentation on an image interpreted as a graph is introduced. This spurred a decade of follow up work [28, 26], 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 object 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 and Blake, 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 [129]. This serves an identical 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. This graph-cut based segmentation was improved [137] by the idea of the “normalized cut”, which optimizes both the total dissimilarity between the segments as well as the total similarity within the segments. Another follow up work by Zeng et al. proposed a topological constant on the segmentation which is a very desirable property in real world segmentation problems [169]. Other publications [149, 7] demonstrated that the binary graph-cuts based segmentation is extended to a multi-label problem. These techniques iteratively solve binary graph cut problems, optimizing a multiple-object-type segmentation problem. Li et al. first perform a graph cut on superpixels of the image [103]. 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. 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 [92]. In this thesis we directly utilize graph-cut based image segmentation, so we present a full discussion of the methods in Chapter 5.1.

2.2.2.2 Depth and LiDAR Segmentation

While much of the work in image segmentation has been on standard color images, there has also been some work on segmenting depth images. Fernandez and Aranda use a dynamic programming approach to label points in a pair of stereo images [64]. They use the reconstructed depth images to refine and verify the segmentation of the color images. In recent work on 3D television (3DTV), depth image segmentation has become popular. Researchers are interested in quickly estimating coherent planes of objects so that their images can be displayed as a stereo pair that is interpreted as an image with depth. The result is the ability to create a virtual view of an image from a slightly different perspective that can be projected to one eye of the observer, while the original image is projected to the other eye (by the use of active or passive filters, either in the device or on the viewer (3D glasses, etc)). In another paper [157], Wang et al. are interested in filling occluded parts of the scene from the new view by segmenting the depth image so that it can be intelligently filled in without introducing erroneous points by simply interpolating between foreground and background objects. In yet another publication [40], Dal Mutto et al. attempt to automatically segment an entire image into regions consistent in color and depth. The most common technique for depth image part-segmentation [126, 13, 8, 12] is to identify planar surfaces and label each surface as a part. This is useful in industrial applications such as quality control, but in this thesis we are interested in whole-object segmentation, so these techniques do not apply.

In this thesis we are most interested in segmenting LiDAR data sets. Due to the huge size of these data sets, segmentation is a particularly important step in any processing pipeline. By segmenting large planar objects from a scene (e.g. the ground plane), the set of points left to search for other tasks such as object detection is greatly reduced. Just as in image segmentation, there are two complimentary techniques - clustering (analogous to the region-based methods) and surface growing (analogous to the boundary methods). In a clustering algorithm, the idea is to group together points that are spatially near each other and share some common properties. In a surface growing algorithm, we start at a seed point and move outwards, collecting points that fit a set of criteria until the criteria are no longer met (the boundary condition).

Several methods [147] take a similar approach as the parts-based range image segmentation and simply detect and extract planar surfaces in the point cloud. Segmenting planes is the most common and useful segmentation to perform in range data. 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. Yang and F use 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 [167]. This is opposed to most commonly used methods which are very local in nature.

Many other techniques have been introduced, relying on various mathematical formulations. Biosca and Lerma draw from relatively recent work in fuzzy clustering to segment point clouds [24]. Boykov and Kolmogorov use graph-cuts based technique to extract the surfaces of human organs [27]. Yu et al. use 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 [168]. Liu and Zhang use spectral clustering to extract components of a mesh [107]. They construct an affinity matrix between all of the points and then compute eigenvectors of this matrix which can be interpreted as clusters.

In a publication by Kalogerakis et al., meshes are separated into meaningful parts [87]. For example, on a mesh of a human the mesh would be divied into "head", "leg", etc. Their method creates 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.

A couple of articles use a single pass clustering algorithm [91, 90]. A radially bounded nearest neighbor (RBNN) tree is computed on the points, and then use an iterative cluster merging strategy to compute segments. This method is extremely fast, but the segmentation produced is quite coarse.

Marshall et al. segment range images into primitives (spheres, cylinders, etc.) by using least squares fitting to a known collection of models [111]. This type of technique works well in a very controlled environment, but in uncontrolled real world scenes, objects rarely fit these models well.

In a publication by Latecki and Lakaemper, an expectation maximization algorithm is used to determine the outlines of rooms acquired by a mobile robot [97]. This is a crucial step in localization, or allowing the robot to know where it is in an environment. With a similar goal, Latecki et al. presents a method to sequentially segment an merge line segments to create the same type of wall map [98]. Jagannathan et al. use a curvature based region growing approach that is parameter free [83]. They use techniques of graph morphology to grow a region by dilating the mesh graph and only keeping nodes attached that pass a specified 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.

A publication by Golovinskiy and Funk explores creating a graph on a point cloud, and using a min-cut algorithm to separate objects from the background [72]. The 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 is provided. The weights on the graph edges decrease with the distance from the foreground points.

One way to approach the segmentation problem is using principles of perceptual organization. Lee and Schenk use principles like parallelism, continuity, connectedness, and others to segment point clouds [99]. The method is hierarchical, first grouping points into patches, then patches into surfaces, and finally surfaces into objects.

Many segmentation methods are classifiers. That is, they rely on training data to “learn” classes of objects, then attempt to determine which one of the specified set of classes a group of points belongs to. In an article by Anguelov et al., a Markov Random Field is created on the point set to enforce the constraint that points near to each other are likely to belong to the same object [9]. The authors then train a classifier on a large database of object classes and use graph cut inference to classify the points in a scene.

There is a large body of work in segmenting building from the ground in aerial LiDAR scans [155, 16]. In this problem, typically a statistical approach is taken to classify points as “ground” or “not ground.” This relies on the background being extremely uniform, which is far from the case in general terrestrial LiDAR object foreground/background segmentation. Therefore this type of technique is not applicable to the data sets of interest in this thesis.

2.2.3 Object Detection

In the object registration problem, our goal was to align two entire images. In the *object detection* problem, we would instead like to identify the location of a specific object, or part of an image, in a target image. This problem is also referred to as “object localization.” This is a hard task for even humans to perform. In a publication by Bravo and Farid, the effects of clutter and orientation on the ability and accuracy of a human observer to recognize objects in images is studied, demonstrating that this is indeed a very hard problem [31]. In this section, we describe algorithmic methods to perform such detections.

2.2.3.1 Object Detection in Images

Most object detection techniques rely on finding *keypoints*, and then computing *features* or *descriptors* at these keypoints. A keypoint is simply a good place to compute a feature. A feature is simply some way of representing the local behavior of the image at a particular point. The basis of many object detection algorithms is comparing these descriptors to either features from a target image or a database of images.

A very popular feature based method is the *Scale Invariant Feature Transform* [108]. It constructs images at multiple scales (blurred and downsampled versions of the original image), causing it to be quite robust. Similar algorithms exist which contend to be improvements to this idea, either in speed or performance, such as Speeded Up Robust Features (SURF) [17]. SURF uses integral images ([151]) to enormously speed up the necessary correlation computations. A great summary and performance comparison of several features can be found in an article written by Mikolajczyk and Schmid [115].

In a publication by Matei et al., feature based methods are combined with coarse-alignment methods using an algorithm for approximate nearest neighbor search which uses a randomized hashing technique [113].

The *shape context* [19] is another local image descriptor. It encodes the relative direction from each edge point in an object to every other edge point in the object. This is a very good approximation of local shape.

2.2.3.2 3D Object Detection

The problem of object detection is typically framed as "given a 3D mesh, point cloud, or other representation of an object, determine where in a data set the object exists, if at all." A main technique used to detect objects in 3D is to compute "features" at specific points on the object, and compare them to features in the scene. Features can be as simple as the surface normal estimate of the scene/object, or very complicated high dimensional vectors. Depending on the type of feature used, one more more matches are required to localize the object in the scene. This type of match, however, leads to an unacceptably high false positive rate. An example of a bad alignment resulting from a good feature match is shown in Figure 2.20.

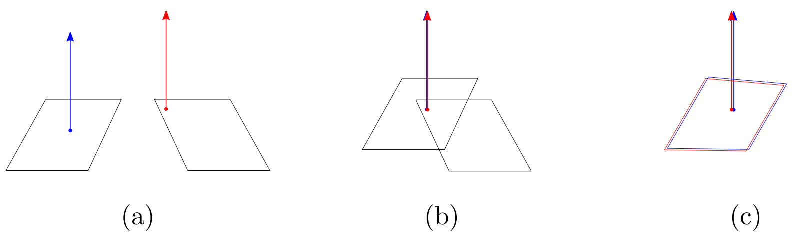


Figure 2.20: Two objects (planes) with perfectly matching normal features that will result in a bad registration. a) Two objects (planes) with ambiguously matching features. b) The objects incorrectly aligned using this feature. c) The expected correct alignment.

To remedy this, the notion of *geometric consistency* of feature matches was introduced. 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. This is a reasonable notion, but its computational complexity is overwhelming. The number of points N_s in a typical scene may be near a million, and the number of points on a typical object, N_o may be 100k. This means that there are 1.1M features to compute, which is itself a daunting task. Then if a percentage of these feature matches are considered in a geometric consistency framework, there are a combinatorial number of comparisons to be performed. A sketch of the problem that is addressed by geometric consistency is shown in Figure 2.21.

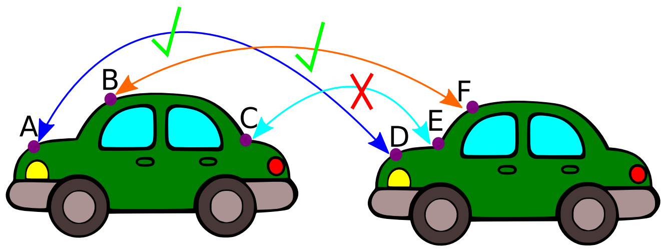


Figure 2.21: Diagram showing geometric consistency.

To remedy this problem, we need the notion of "keypoints." The idea is to first find points that are reasonably "descriptive." By descriptive, we generally mean that a point has a high curvature value, and/or is significantly different from its neighbors. For example, a point on a planar region of an object is not very descriptive, as it could look very much like any other point on any other planar region in the scene. However, the corner of a box, for example, is quite descriptive, and should match to a much smaller set of points in the scene. Once we identify a set of keypoints, we then only compute features at those points, making our feature comparison and geometric consistency tasks much smaller and computationally tractable.

In a paper by Horn, a method for representing the shape of surfaces, the Extended Gaussian Image (EGI) is introduced [78]. EGI's are a description of a surface by projecting its points onto a unit sphere. EGI's have the nice property that they are translation invariant, and rotationally well behaved, as the EGI rotates exactly as does the object. Makadia et al. uses a fast comparison of EGI's performed in the frequency domain to detect objects in a scene [110].

A very popular 3D descriptor is the Spin Image [85, 84]. 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. The resulting spin image is essentially a histogram, but the major benefits of this representation are that it is 2D, and that it is completely independent of the pose and position of the object. Unfortunately, no descriptor is perfect, and the spin image descriptor suffers greatly in the presence of occlusions. Spin images have also been extended to include color information corresponding to the points [33].

Kortgen et al. introduces the 3D shape context, a 3D analog of the original Shape Context [95]. This is a very natural extension - the 3D shape context encodes the relative locations of points in a spatial histogram. The extension is from a planar, polar histogram to a 3D, spherical histogram.

The transformations computed from feature matches produce a *coarse* alignment of the two data sets. Typically refinement step is performed after the coarse alignment. The most prolific and very highly cited technique for alignment refinement is ICP, which we previously mentioned in Section 2.2.1.

2.2.3.3 3D Object Detection Verification

Our first contribution in this thesis is related to the verification of the result of object detection algorithms, so it deserves special attention here.

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. There, however, have been a few methods to detail here.

In a publication by Vasile and Marino, a method is introduced to find military vehicles in LiDAR scans of outdoor scenes [148]. Their final verification procedure, a “goodness of fit” test, used a weighted spin image [85] correlation coefficient. A parts based approach has also been implemented to classify objects with heavy self-occlusion into one of several predetermined classes [81]. Chevalier et al. 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) [37]. In each of these cases, our verification procedure could be used to provide an analyst with a method-independent, easily-interpretable physical check of the final detected object position.

Visibility consistency Huber used a method of determining the quality of alignment between two surfaces derived from range scans [80]. A free space violation occurs if, after alignment, points in one of the scans occur in the free space of another scan’s perspective. This technique requires preprocessing to extract surfaces from the range images, and hand-labeled training data to estimate the probability distributions of the distances between two surfaces along each ray in the case of correct and incorrect alignments. Mian et al. introduced the related concept of “active sensor space violation” as a means of determining the accuracy of a model-to-scene registration [114]. 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.

In a publication by Iv et al., a two-step method was proposed for finding multiple similar objects in large data sets [82]. First, possible positions are identified using spin images. These positions are then verified using Extended Gaussian Images [78]. The verification procedure requires hand-labeling parts of the input to provide exemplars of the objects of interest. A verification function based on a learned linear combination of several measures of registration accuracy was proposed as well [139]. These measures include variation in the normals of corresponding points, the stability of the covariance matrix of the estimated transformation, and a novel boundary alignment check.

2.3 Filling-in Missing Data

2.3.1 Image Completion

The image completion problem is the problem of attempting to fill a hole in an image. This hole can either be caused by corruption of originally valid data or by occlusion in the scene itself. For example, a photograph may have been folded or torn leading to corruption of the data it contains, or 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. The problem is also referred to as *inpainting*. This name comes from the name given to the process in which an artist corrects blemishes in a physical painting manually.

The image completion problem is typically approached in 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.

2.3.1.1 Differential Equation Based Image Inpainting

One class of techniques which solves differential equations over a hole in an image, attempting to propagate information smoothly from the boundary into the hole. Researchers have used variations of this technique [21, 34, 35] to attempt to fill small holes in an image. A simpler and much faster approach was presented in an article by Oliveira et al. [120]. A diffusion kernel is repeatedly convolved with the image, building up color in the hole during each pass. This solution is orders of magnitude faster than actually solving the differential equation, while yielding similarly good results. A similar method "pushes" the image values directly into the hole with a simple summation [145].

While reasonable results have been obtained on small, thin holes, differential equation based methods are typically not suitable for filling large unknown regions because they cause heavy blurring artifacts. As this type of hole is the focus of our contribution in this thesis, we will not focus on this type of method.

2.3.1.2 Exemplar/Patch Based Image Inpainting

A second, and recently strongly preferred, class of techniques to inpaint image holes are 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. An example of a case in which it is impossible to recover the actual scene information is shown in Figure 2.22.

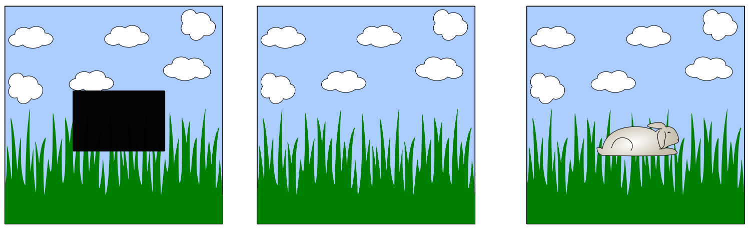


Figure 2.22: A demonstration of what we would expect the inpainting to produce, and how it is unrelated to what could have actually been present in the scene.

A review of many techniques that attempt to solve this problem can be found in a publication by Telea [144].

Instead of simply copying the patches as-is into the hole, one article [60] demonstrates that the patches are composited using a multi-scale Laplacian pyramid approach in an attempt to ensure that there is a smooth transition into the hole.

In a publication by Pérez et al., it is shown that by solving a Poisson equation with the known hole border as the Dirichlet boundary condition, part of one image can be very convincingly pasted into another image [123]. A thorough discussion of this method is provided in 6.2, as we utilize these techniques directly in our contributions.

Levin et al. realize that by synthesizing image gradients and then reconstructing the image from the gradients, a much smoother completion can be obtained [100].

It has been repeatedly shown that the order in which these patches are copied is extremely important. The ordering of the patch copying 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 is shown in more than one publication [39, 22]. In an article by Xu and Sun, to determine the filling priority of a target patch, the authors compute a measure of the similarity to nearby target patches [166]. 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.

In work by Sun and Jia, two very useful techniques are introduced [142]. 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 hopefully improve the overall quality of the inpainting. The other 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 get right.

A fast algorithm for searching for an exact closest matching patch is presented in a document by Xiao et al. [162]. While very complicated, it allows efficient GPU implementations which enable 3D patch matching in reasonable computational time.

A randomized algorithm for constructing the Nearest Neighbor Field (NNF) is presented in another publication [14]. 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. A more recent article takes this one step further by extending the possible patch set to allow for rotations of patches, while still keeping the massive performance increase [15].

Researchers [135, 136] have 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 use a similar technique to remove shadows from images while maintaining the texture beneath them [104]. These gradient based inpainting techniques are very important to this thesis, as they are the basis for our extension of this concept to LiDAR inpainting. We explain these procedures in detail in Chapter 6.2.

In a paper by Goyal and Diwakar, the region to search for source patches is not the entire image, but rather a window of specified size around the target patch [73]. This is typically a reasonable assumption, but disallows some of the power of patch based inpainting, as we will discuss in 7. Thangamani et al. recognize that the search space can be reduced to neighboring superpixels instead of to a circular region with a fixed radius [146].

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 whole, 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 from the ground truth region, yet still look very convincing. This "convincingness" is exactly our goal. The idea of "evaluation by questionnaire" is used in a publication by Kawai et al. [89], among others. The authors simply ask humans questions like "which image looks better?", etc.

In more than one publication, several small modifications are proposed to the algorithm [128, 39]. A major observation that the confidence value quickly drops to zero as the filling proceeds towards the center of the hole. A regularization term is introduced to remedy this problem.

In an article by Criminisi, the importance of the ordering of the filling is pointed out [39]. He notes the importance of filling linear structures first, as these are critical to human interpretation of the resulting image. If the linear structures are broken, it is almost always obvious that the image has been modified.

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 [94]. 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 [7]. 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 ($< 200x200$), the technique takes tens of minutes. Since the complexity grows exponentially with the image size, this is not yet feasible for real-world images.

In one publication [138], Simakov et al. introduce 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 "incorrect" structure, for example, arbitrarily placed patches from the source image, cannot be present in the output image, leading to necessarily artifact-free output. While this method works well, it is very slow, taking 5 minutes for images on the order of 250x200 pixels.

Some researchers [25, 161] have filled sequential series of images (video) using similar methods. In a publication by Wexler et al. [161], a multi-scale method is introduced, 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 also note that the average patch comparison problem (that a smooth signal will compare well to non-smooth signals).

Almost all of these patch-based inpainting techniques are "greedy." Because of this, it is extremely important to get every patch "correct." Once a single "incorrect" patch is copied, the result in the remaining region of the completion is very poor. To remedy this, many publications present a method to use the same patch copying technique but presented a global optimization formulation using belief propagation [94, 93]. Due to the extreme number of labels in this graph labeling formulation, a special form of belief propagation known as Priority Belief Propagation was also necessary and introduced.

2.3.1.3 Texture Synthesis

Texture synthesis is an almost identical 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 pebbles, and create a large region of pebbled ground in an image. It has been shown that many approaches to this problem can produce visually appealing results.

In one publication by Efros and Leung, a Markov Random Field (MRF) is constructed on the source texture, and one pixel at a time is synthesized based on the probability of the occurrence of a pixel given its neighbors in the already known region of the resulting texture [62].

Efros and Freeman append existing patches to the existing texture using a graph-cut technique to find the best boundary at which to join two patches [61].

In addition to simply synthesizing texture in an "empty" region, similar methods can be used to "transfer" texture from on image to another. For example, a pair of images, one original and one filtered is provided to the algorithm in [76]. The process by which the filtered image was produced from the original is "learned", then applied to a new, independent input image.

A very interesting system which uses these ideas is presented in one publication [10]. The user is allowed to "paint" (using a free-draw type tool), and then texture is applied to these rough drawing from a small patch of a different texture. For example, by drawing a single pink line on a solid green background, a row of flowers on a grass background can be produced.

Essa and Kwatra formulate the texture synthesis problem as a global energy minimization problem so that the entire textured region is optimized simultaneously [63]. As with most global techniques, it is must slower than competing algorithm, taking around 10 minutes for even a very small (256x256) texture

2.3.2 3D Hole Filling

There has been previous work on filling small holes in range data. For example, Stavrou et al. use 2D image repair algorithms on the depth image of a LiDAR scan [141]. Their technique is only demonstrated on very simple scenes with very simple objects. In a publication by Sharf et al., a hole-filing algorithm for point sampled surfaces is proposed [134]. The authors note several problems with working in this domain (though they seem to do so successfully), including the high number of degrees of freedom of aligning two point-sampled surface patches, the difficulty in defining a coordinate system, and the boundary of a hole being ill-defined. A similar method is described in another article [122] where the point-sampled surface hole filling is extended to work with textured scans. Once a candidate patch of points is coarsely positioned near the hole, a height field is formed and a Poisson equation is solved to join this patch of points smoothly to the points defining the hole.

In an article by Xu et al., an image is used to guide mesh hole-filling [165]. By estimating normal vectors for points inside the hole, the geometry of the hole can be reconstructed. This method relies on training image patches.

Techniques based on stereo [68, 69] and structure-from-motion [41, 1, 32] use multiple images to estimate 3D geometry. This geometry can be quite accurate, but is generally much sparser and more irregular than is typically acquired with a range scanner. Our interest in this paper is in learning 3D structure from a single image/scan pair.

Our overall technique is inspired by recent work on exemplar-based image inpainting [39] and image analogies [76]. Criminisi presented an algorithm for automatically filling in a desired target area (e.g. a region to be replaced in a digital photograph) with patches from a source region so that the resulting image seems un-manipulated [39]. We adopt a similar idea of filling in LiDAR holes in a prioritized order depending on the amount of known geometry near a given pixel on the hole boundary. Hertzmann et al. automatically created image analogy filters by transferring a learned relationship between a (original image, filtered image) pair to a new original image [76]. Our algorithm uses a similar idea: we create a database of (image patch, 3D geometry) pairs from the non-hole regions of a scan to estimate the 3D geometry that corresponds to a new image patch.

When reconstructing a function from its gradients, the gradient field must be integrabile for an exact solution to exist. Unfortunately, by inpainting a gradient image, this integrability requirement is almost never satisfied. Ramifications of this fact have been discussed in great detail [124, 159, 5, 6, 74]. Fortunately, though, the solving the Poisson equation often provides a reason solution dispite this theoretical problem. There have been several techniques for solving a discrete Poisson equation [30].

There are many very mathematical papers discussing the problem of reconstructing a surface from its gradients. One publication by Wei and Klette provides a summary of several classes of this type of algorithm [160]. In another paper [118], the integrability constraint is not enforced, but instead the problem is solved in a higher dimensional space which has a closed form solution. Despite this work, in practice the gradient fields we produce seem to be well-enough behaved as to produce good results even though they are not theoretically integrable.

A subfield of computer vision which is very interested in the process of recovering a surface from a gradient field is Shape From Shading (SFS). In the shape from shading problem, algorithms attempt to use shadows and lighting in a single image to determine 3D properties of the scene [77, 67]

A new field in which there is a lot of active work on depth hole filling is 3D-television (3DTV). Several methods [119, 71] have been presented to fill the holes necessary to reconstruct 3D views of a scene from a different view point.

Bhavsar and Rajagopalan inpaint depth directly [23]. 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.

In a publication by Becker et al., 3D patches are directly copied from elsewhere in a LiDAR scan to fill a hole, guided by an image of the scene [18]. This type of technique does not have a smoothness constraint the on 3D geometry, so there is nothing preventing the resulting geometry from being misaligned near the center of the hole.

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

Wang and Oliveira have published a document takes advantage of the natural symmetry and planarity of most objects and repairs holes in the surfaces [154].

Wang et al. simultaneously inpaint color and depth values [156]. However, we will show in Chapter 6 that this technique only works for extremely small holes, or holes that have approximately uniform depth.

Another publication [150] extends the differential equation based inpainting methods mentioned in Chapter 2.3.1.1 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.

In multiple publications [158, 132], a small, local patch of points is projected onto a plane. These points are treated as a height field, and the gradient of this field is used to reconstruct the surface. This technique is very sensitive to the choice of projection plane, and has only been demonstrated to fill very small holes in a mesh. It is also sensitive to the resolution of these projected images.

CHAPTER 3
Custom LiDAR Tools

While image processing can be considered a relatively mature field, LiDAR processing is still in its infancy. 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 Microsoft Paint and Adobe Photoshop. No such packages are prolific for 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 were developed and used to support the contributions in our research. First, we have developed a synthetic LiDAR scanner to produce data sets similar to those that would be acquired from a real LiDAR scanner, but from 3D models and software rather than laser and real world scenes. Second, we have implemented an interactive recoloring, or resectioning, algorithm to transfer the color from a digital photograph onto a LiDAR scan. Finally, we mention several other tools that were developed to study and test known algorithms and to support related work. These tools are all released under open source licenses and have been made available for other researchers to find and use, accelerating the advancement of the field.

3.1 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. So 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 prevalent, 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 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 algorithms are behaving properly before moving to real-world LiDAR scans. If more realistic data is required, noise can be added to the points to attempt to simulate a real LiDAR scan.

Developed as a tool for this research, the synthetic LiDAR scanner was also submitted to the Insight Journal [42] 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 Best Submissions of the Year [43]. It has since been adopted by and integrated into the Point Cloud Library (PCL), a recently released software toolkit for 3D data processing.

Scanner Model

We have 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 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. This ray casting is performed using a Modified BSP Tree, a spatial data structure to facilitate ray-triangle set intersections that are much faster than a full linear search. This data structure proved significantly faster than the Octree [133] for this task.

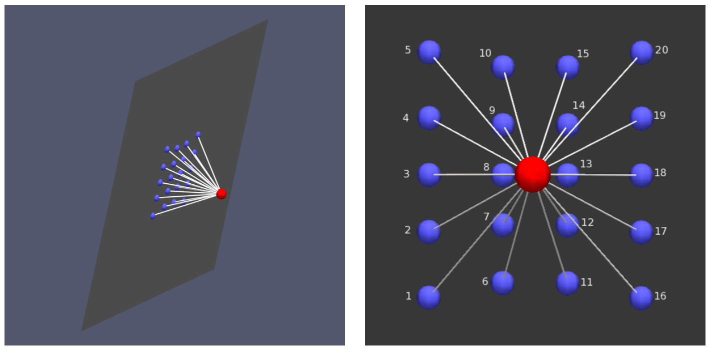
Several parameters defining the scanner orientation, angular resolution, angular bounds, and noise model can be specified. The synthetic scan output is identical to that from the real Leica scanner, in the proprietary PTX ASCII format.

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 ϕ angle (how far “up and down” the scanner should scan)
- Min/Max θ angle (how far ”left and right“ the scanner should scan)
- Scanner transformation - the orientation of the scanner in the scene.
- Number of strips (sampling in the θ direction).
- Number of points per strip (sampling in the ϕ direction).

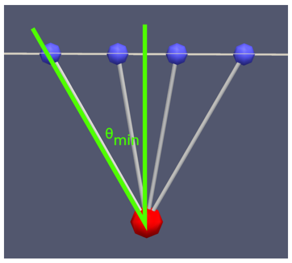
To demonstrate the angles which must be specified, a $(\phi = 5, \theta = 4)$ scan of a flat surface was acquired, as shown in Figure 3.1a. Throughout these examples, the red sphere indicates the scanner location, the cylinders represent the scan rays, and the blue points represent scan points (intersections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



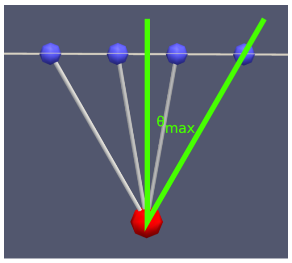
(a) 3D View of the acquisition of the scene. (b) Order of point acquisition.

Figure 3.1: Diagram of acquisition process.

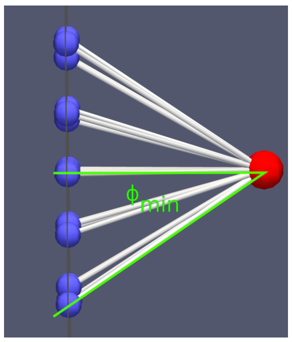
The *theta* angle of a ray is shown in Figure 3.2a, 3.2b, a top view of the scan of a flat surface. The min and max θ angles are labeled. It’s range is $-\pi$ to π. $-\frac{\pi}{2}$ is left, $\frac{\pi}{2}$ is right.



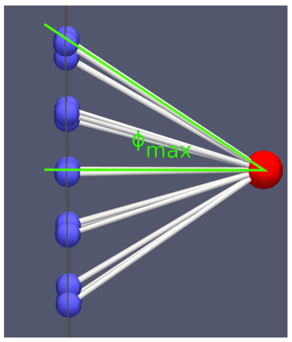
(a) Minimum θ angle (viewed from above)



(b) Maximum Theta angle (viewed from above)



(c) Minimum Phi angle (viewed from the side)



(d) Maximum Phi angle (viewed from the side)

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.

The elevation angle, ϕ, is shown in Figure 3.2c, 3.2d, a side (left) view of the scan of a flat surface. The min and max ϕ angles are labeled. It's range is $-\frac{\pi}{2}$ (down) to $\frac{\pi}{2}$ (up).

Figure 3.3: Diagram of Phi angle settings. This is a view from the side. The scanner is represented as a red sphere, while the acquired points are shown in purple.

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, you 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 3.4 to demonstrate this.

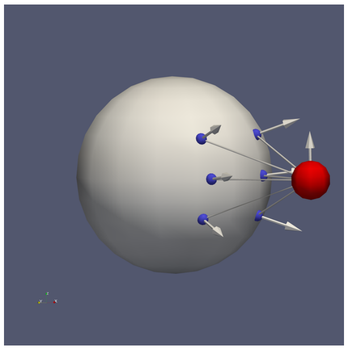
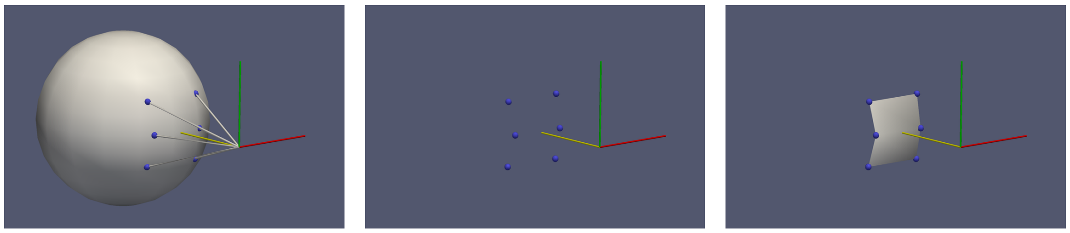


Figure 3.4: Scene intersections and their normals

Mesh Output

In addition to the point cloud output, the synthetic scanner provides the option to mesh the output points. This type of meshing is straight forward for scanners that acquire points on a grid. We provide the option of using a naive grid mesh, or performing a Delaunay triangulation to create a triangle mesh from the LiDAR points. An example of the output point cloud and corresponding mesh is shown in Figure 3.5



(a) The scan/scene setup. (b) The resulting points. (c) The resulting points and mesh.

Figure 3.5: Effect of CreateMesh flag.

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 3.6a. In a real world scan, however, this is clearly not the case. To make the synthetic scans more realistic, we have 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 (double LOSVariance). An example of a synthetic scan with LOS noise added is shown in 3.6b. 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 (double OrthogonalVariance). An example of a synthetic scan with orthogonal noise added is shown in 3.6c. Note that the green (noisy) rays are not aligned with the gray (noiseless) rays, but they are the same length.

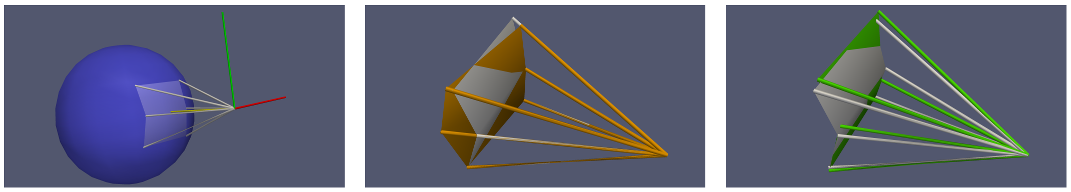

(a) A noiseless synthetic scan (b) A synthetic scan with line-of-sight noise added (c) A synthetic scan with orthogonal noise added

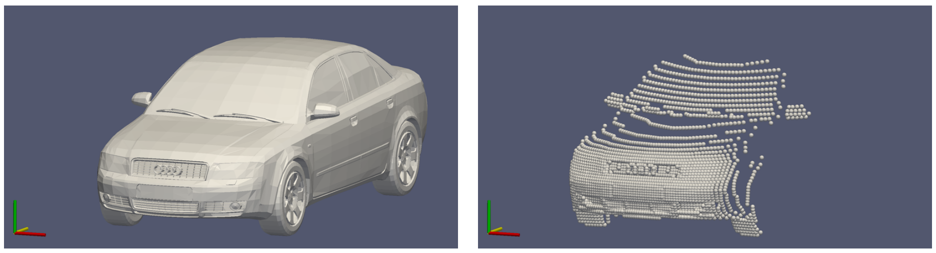
Figure 3.6: The noise model.

Example Scene

As an example, a car model with 20k triangles was scanned with a 100x100 grid. On a computer with a Pentium 4 3GHz processor with 2GB of RAM, the scan took 0.6 seconds. Figure 3.7 shows the model and the resulting synthetic scan.

3.2 Recoloring a LiDAR Scan from an External Image

Some laser scanners come with an internal color camera inside the device. This camera serves to make it easier for surveyors to align their scans in the field, as well as recall the data that they captured during later review. Unfortunately, typically these cameras are quite low quality. Worse, they are not accurately registered to the scan points. Even if the manufacturer promises that the color image is registered to the



(a) A 3D model of a car. (b) A synthetic scan of the car model.

Figure 3.7: A car model and the resulting synthetic scan.

camera, the registration accuracy may not be as accurate as necessary. Additionally, by moving the scanner around to perform scans, it will inevitably be rattled and shaken enough for the camera to become misaligned.

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 registering color images to point clouds automatically [68, 112, 69, 4, 41, 1], 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' 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.

GUI

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 3.8 shows a screenshot of the correspondence selection interface.

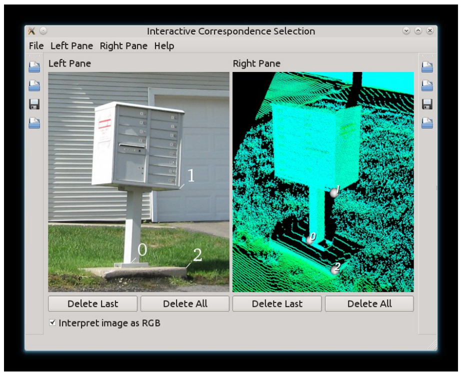


Figure 3.8: Screenshot of the correspondence selection GUI. Three corresponding points have been selected in an image and a point cloud. The point cloud is pseudo-colored by the intensity of the returns.

In the example you will notice that 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 projection matrix between the data sets so that we can 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 were all colored with the same color, it is very hard to identify such correspondences.

3.2.2 Resectioning

Clearly, the external camera cannot be placed exactly at the source of the laser, so it will 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 projection matrix between the selected correspondences in 3D points and the externally obtained image. We assume a linear model, that is, a mapping from the scene points X to the image points x via a $3x4$ projection matrix P, as shown in Equation 3.1. The basic equations are introduced here, but a full discussion is available in [75].

Obtaining this external image and mapping the colors onto the scan points simultaneously fixes two problems - the exposure problem and the alignment problem.

$$x = PX \tag{3.1}$$

If we write this in a little bit more verbose notation:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} P_{1,1} & P_{1,2} & P_{1,3} & P_{1,4} \\ P_{2,1} & P_{2,2} & P_{2,3} & P_{2,4} \\ P_{3,1} & P_{3,2} & P_{3,3} & P_{3,4} \end{pmatrix} X \tag{3.2}$$

Dividing the right hand side by its last entry:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{(P_{1,1}P_{1,2}P_{1,3}P_{1,4})X}{(P_{3,1}P_{3,2}P_{3,3}P_{3,4})X} \\ \frac{(P_{2,1}P_{2,2}P_{2,3}P_{2,4})X}{(P_{3,1}P_{3,2}P_{3,3}P_{3,4})X} \end{pmatrix} \tag{3.3}$$

we see that each correspondence generates two equations. For the x coordinate:

$$x = \frac{(P_{1,1}P_{1,2}P_{1,3}P_{1,4})X}{(P_{3,1}P_{3,2}P_{3,3}P_{3,4})X} \tag{3.4}$$

and for the y coordinate:

$$y = \frac{(P_{2,1}P_{2,2}P_{2,3}P_{2,4})X}{(P_{3,1}P_{3,2}P_{3,3}P_{3,4})X} \tag{3.5}$$

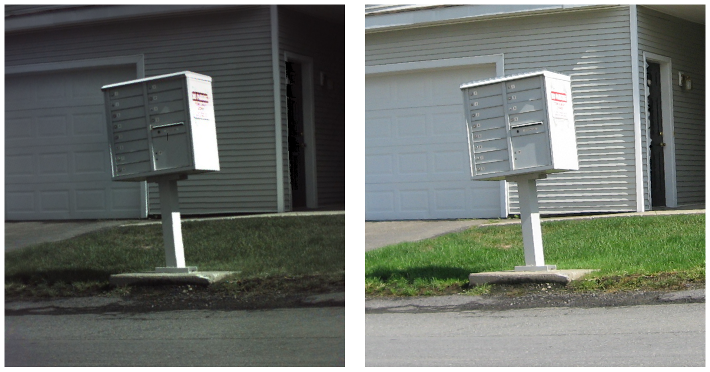
To solve for P, we need at least 6 correspondences (since there are 12 entries in P and each correspondence generates 2 equations). However, in practice, with real world data sets of the accuracy we are used to seeing with laser scanners, around 10 correspondences are necessary to obtain a valid result. 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:

$$AP_{vec} = \begin{pmatrix} X_x & X_y & X_z & 1 & 0 & 0 & 0 & 0 & -xX_x & -xX_y & -xX_z & -1 \\ 0 & 0 & 0 & 0 & X_x & X_y & X_z & 1 & -yX_x & -yX_y & -yX_z & -1 \end{pmatrix} \begin{pmatrix} P_{1,1} \\ P_{1,2} \\ P_{1,3} \\ P_{1,4} \\ P_{2,1} \\ P_{2,2} \\ P_{2,3} \\ P_{2,4} \\ P_{3,1} \\ P_{3,2} \\ P_{3,3} \\ P_{3,4} \end{pmatrix} = 0 \tag{3.6}$$

The two rows of A generated by each correspondence are stacked to form at least a $12x12$ matrix. A well known result from linear algebra is that the solution to an equation of this form is the singular vector corresponding to the smallest singular value. After performing the singular value decomposition of A, $A = UDV^T$, 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.

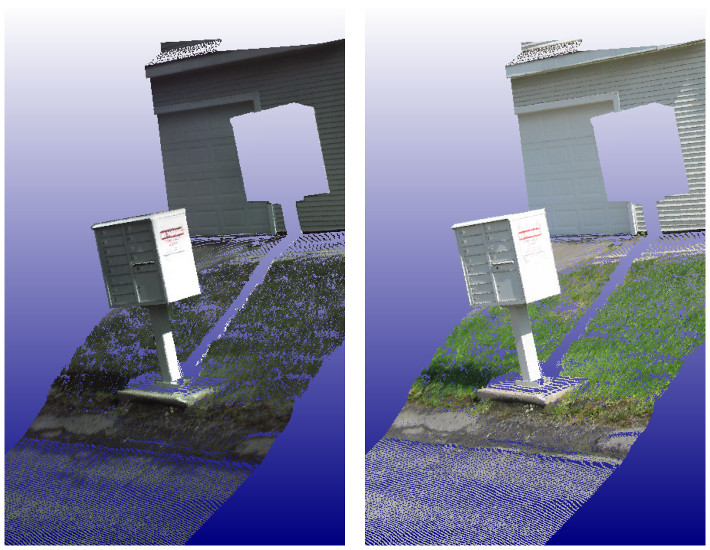
Though this is correct from a mathematical perspective, it is strongly suggested to use the "normalized DLT". In this method, the exact same procedure is performed, but instead of using the image points x_i and the 3D points X_i directly, each set of points is first normalized so that it has zero mean the average distance of the points to the origin $\sqrt{2}$.

This solution can be used as an initialization for a non-linear solution, but in our experiments the linear solution was very good and it was unnecessary to refine the solution.



(a) Poor image quality (b) Corrected colors.

Figure 3.9: Recoloring the scanner image from a digital camera image.

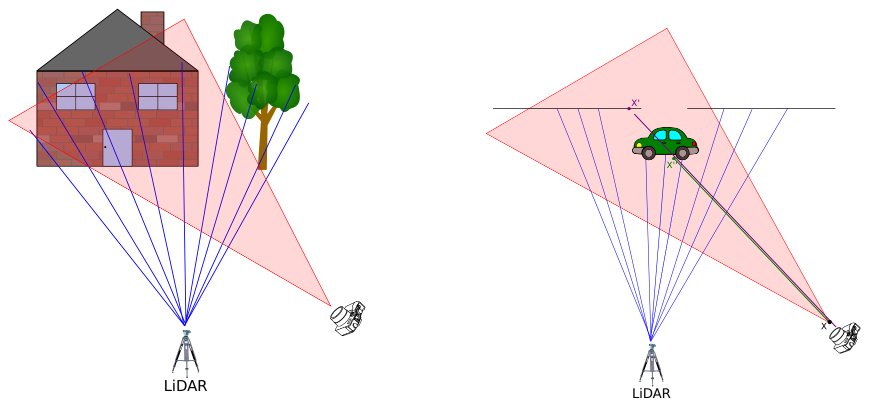


(a) (b)

Figure 3.10: Bad alignment, and corrected alignment of color and range images. a) Poor quality image, and poor alignment of the color image to the scan points. Note the white pixels from the side of the mailbox that appear in the grass. b) Scan points colored by the external image after projecting.

3.2.2.1 Recoloring Scan Points

Once we have P, the projection matrix which maps the 3D points to the 2D points, we can project all of the 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. For many points, this is not the case - this happens when the scanner sees parts of the scene that are not seen by the camera, as shown in Figure 3.11a.



(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.

Figure 3.11: Resectioning problems.

Another case which requires special attention are points that were visible to the scanner but not to the camera. As shown in Figure 3.11b, some image pixels have multiple 3D points which project to them. To compensate for this occlusion, instead of directly projecting a 3D point into the image, we can intersect the line segment from the 3D point to the camera location with a mesh of the scan. If there is no intersection, the point should be projected and take on the color of the image pixel. If there is an intersection, the point has no way to obtain new information from the image, so its color should remain unchanged.

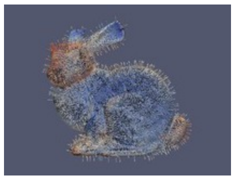
3.3 Other Tools

The synthetic LiDAR scanner and correspondence/resectioning tools were used directly 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 in various journals and magazines. Their citations are provided in line.

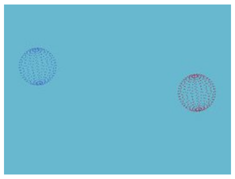
A Greedy Patch-Based Inpainting Framework: A flexible platform for testing patch based inpainting algorithms which allows easy replacement of the patch difference function and the priority function. [44] *Best Submission of the Year 2011 award*

Poisson Editing: An implementation of the hole filling and cloning techniques described in [123]. ([56, 45] *Best Submission of the Year 2011 award*)

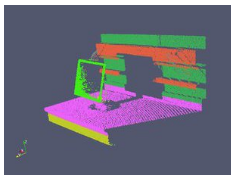
Criminisi Inpainting: An implementation of the popular inpainting algorithm described in [39, 55]



Point set processing: We have written several operations on point sets including outlier removal, curvature estimation, normal estimation, and normal orientation. [46]



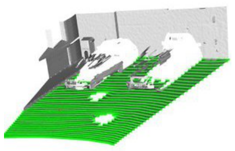
Clustering segmentation: An iterative clustering procedure based on a Radially Bounded Nearest Neighbor search. [54]



Hough Plane Detector: Find planes in 3D point cloud data. [57]

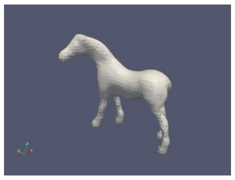
K-Means Clustering: K-Means clustering is an excellent technique for clustering points when the number of clusters is known. We have implemented the algorithm along with the K-Means++ initialization method which finds the global optimum much more frequently than a naive/random initialization. [48]

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. [58]

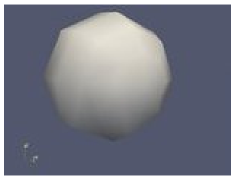
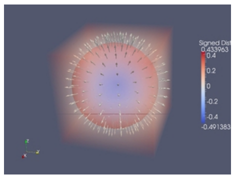
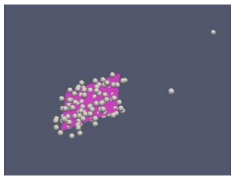
	**Stratified Mesh Sampling**: A Paraview plugin for stratified mesh sampling. Stratified sampling is a technique to uniformly resample a mesh. [52]
 Signed Dist 0.433963 0.4 0.2 0 -0.2 -0.4 -0.491383	**Point Set Surface Reconstruction**: We have implemented a basic algorithm to produce a mesh from a 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]

Table 3.1: Other LiDAR tools developed during the course of this thesis.

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

Object registration (described in 2.2.1), or aligning one object with another, is an extremely important step in many common 3D data processing algorithms.

In problems including object detection, object recognition, and 3D model construction, a common final step is aligning a 3D model with a 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 [59].

A common choice of registration algorithm is Iterative Closest Points (ICP) [170]. ICP directly minimizes a cost function describing the difference between the two objects at their currently 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 [131]. 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.

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 [109], [66]. 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 4.1.

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

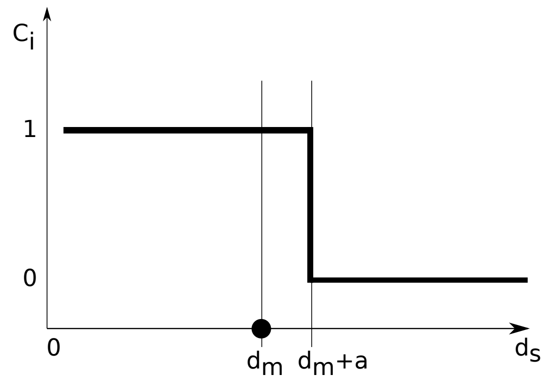


Figure 4.1: Diagram of consistency function

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 4.2 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.

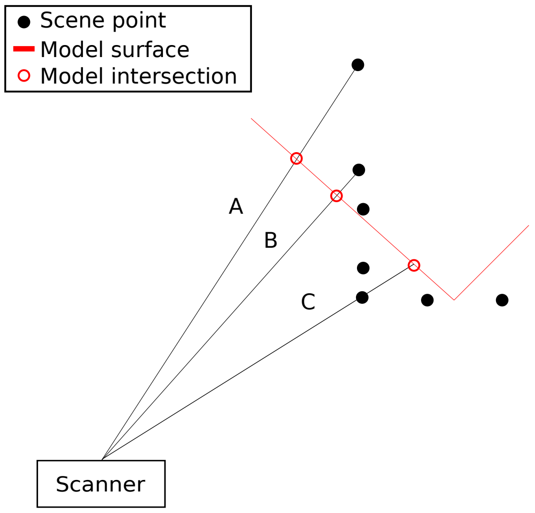


Figure 4.2: Consistency example for 3 rays.

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 paper. 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 [117]), the information content at each point could be related to the quality of a planar fit, with more locally complex regions containing more information. [106] 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 paper, we set $\sigma = 0.01m_v$.

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 [20] and compute the update for only those points. Figure 4.3 illustrates an example of the information observation process, showing the influence of one scene point on three model points.

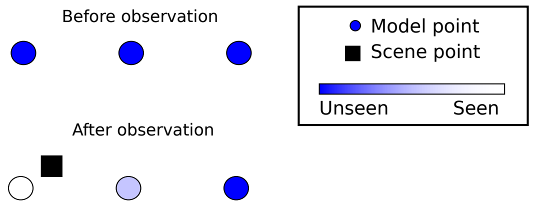


Figure 4.3: Confidence Example

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.

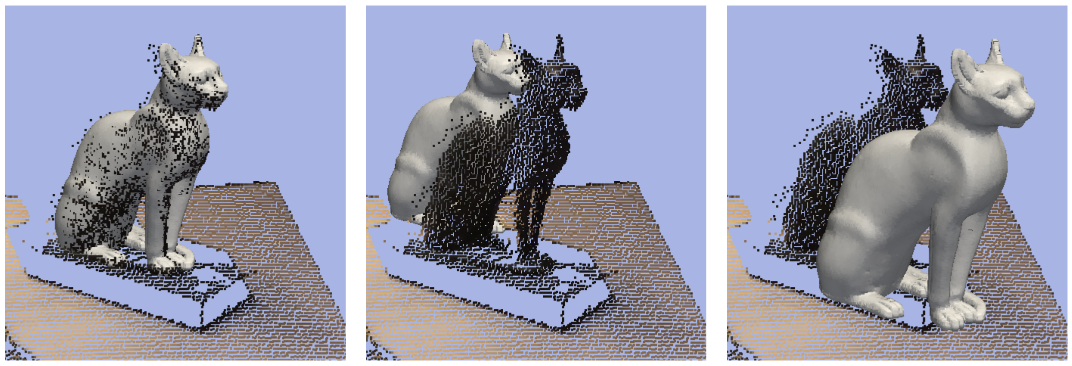
4.3 Experimental Results

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.

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 4.4a). 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 $\sigma = 0.5$cm for the confidence calculations.

We then placed the model behind the correct position, i.e., in the "shadow" of the LiDAR scan (Figure 4.4b). 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 4.4c). 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 4.4: Cat sculpture in varying positions.

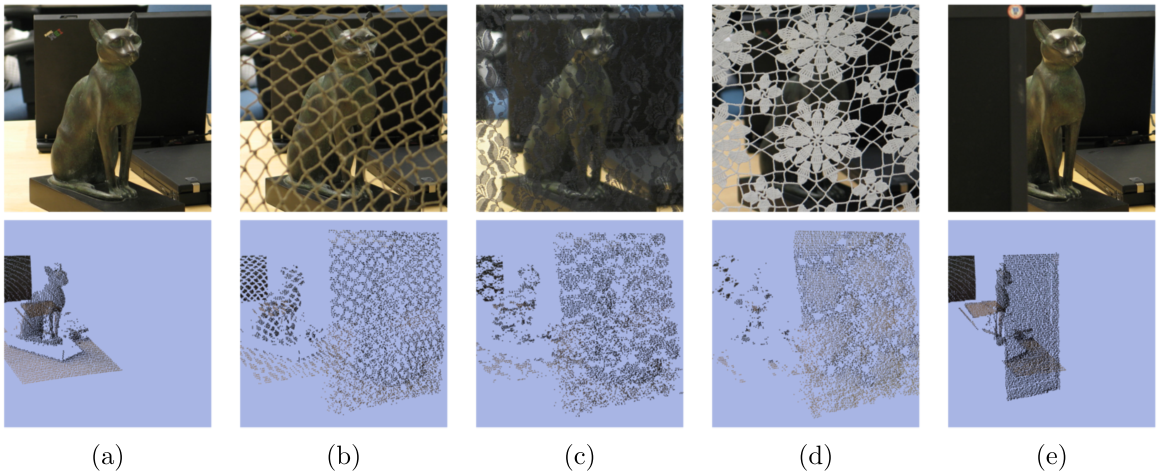


Figure 4.5: Cat sculpture scans with varying occlusions. (a) No occlusion, (b) Light, sporadic occlusion (net), (c) Heavy, sporadic occlusion (lace), (d) Heavy, sporadic occlusion (tablecloth), (e) Heavy, contiguous occlusion (monitor)

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 4.4.

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 4.5 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 4.5a 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 4.5b, we scanned the scene through a net to imitate a scaled down camouflage net. The confidence of the model decreases 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 4.5c, 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 4.5d, we occluded the cat sculpture with a tablecloth. The results are similar to the lace fabric. Finally, in Figure 4.5e, 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 4.5b.

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.

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. 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 4.6, 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 4.6 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.

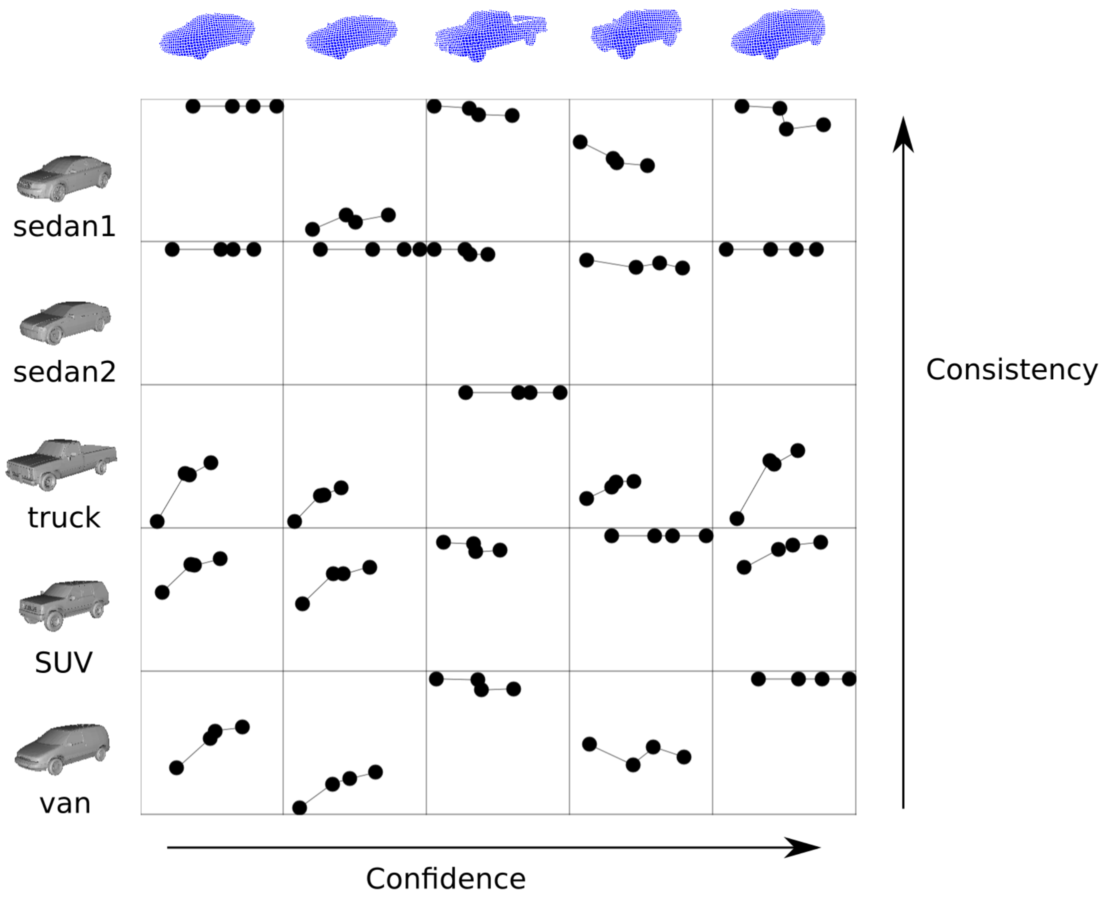


Figure 4.6: Confidence/Consistency evaluation between all combinations of five automobile models. Rows: models, columns: LiDAR scans. Each dot (left to right) represents an additional scan taken from the front, driver side, rear, and passenger side viewpoints, respectively.

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.

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.

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.

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 4.7a and 4.7b. 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 (4.6).

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

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 [131]. 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 [113]). 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.

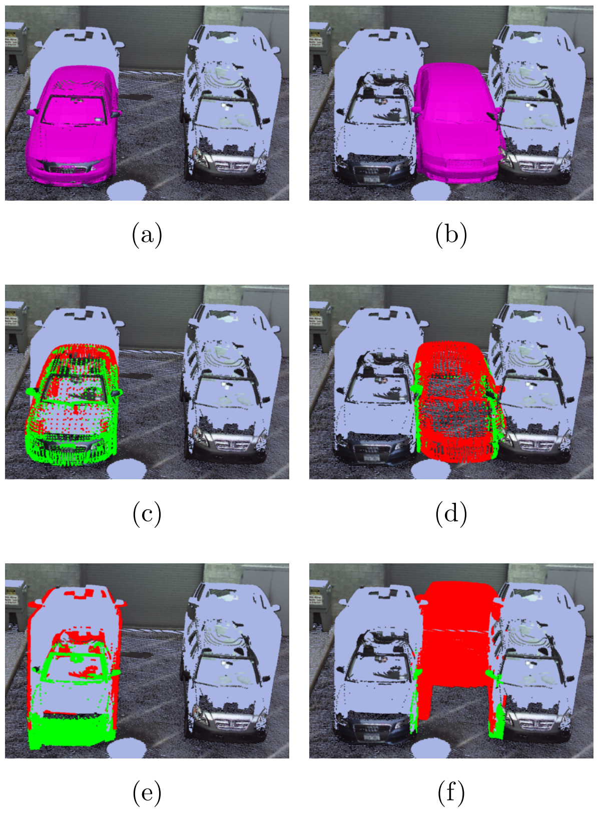
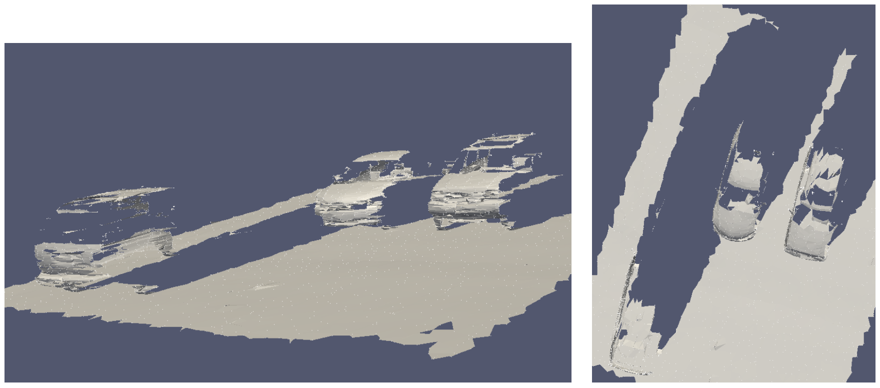


Figure 4.7: Parking lot demonstration. (a) Model registered to correct position in scene. (b) Model registered to incorrect position in scene. (c) Model points at correct position colored by confidence (unseen: red, seen: green). (d) Model points at incorrect position colored by confidence (unseen: red, 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). This figure is best viewed in color.

In Figures 4.7c and 4.7d, 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 4.7e and 4.7f, 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 the consistency and confidence measures play a useful dual role for understanding if a hypothesized position makes physical sense.

Figure 4.8 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 4.8: Parking lot scene with three cars.

Figure 4.9a 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 4.9b 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 4.9c, 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.

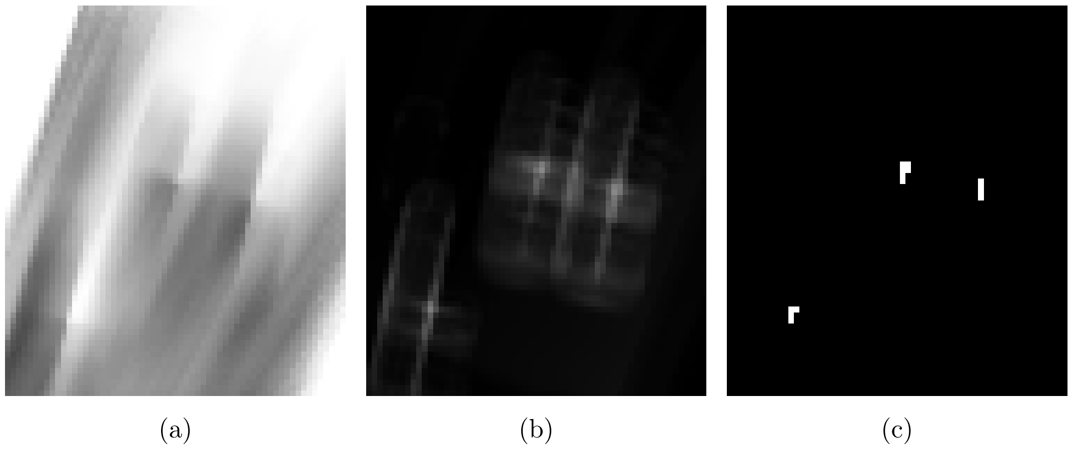


Figure 4.9: Consistency/confidence heat maps. (a) Consistency heat map (b) Confidence heat map (c) Dual thresholded with consistency $>$ 0.75 and confidence $>$ 0.3.

Demonstration: Sliding a Window Along a Building

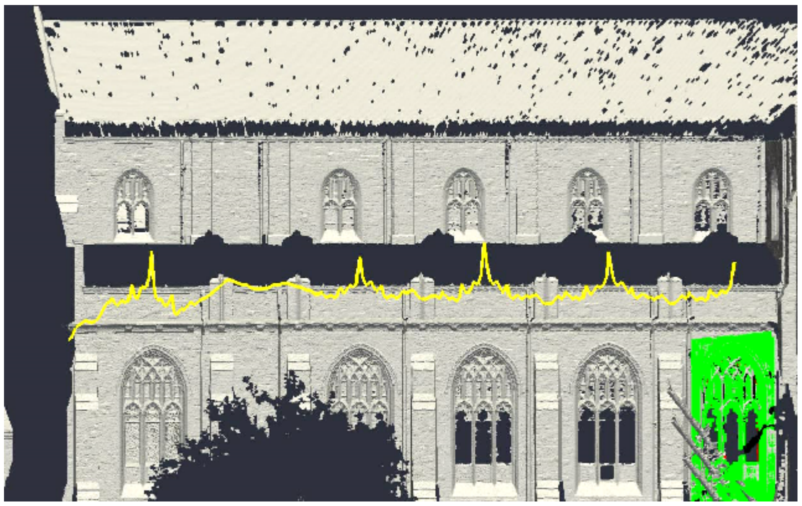


Figure 4.10: Confidence that the window is at a continuous position, sliding parallel to the building facade

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 4.8 on a Pentium 4, 3GHz computer with 2GB of RAM. This could be improved by using a lower resolution model (our car model is $\approx$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 plan to 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 comparison 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 [102] 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 4.5, 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 clicks. A two-step algorithm is introduced which operates on a combined color-and-depth image which first takes advantage of sharp depth discontinuities present at some points of an object, and uses this initial result to perform a second segmentation utilizing the available color information.

5.1 Traditional Image Segmentation

An extremely hot topic in computer vision in recent years is "graph-cut" techniques [29, 28, 26].

In the language of graph theory, 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 exactly 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 to the set of edges E between each vertex corresponding to a pixel and to all of its neighbors. The "neighborhood connectivity" 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 an edge to E between every vertex corresponding to an image pixel and both F and B. These edges are referred to as “t-edges”, indicating that they connect a node to one of the terminal nodes. Recall the resulting graph shown in Figure 5.1.

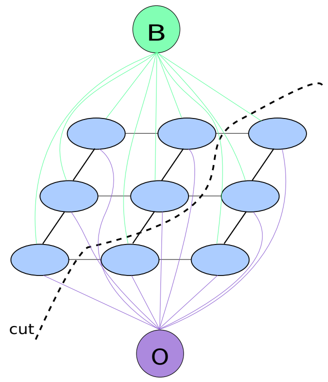


Figure 5.1: The graph constructed from an image. Each pixel is connected to its neighbors as well as to a foreground and background node. A cut divides the foreground and background nodes by disconnecting edges.

Every edge in E is assigned a “weight”, w, which indicates the similarity between the two vertices it connects. That is, w is small if the vertices are dissimilar and large otherwise. In graphs on images, the weights on n-edges is typically computed to be some function of the RGB values at the corresponding image pixels. These n-edges contribute to the “binary term” of an energy function (which we construct momentarily), indicating that two of the vertices play a role. This function is often simply an exponential. That is, the weight between two vertices i and j is taken to be

$$w_{i,j} = \exp\left(-\|I_i - I_j\|^2\right) \tag{5.1}$$

Here, $|I_i - I_j|$ is the “distance” between the color values at pixel i and j. The way it is written here, it indicates the squared norm of the difference between corresponding channels of the RGB image, as shown in Equation 5.2

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

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 [86]), or any other vector distance function (the L1-norm, for example).

The t-edge weights contribute to the "unary term" of the energy function, as they are only connected to one vertex corresponding to an image pixel. Because of this, we need some way to compute a weight from only one RGB value. Typically, a probability density function is constructed indicating the likelihood that this pixel belongs to the foreground or to the background.

A "cut" on a graph is defined as a subset of edges of E such that if we disconnect the vertices connected by these edges there is no longer a "path" from F to B. We can see now that this separation is exactly what we are looking to obtain - the resulting image segmentation is simply that the foreground or object consists of all nodes that are still connected to F while the background consists of all the nodes that are still connected to B. There is a huge set of possible cuts on this graph, but the segmentation we are looking for is the "minimum cut". That is, the cut that cuts edges that have the lowest weight sum, which we also call the "energy", E, of the cut. That is,

$$E = \sum_{i \in E} w_i \tag{5.3}$$

Typically an interface is provided to allow a user to "scribble" on the image (shown in Figure 5.2), indicating some hard constraints on the problem. That is, some pixels are very easy to mark as belonging to the object of interest (pixels in the interior of the object). These pixels are often referred to as *seeds*. Likewise, pixels far away from the object the user can very casually mark as "definitely background". These constraints are very important - without them the resulting energy minimization problem is, though well-posed, a meaningless result.

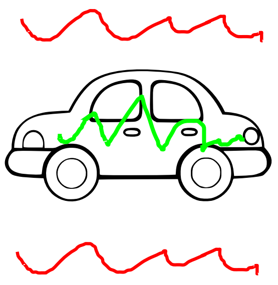


Figure 5.2: An image with user drawn "scribbles" on the foreground (green) and background (red).

For a pixel indicated by the user to be foreground, the corresponding weight on 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 so that this edge can never be cut. User-marked background pixels and handled exactly oppositely.

The users scribbles also provide a nice set of pixels to create the function to compute all remaining t-weights. Often the pixels in each of these foreground and background scribble sets are used to construct a probability density function of the RGB values of the object and background. That is, for each vertex i,

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

$$w_{i,B} = -\lambda \log P_F(I_i) \tag{5.5}$$

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. The inter-node weights are computed using a simple similarity measure

This whole procedure is often performed in an interactive way. That is, if the segmentation is determined to be incorrect or need refinement, the foreground and background scribbles can be improved or extended and the solution recomputed.

In RGB image segmentation, the boundary term (responsible for encoding the attraction between neighboring pixels) is taken to be:

$$B_{p,q} \propto \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right)\frac{1}{dist(p,q)} \tag{5.6}$$

where p and q are neighboring pixels, I_x is the image value at pixel x, and $dist(p,q)$ is the physical distance, in image coordinate units) between pixel p and q. Because p and q are always neighbors in the 4-neighborhood formulation of the problem, was can set $dist(p,q) = 1$, obtaining

$$B_{p,q} \propto \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right) \tag{5.7}$$

5.2 LiDAR Image Segmentation

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 single image.

Our scan points are already provided in the form of a spherical grid, due to the process and ordering of their acquisition. We construct a *depth image* from the LiDAR points by computing their distance to the scanners location, and placing this value in a float-valued image of the same size as the acquisition grid. This process is shown in Figure 5.3.

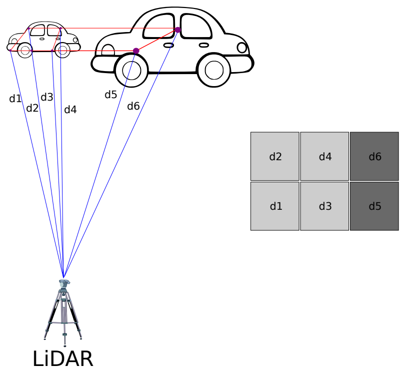


Figure 5.3: Constructing a depth image.

We then append this 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).

Alternatively, we can represent the LiDAR data as a 6-channel image, where the last 3 channel appended to the RGB channels are simply the X, Y, and Z coordinates of the 3D points. This forms an image of type $RGBXYZ$. Throughout this thesis, we have chosen to use the $RGBD$ representation.

Difference Between LiDAR Pixels

A critical step in the formulation of the graph-cut problem is determining the distance function between vertices. As we mentioned earlier, in pure RGB images, a reasonable weighting is to set

$$\lambda_r = \lambda_g = \lambda_b = 1 \tag{5.8}$$

meaning we value differences in each color channel equally. However, it certainly does not make sense to set

$$\lambda_r = \lambda_g = \lambda_b = \lambda_d = 1 \tag{5.9}$$

as 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 approximate 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.

A the most general extension to Equation 5.2 with these new types of images is

$$I_p - I_q = \lambda_r(|P_r - Q_r| + \lambda_g|P_g - Q_g| + \lambda_b|P_b - Q_b|) + \lambda_d(|P_d - Q_d|) \tag{5.10}$$

where each λ_c is a scalar weight for channel c of the image.

One method of setting these λ's is to *standardize* the channels, a common practice in statistical data analysis. To standardize the channels, we find the mean and standard deviation of each of each component over the entire image. Then we subtract the corresponding means and standard deviations from the pixels to obtain a new 4D image in which pixels can be directly compared, and their differences are related to the actual difference we would expect to see.

While this is one technique, it assumes that we wanted to evenly weight the different components. In some applications, we might know a-priori that depth differences are extremely bad, so we might want to make a small difference in the depth channel lead to a very large pixel difference value. In other applications, we might have some knowledge about the colors in the image which encourages us to weight one color channel different from others.

The solution we have chosen to use throughout this chapter is to weight the collective color channels equally with the depth channel. That is, after standardization, we use

$$\lambda_r = \frac{1}{3}, \lambda_g = \frac{1}{3}, \lambda_b = \frac{1}{3}, \lambda_d = 1 \tag{5.11}$$

In the best case, we could use machine learning to pick a set of weights that is known to 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.

If we had chosen the $RGBXYZ$ representation, we could select weights for the RGB channels, and use the Euclidean distance between the XYZ channels, representing the actual distance in space between the points corresponding to the two pixels.

5.3 Contribution: Two Step LiDAR Segmentation

Unfortunately, after much experimentation with weighting the color and depth channels, we were unable to find a combinations of weights that worked well across multiple data sets. During our experimentation, however, we made an interesting realization. When the depth channel weight was very high relative to the *RGB* channel weights, we always obtained a resulting segmentation that was a strict subset of the object. That is, though we did not label all of the object points as foreground, we never labeled background points as foreground.

We motivate this technique with a simple real-world example. Consider the scene shown in Figure 5.4.

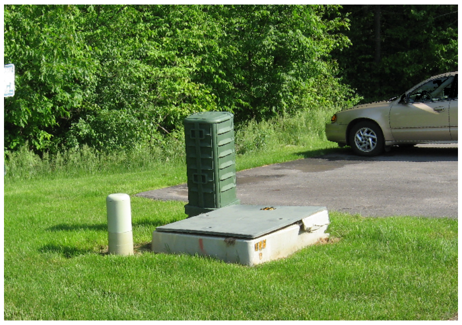


Figure 5.4: An electric box in a grassy field.

After the user marks simple foreground and background strokes as shown in Figure 5.5, using a depth-only segmentation produces the result in Figure 5.6.

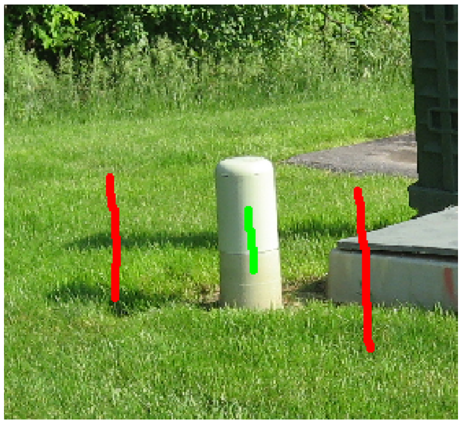


Figure 5.5: The user specified foreground and background pixels in the form of rough strokes.

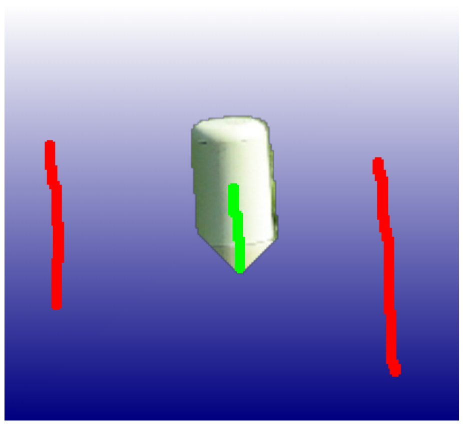


Figure 5.6: The resulting segmentation using depth information only.

We see that the resulting pixels are all from the object, but not all object pixels are contained in this set. This result makes sense, as by looking at a depth image, we can observe that parts of the object that are far away from the ground are very obvious to determine the boundary of the object from (by inspection), but as we move towards parts of the object that are near the ground, the boundary becomes much harder, then nearly impossible to discern. This is shown in Figure 5.7

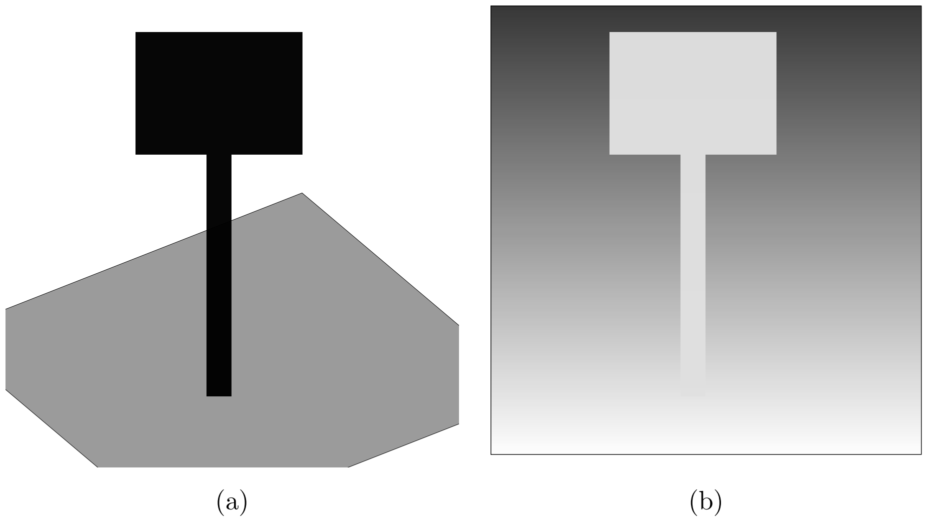


Figure 5.7: A demonstration of the regions in which it is easy to determine an object boundary in a depth image. a) A sketch of a scene of a sign attached to the ground. b) A depth image of the scene. Observe that it is very easy to visually separate the sign from the scene near the top of the image, but as we approach the point where the sign joins the ground, it is very hard to distinguish.

These "definitely foreground" pixels are now used as a much stronger initialization to a second segmentation, as shown in Figure 5.8. This time, very small weights are applied to the depth channel, mainly utilizing the color information in the image. By using these foreground pixels it is effectively as if the user, rather than scribbling very casually, had instead painstakingly very carefully filled in the object at a pixel by pixel level.

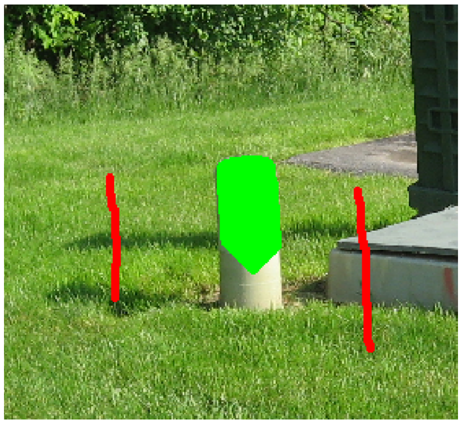


Figure 5.8: The new foreground pixels resulting from the depth segmentation.

5.3.1 Background Pixel Inference

While the two step method described in 5.3 clearly performs better than any single-step segmentation we were able to obtain, it is not perfect. In some cases, the color segmentation step “bleeds” over a hard depth boundary, which we should certainly be able to avoid. The method of using the under-segmentation based on depth values alone as seeds for the foreground is a step in the right direction, but we have not yet added an constraints to the background seeds.

Our goal is to allow the color segmentation to be used only where the depth segmentation is uncertain. Since we have obtained a very accurate object labeling near many parts of the object boundary, we can use this to our advantage. By dilating the region that we have marked as foreground in the initial depth segmentation step and then extracting its boundary, we obtain known background pixels. By labeling these pixels as background seed pixels, we prevent the color segmentation step from leaking into this region (which would have otherwise occurred in many cases, such as in the green electric box with the green grass in the background. The boundary of the dilated region is shown in Figure 5.9.

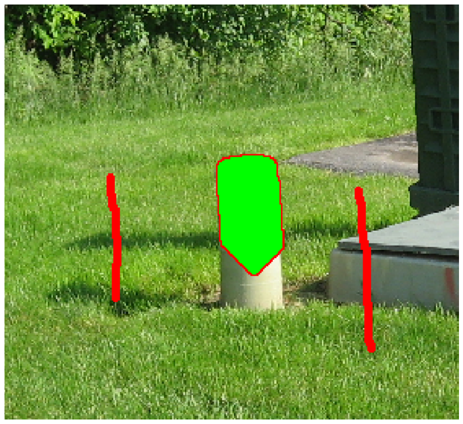


Figure 5.9: Naive boundary of depth segmentation.

However, not all of these dilated region boundary pixels belong to the background. If they did, we would have already obtained a perfect segmentation! To prevent the pixels that are in the uncertain region from erroneously being labeled as background, we perform a test at every proposed background pixel. Namely, we compute the depth difference from each proposed background pixel to its neighboring foreground pixels. If the depth is above a threshold, we are confident that this actually a background pixel - referring to our assumption about how easy it is to manually recognize the object boundary “far away” from the ground. If the depth difference is not larger than the threshold, the pixel is not marked as a background seed. The resulting background pixels are shown in Figure 5.10.

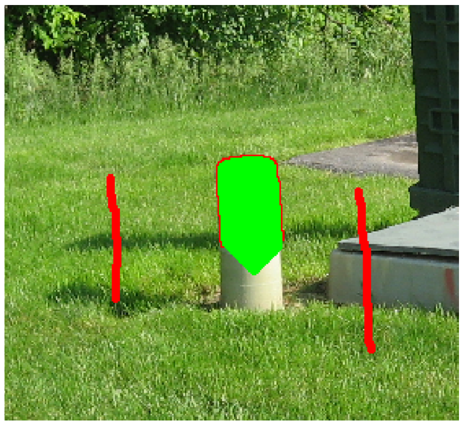
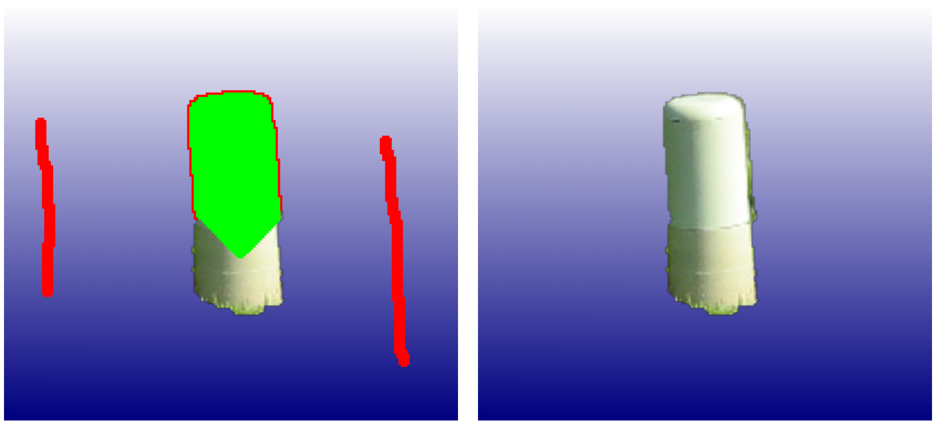


Figure 5.10: Background pixels marked after background inference.

5.4 Experiments

In this section, we apply our proposed segmentation algorithm on several real-world data sets. In each example, the minimal user strokes, the result of the depth-only segmentation, the computed background pixels, and the final segmentation are shown.

Continuing the example from previous sections, the electric box is correctly segmented from the scene, as shown in Figure 5.11.



(a) The final strokes on the electric box. (b) The final extracted electric box.

Figure 5.11: The final segmenting of an electric box.

Placeholder of mailbox segmentation

Figure 5.12: Segmentation of a mailbox.

Placeholder of trashcan segmentation

Figure 5.13: Segmentation of a trashcan.

CHAPTER 6
LiDAR Inpainting

There has recently been a lot of work on the image completion problem in standard RGB images. Image editing tools like Photoshop now routinely include methods to seamlessly remove objects from an image, as well as repair artifacts. In this chapter, we introduce an algorithm to segment an entire object from a LiDAR scan. The proposed method does not require a training database, which allows it to be applied to a very broad range of objects. Our work draws from and extends the exemplar-based image completion field. The goal of our method is to fill holes left by removing objects from the scene. Because of the technique we use, we can fill in the scene behind an object without removing it using the same technique. This makes the data set much more useful to the user, as it can now be viewed from any angle and not have large chunks of missing data caused by the LiDAR shadow effect.

In this chapter, we first detail the methods which we build upon. We then go on to introduce the algorithm and show real-world examples of filling texture and structure in LiDAR scans.

6.1 Patch Based Image Inpainting

Before proceeding, we must define some terms. The **hole**, or **target region**, is the portion of the image which is not known, and we wish to fill. The **source region** is the portion of the image which is known (is not part of the hole) at the beginning, or has been already filled. A **target patch** is any patch whose center is on the hole boundary. The collection of all target patches is the set from which to choose the patch to fill at each iteration. A **source patch** is a patch entirely composed of pixels from the source region.

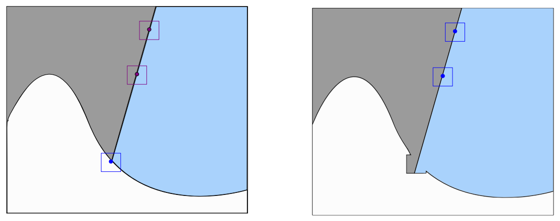


Figure 6.1: A conceptual demonstration of patch based inpainting. a) Image to be filled and potential source patches. b) The target patch properly filled.

Target patches have two distinct regions, the region that overlaps the hole (which we call the "hole region"), and the region that overlaps the source region, which we call the "valid region"). Although source patches have only one region (all pixels are "valid"), 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.

Additionally, we can refer to a *target pixel* rather than the *target patch*, but this pixel is simply the center pixel of it's corresponding target patch.

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 consists of zero pixels.

There are two major components of an exemplar-based inpainting algorithm:

- How do we choose which boundary pixel to fill? There has been a lot of effort dedicated to determine a good order in which to fill target patches. Namely, [39] notes the importance of filling patches that continue linear structures first. The argument is that these linear structures are critical to human interpretation of the resulting image, and we have confirmed this experimentally. 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.

- How do we decide which source patch to copy into a specified target patch? Once a target pixel is selected, we must find the "best" source patch to copy to its location. To compute the difference between a source patch and a target patch, many researchers compute the sum of the differences of corresponding valid pixels. As we discuss in detail in Chapter 7, this does not always prove as discriminative as is needed to always produce accurate matches.

6.1.1 Filling Priority

Since most exemplar-based inpainting methods are greedy, selecting a good order in which to fill the hole is very important. A few techniques for selecting a patch to fill are described here.

- Random

  The simplest approach is to simply choose a random pixel on the current hole boundary. This is fast and easy, but likely not a very good choice.

- Onion-Peel

  Filling in pieces near the edge of the hole should intuitively be easier than filling in pieces deep within the hole. Therefore, boundary pixels near the outside of the original hole should be preferred over boundary pixels that are now deep inside of the original hole. This technique gets its name because with a regularly shaped hole, the algorithm will chew away at the outside of the hole, all the way around the hole, before moving further inside the hole, an order which resembles peeling back the layers of an onion. To enforce this behavior, a confidence image is maintained. Initially, the confidence outside of the whole is 1 (very sure) and the confidence inside of the hole is 0 (totally unsure). You can think of confidence as a measure of the amount of reliable information surrounding the pixel. To compute the confidence at a particular target pixel,

- Linear Structure Preserving

  Popularized in [39], pixels where the *isophotes* of an image are strong and aligned with the boundary normal are preferred. Isophotes are simply the image gradient rotated by 90 degrees, indicating the direction of least increase rather than greatest increase. These vectors indicate the direction of linear structures in the image. Unfortunately, in real images with natural textures, the gradient is extremely noisy. That is, we would like to think that the boundary between say grass and a sidewalk has very strong isophotes indicating the direction of this boundary, but do to local shading, noise, and other phenomena, these vectors are extremely unreliable.

6.1.2 Algorithm Demonstrations

Algorithm Synthetic Demonstration

Figure 6.2 shows a synthetic demonstration of a patch-based inpainting algorithm. The image consists of a black region (top) and a gray region (bottom). This simple example is used for testing because we know the result to expect - the dividing line between the black region and gray region should be continued smoothly.

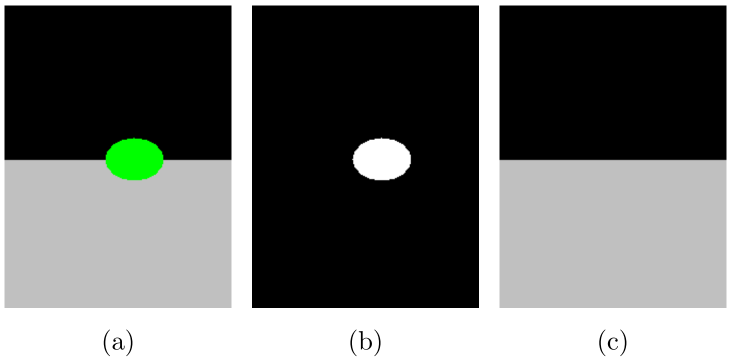
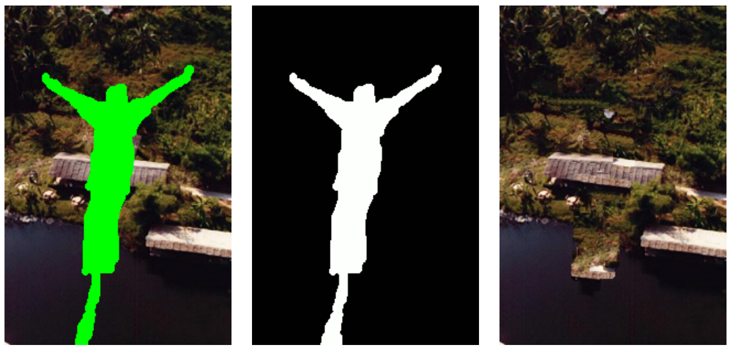


Figure 6.2: A synthetic demonstration of patch based inpainting. a) Image to be filled. The region to be filled is shown in bright green. b) The mask of the region to inpaint. c) The result of the inpainting.

Algorithm Realistic Demonstration

In real images, the result will never be as "good" as in Section 6.1.2. Below we show an example of completing a real image. This result shows the typical quality of inpainting that the algorithm produces.

Figure 6.3 shows a real example of the algorithm. 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. (The patch radius, or half-width, is always specified rather than the side length because this guarantees the patches to have odd size, which ensures they have a well defined center pixel. The patch side length is simply $l = 2r + 1$).



(a) (b) (c)

Figure 6.3: Realistic demonstration of exemplar-based inpainting. a) Image to be filled. The region to be filled is shown in bright green. b) The mask of the region to inpaint. c) The result of the inpainting.

Notice that although the resulting image looks feasible on its own, when we look at Figure 6.3a, the completion we would expect has the shoreline completed much more directly. We show the implications of this problem on our work to inpaint holes in LiDAR data in Section 6.7.

Of course, there are cases where we cannot expect the algorithm to compute a reasonable completion. For example, in Figure 6.4, there algorithm does not know to "stop" completing the stop sign post, so we cannot predict what will happen when the completion gets to the post/ground intersection. This partially depends on the fill order, but mostly depends on random chance.

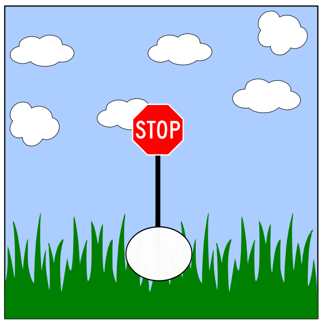


Figure 6.4: A case where we do not expect a good result.

The way we interpret this condition is that the algorithm must be able to "join" a structure on one side of the hole to the same structure on the other side. In this case, we could expect the ground/sky boundary to be successfully completed (barring the post "getting in the way"), but as there is no corresponding post on the bottom of the hole, we would not expect a good result in this case.

Implementation Caveats

Gradient Computation on an Image with a Hole

There is a small "gotcha" when computing the gradient of an image near a hole. Typically the gradient of an image is computed by convolving a gradient kernel with the image. However, if we compute the gradient of an image with a hole in this fashion, the gradient computed produces the result shown in Figure 6.5.

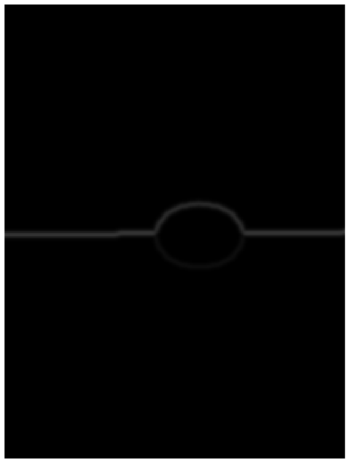


Figure 6.5: Result of naively computing the image gradient. The gradient magnitude is shown here.

The high values of the gradient magnitude surrounding the target region are very troublesome. The resulting gradient magnitude image using this technique is sensitive to the choice of the pixel values in the target region, which we actually want to be a completely arbitrary choice, as it should not affect anything. More importantly, the gradient plays a large part in the computation of the pixel priorities, and this computation is greatly disturbed by these erroneous values. Simply ignoring these boundary isophotes is not an option because the isophotes on the boundary are exactly the ones that are used in the computation of the priority.

The erroneous values of the gradient magnitude surrounding the target region are very troublesome. The resulting gradient magnitude image using this technique is sensitive to the choice of the pixel values in the target region, which we actually want to be a completely arbitrary choice (as, since it is unknown, it should not affect anything). More importantly, the gradient plays a large part in the computation of the pixel priorities, and this computation is greatly disturbed by these erroneous values. Simply ignoring these boundary isophotes is not an option because the isophotes on the boundary are exactly the ones that are used in the computation of the Data term. To fix this problem, we have examined three very different solutions.

1. Immediately dilate the mask specified by the user. This allows us to compute the isophotes 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. Unfortunately, in doing this we have made the problem slightly harder by making the hole slightly bigger. This is not a big deal, but it is not desirable.

2. Before performing the patch-based inpainting, use a non-patch based technique to coarsely fill the hole (e.g. Poisson editing/hole filling). As we discussed in Section 2.3.1.1, these solutions are not acceptable to fill large holes, but they are reasonable near the hole boundary. By doing this, we have produced values inside the boundary of the hole that when used in a gradient computation yield a gradient image that is very close to what we would expect.

3. Use an adaptive gradient kernel. Typically the same kernel is applied to every pixel in the image to compute the gradient. Typically a "central difference" gradient is used, which requires information to be available in all directions from the pixel at which the gradient is currently being computed. The central difference provides the most accurate result, but we can fall back to single sided difference kernels (forward difference or backward difference) when all of the pixels around the center pixel are not available, as is the case in an image with a hole. We adaptively re-weight the values in the kernel so that they approximate the gradient operation using only the valid pixels in the

neighborhood.

Figure 6.6 shows the result of fixing this problem.

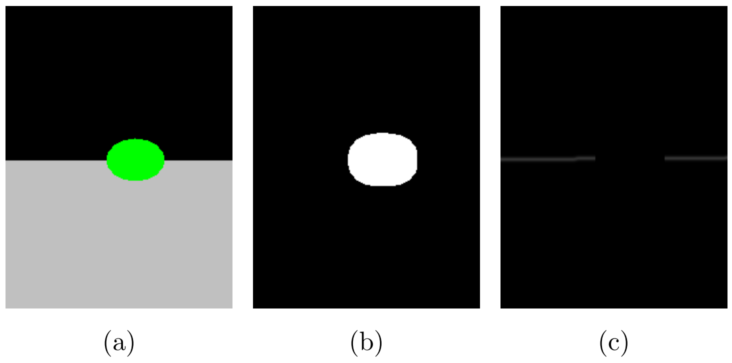


Figure 6.6: Procedure for fixing the erroneous gradient problem. a) Image to be filled. The target region is shown in green. b) The dilated mask. c) The gradient magnitude with pixels in the new (enlarged) target region set to zero.

As you can see, this gradient magnitude image is exactly what we would expect.

Boundary Normals

If we compute the normals directly on the boundary of binary mask, the set of resulting vectors are too discretized to be of any value. Therefore, we first blur the mask, then compute the gradient of the blurred mask, and then extract the gradient values at the boundary of the original mask to obtain the vectors that we desire.

Updating the Boundary Queue

When implementing a patch based inpainting technique, the boundary pixels must be tracked. At first glance, it seems that a reasonable procedure is to simply mark all of the pixels that were filled as “not boundary”, and then re-add the pixels on the boundary of the filled patch that are touching the remaining hole to the set of active boundary pixels. However, it is important to notice that some pixels may be “stranded”. That is, pixels which used to be boundary pixels are no longer boundary pixels after filling, but they were not removed from the boundary set because they did not fall inside the target patch region. This situation is shown in

Figure 6.7. To remedy this, we explicitly check the pixels in a dilated region around the target region and remove them from the boundary set if they are no longer on the boundary. 6.7

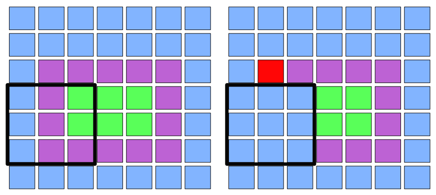
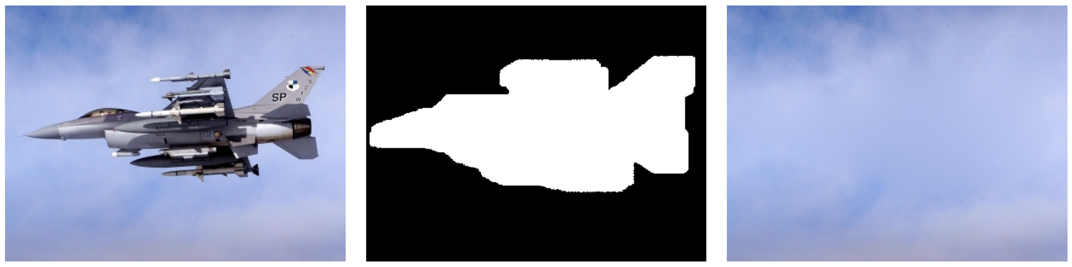


Figure 6.7: Updating the boundary queue during inpainting.

6.2 Poisson Hole Filling

As a motivating example of the power of Poisson methods in image editing, we present the problem of filling in a missing region in an image in a plausible way. In the following example, our goal is to remove the jet from the image in Figure 6.8a. To do this, we first specify the region to be removed (shown in Figure 6.8b). To motivate the problem by example, the result of the filling is shown in Figure 6.8c.



(a) Original image (b) Mask (c) Filled image

Figure 6.8: A demonstration of Poisson hole filling

Here, we have found the “smoothest” way to fill the hole Ω in the image, while exactly matching the target image, $T(x, y)$ on the hole boundary, $\partial\Omega$. That is, we have found the function $I(x, y)$ defined over the specified region that exactly meets our boundary condition at every boundary pixel, but has the smallest possible sum of gradients. This problem formulation can be expressed mathematically as shown in Equation 6.1.

$$\min_{I(x,y)\in\Omega} \iint_{\Omega} \|\nabla I(x,y)\|^2 \; dxdy \tag{6.1}$$

$$s.t. I(x,y)|_{\partial\Omega} = T(x,y)|_{\partial\Omega} \tag{6.2}$$

That is, we want to find the function (or image in this case) $I(x, y)$ that has a gradient that most closely matches the gradient of S, while at the same time I takes exactly the values of S on the hole boundary.

Using the calculus of variations (solving the Euler-Lagrange equation), it can be shown that $I(x, y)$ can be found by solving

$$\nabla^2 I(x,y) = 0 \tag{6.3}$$

$$s.t. \; I(x,y)|_{\partial\Omega} = T(x,y)|_{\partial\Omega} \tag{6.4}$$

It can be shown using Calculus of Variations that the best solution for the region inside the hole is given by the solution to

$$\nabla^2 I = 0 \tag{6.5}$$

while ensuring $I(x, y) = T(x, y)$ on the boundary of the hole. This is a Laplace equation with Dirichlet boundary conditions. Equation 6.3 says that we want to make the Laplacian of I equal to 0. Recall the Laplacian is defined as $\nabla^2 I(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$.

It is very unlikely that this equation can be solved exactly, so we must use a discrete solution which solves this equation in a least-squares sense. A discrete approximation to this function at a pixel (x, y) is given by

$$\nabla^2 f(x,y) \approx f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1) - 4f(x,y) \tag{6.6}$$

Another way to write this is to multiply the neighborhood of a pixel by a kernel:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{6.7}$$

From this, for each unknown pixel $u_{i,j}$, the equation is:

$$-4u_{i,j} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} = 0 \tag{6.8}$$

To solve this discretized Laplace equation over the entire hole, we can write a linear system of equations, with one variable for every pixel to be filled. Note that if the pixels are vector-valued (e.g. an RGB image), N of these systems must be solved independently (where N is the dimensionality of each pixel).

A sparse matrix U is constructed row by row, one row per variable. In each row, the value in the Laplacian kernel corresponding to the pixel's location relative to the current variable pixel is placed in the column corresponding to the variable id. When one of the pixels appearing in Equation 6.8 is on the border of the hole (and is therefore known), $u_{(\cdot,\cdot)}$ is replaced with $p_{(\cdot,\cdot)}$, the value of the pixel from the original image. In this case, rather than place the value of the Laplacian kernel in the U matrix, we instead multiply it by the image value and subtract the result from the corresponding entry of the b vector. That is, we move the known value to the right side of the equation.

A vector H_v, the vectorized version of the solution to the set of hole pixels, is created as the unknown vector to be solved in a system of equations.

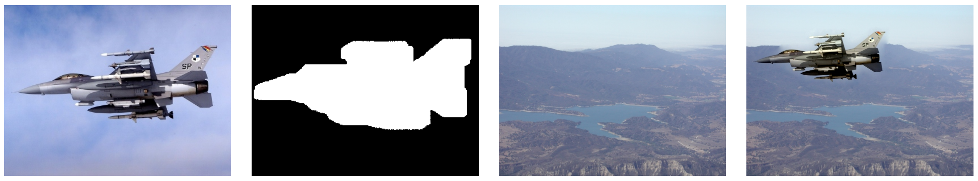
The linear system is then

$$U^T H_v = b \tag{6.9}$$

After solving for H_v, the resulting H_v is then remapped back to the pixels corresponding to each variable id to construct H. U is incredibly sparse, so a sparse solver should definitely be used.

Region Copying

In Section 6.2 we described a technique to fill a hole in a target image. In this section, we describe how a very similar approach can be used to solve a quite different problem. That is, we now wish to copy one region of an image into another image in a visually pleasing way. This is also referred to as "cloning", or "seamless cloning."[123] This is again best motivated with an example. In the following example, the goal is to copy the jet from Figure 6.9a into the image of the canyon shown in Figure 6.9c. This source and target pair along with the mask and the result are shown in 6.9.



(a) Original image (b) Mask (c) Target Image (d) Resulting Image

Figure 6.9: A demonstration of Poisson region copying

The mathematical formulation is quite similar to 6.1. That is, instead of simply minimizing the sum of the gradients, we now wish to minimize the sum of the difference of the gradients of the resulting image I from a *guidance field*, $G(x, y)$, which are the gradients of a source image, $S(x, y)$. This problem can be expressed as in Equation 6.10.

$$\min_{I(x,y)\in\Omega} \iint_{\Omega} \|\nabla I(x,y) - \nabla S(x,y)\|^2 \; dxdy \tag{6.10}$$

$$s.t. I(x,y)|_{\nabla\Omega} = T(x,y)|_{\nabla\Omega} \tag{6.11}$$

Again, the solution to this variational problem can be expressed in terms of differential operations on the images, as shown in Equation 6.12.

$$\nabla^2 H = div(G) \tag{6.12}$$

The boundary condition this time is again Dirichlet and specifies that the resulting $I(x, y)$ be equal to the target image, $T(x, y)$, at the hole boundary. In the problem at hand of copying a region of an image into another image, the guidance field is set to exactly the gradient of the source image. That is,

$$G = \nabla S \tag{6.13}$$

In this case, the right hand side of the equation has become $div(\nabla S)$, which is exactly the Laplacian of S. Other choices of guidance fields can lead to other interesting effects, several of which are explained in [123].

Just as in the case of hole filling, the discretized Laplacian equation is constructed for each hole pixel (as shown in Equation 6.14), but this time the right hand side is set to $\nabla^2 S(i, j)$. By definition, this equation is now referred to as a Poisson equation rather than a Laplace equation, as its right hand side is non-zero.

$$-4u_{i,j} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} \approx \nabla^2 S(i, j) \tag{6.14}$$

The linear system

$$U^T H_v = b \tag{6.15}$$

is constructed and solved identically as before.

Implementation Caveats

Special care must be taken when computing the Laplacian from the image derivatives. The Laplacian is defined as the divergence of the gradient:

$$\nabla^2 I = div(\nabla I) \tag{6.16}$$

Here, we have the gradient

$$\nabla I(i, j) = (D_x(i, j), D_y(i, j)) \tag{6.17}$$

so the Laplacian is, as always, the sum of the unmixed partials:

$$\nabla I^2(i, j) = \frac{\partial D_x(i, j)}{\partial x} + \frac{\partial D_y(i, j)}{\partial y} \tag{6.18}$$

The discrete operators that are used to perform these computations is critically important. If you use the standard “5-points” Laplacian operator:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{6.19}$$

you MUST use the forward difference operator to take the first set of derivatives (which produce the input to the algorithm):

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.20}$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{6.21}$$

and the backward difference operator to take the second derivatives:

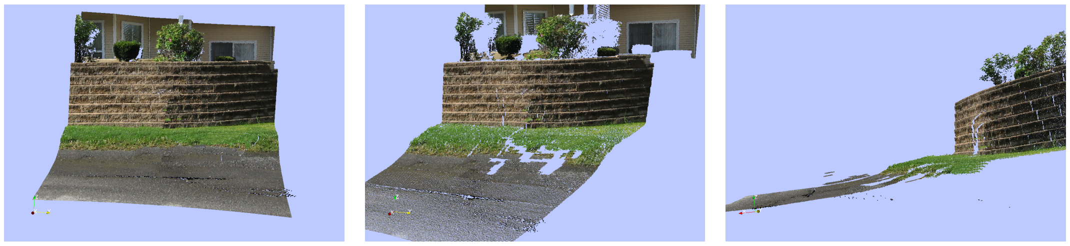
$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.22}$$

$$\begin{pmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.23}$$

or vice-versa. It possible to choose different operators, but you must choose the derivative operators so that if they are applied successively they produce exactly the Laplacian operator that you choose. If you do not do this, the result of the reconstruction will be wildly incorrect, as the underlying equations are no longer valid.

6.3 Depth Inpainting

In this section we introduce an algorithm to extend the ideas presented so far in this chapter to fill holes in LiDAR data. Initially, we approached the problem by attempting to inpaint depth images using the same technique as we have used for color images. Unfortunately, this technique cannot be applied directly. Consider the hole filling shown in Figure 6.10 (we defer the discussion of reconstructing 3D points from a completed depth image to Section 6.5).



(a) (b) (c)

Figure 6.10: Example of inpainting directly in the depth image. a) 3D structure resulting from inpainting directly in the depth image. From the original scanner viewpoint, it looks reasonable. b) From even a slightly different perspective, the result is completely unacceptable. c) Note that several clusters of points (corresponding to patches that were copied) are "floating" at an obviously incorrect depth.

The problem is that although the structure may be very similar from one patch to the next, the actual depth values can vary significantly simply based on the actual position of the object that the patch corresponds to. This is better seen in a more controlled example. In 6.11, we show the first patch that is copied in an attempt to remove the man from the scene.

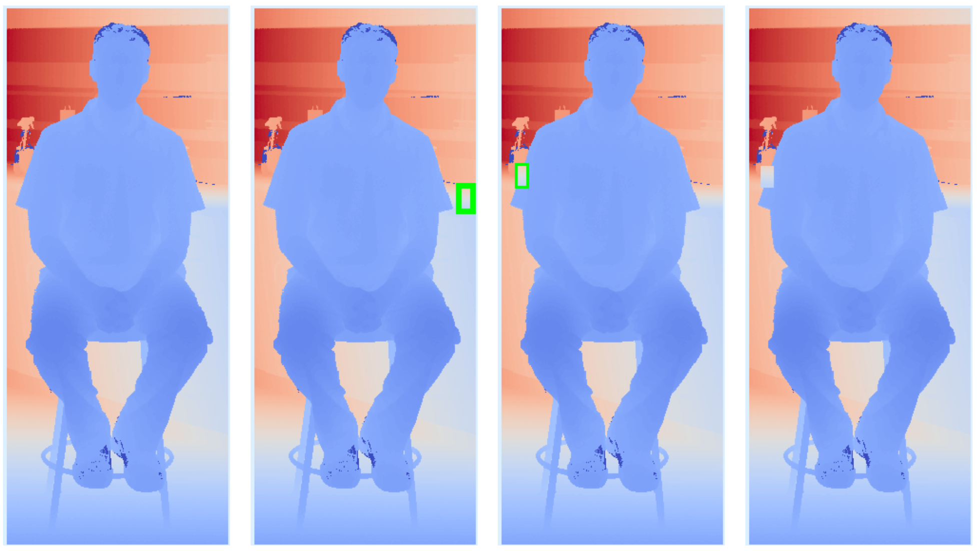

(a) Original image (b) Source patch (c) Target patch (d) Result

Figure 6.11: Example of naive depth image completion

Although the 3D structure of the counter edge is identical on the right and left sides of the person, the actual depth values are different. You can see in Figure 6.11d that the continuation of the copied depth patch into the hole is unacceptable.

In a synthetic demonstration of the problem in Figure 6.12, we show how extreme the error using this technique can be.

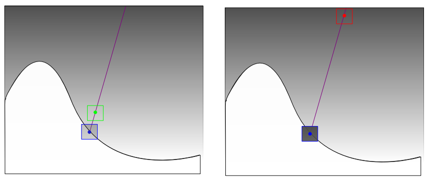
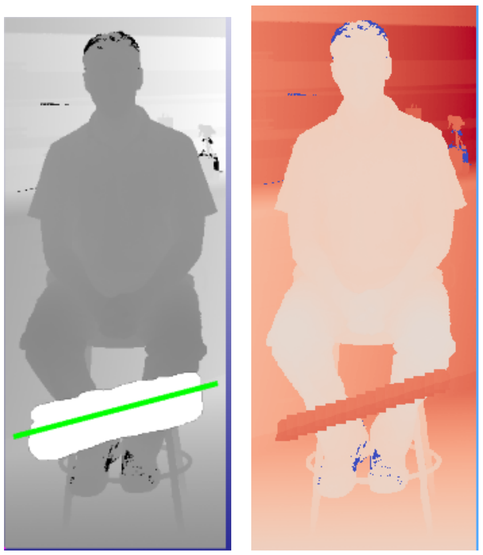


Figure 6.12: Copying depth patches.

The result looks bad when inspecting the 2D depth image, but when looking at the reconstructed 3D structure, the result is even worse. In Figure 6.13, we have attempted to propagate the structure from outside of the hole along the green line. You can see the effect shown previously in Figure 6.13b.



(a) Original image (b) Source patch

Figure 6.13: Example of naive depth image completion

The underlying problem is that, unlike in a color image, there simply are no patches in the image that “look like” the counter edge, and also occur at the same depth as the counter edge that would have been present behind the hole. We note that there is a transformation which could be applied to each patch with similar structure but different absolute depth to allow it to match well with the target patch. In Figure 6.14, we can see that if the red patch was “slid” towards the scanner to the resulting blue patch (imagine that the patch was obtained by scanning the same object placed closer to the scanner), then it would match well with the green patch.

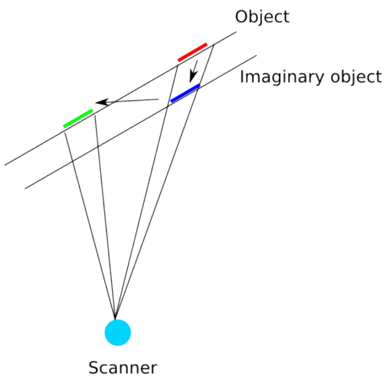


Figure 6.14: Potential patch transformation.

The problem is made even more clear by looking at the floating patches shown in the 3D result (marked with a red line). The patch on the right was copied into the depth image along the entire green line in Figure 6.13, so the depth of this patch diverges from the correct depth/location (indicated with an orange line) of the floor/counter boundary at each step.

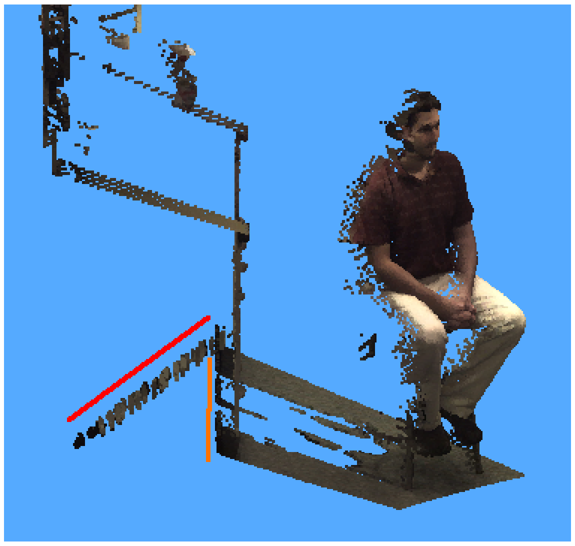


Figure 6.15: Floor result 3D.

It is worth mentioning that for some problems, this process is actually acceptable. Namely, when filling in very small holes in depth images, it suffices to perform this type of direct depth patch copying. Very thin holes have been successfully filled in [119, 71], for example, using a similar technique.

6.4 Depth Image Gradient Completion

To prevent the problem of copying depth patches which have very similar structure but different absolute depth values, we work in the gradient domain. That is, we first compute the derivatives of the depth image, and then do patch matching and copying of these gradient image pixels. The benefit of working in the gradient domain is that the information about the absolute depth is discarded while the information about the structure remains.

You can see in Figure 6.16 that the continuation of a patch in the depth gradient image is quite good even from a patch at a location of a different distance.

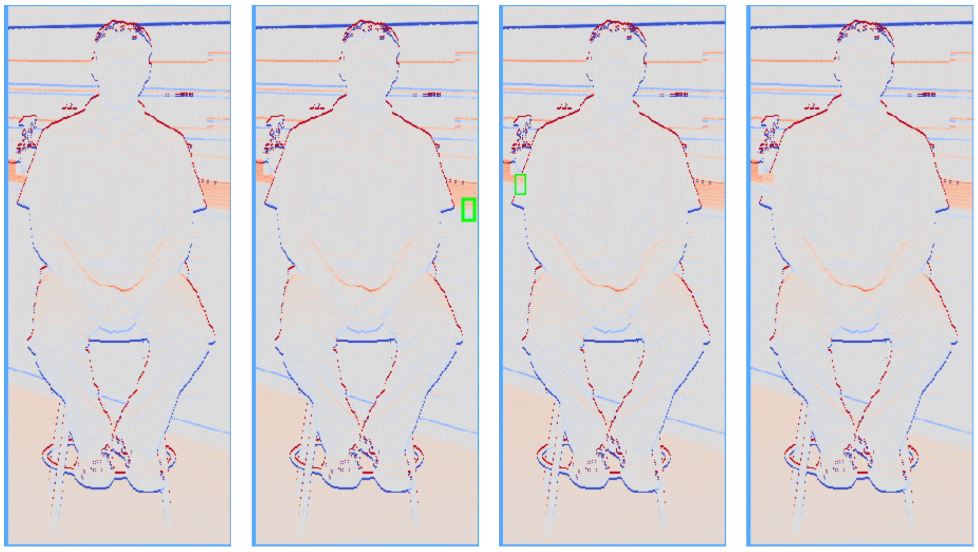
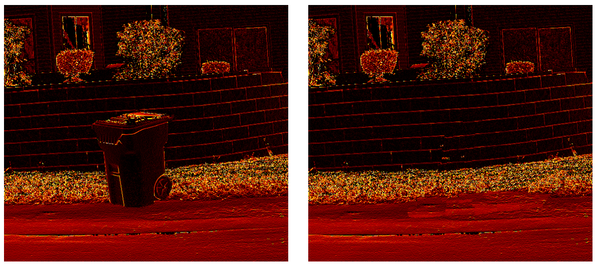

(a) Original image (b) Source patch (c) Target patch (d) Result

Figure 6.16: Example of copying a depth gradient patch

The benefit of working in the gradient domain is that the information about the absolute depth is discarded while the information about the structure remains.

After the completion, to return to an absolute depth we must solve Equation 6.10, this time with the guidance field equal to our inpainted gradients, rather than the gradient of a source image. We know the actual depth values outside of the hole so these serve as the boundary condition for the depth gradient image to depth image reconstruction problem.

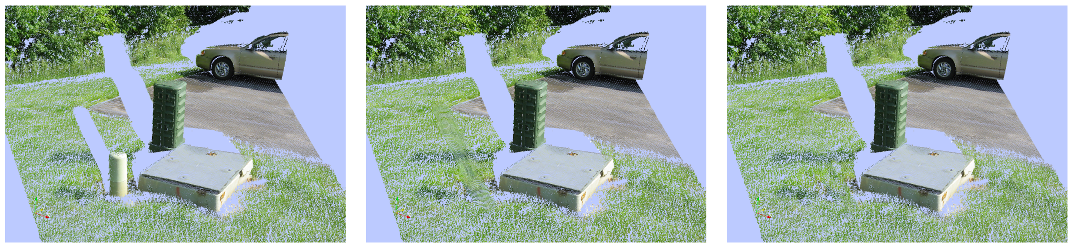
In Figure 6.17, we show the resulting depth gradient magnitude image after the inpainting.



(a) Original depth gradients. (b) Inpainted depth gradients.

Figure 6.17: The depth gradient magnitude before and after inpainting.

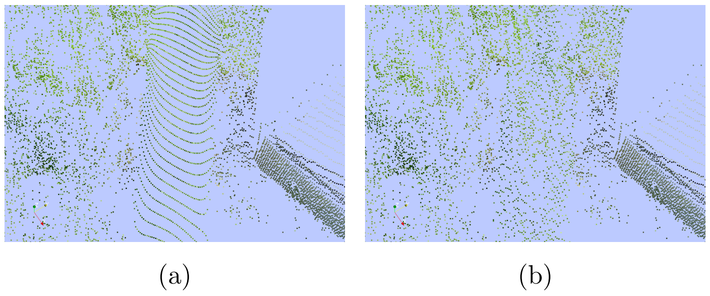
The results using this inpainting technique are dramatically better than simply treating the problem as a missing data problem. That is, if we simply fill the hole with the smoothest surface by solving the Laplace equation in Equation 6.5, significant artifacts are introduced, as shown in Figure 6.18.



(a) (b) (c)

Figure 6.18: Demonstration of filling a LiDAR shadow by finding a smooth surface compared to our gradient inpainting based method. a) 3D scan of an electric box. The scan is viewed from a different viewpoint than it was acquired from, so a long LiDAR shadow is cast behind the box on the left. b) The resulting scene after filling the LiDAR shadow using a smooth completion. c) The resulting scene after filling the LiDAR shadow using our proposed method.

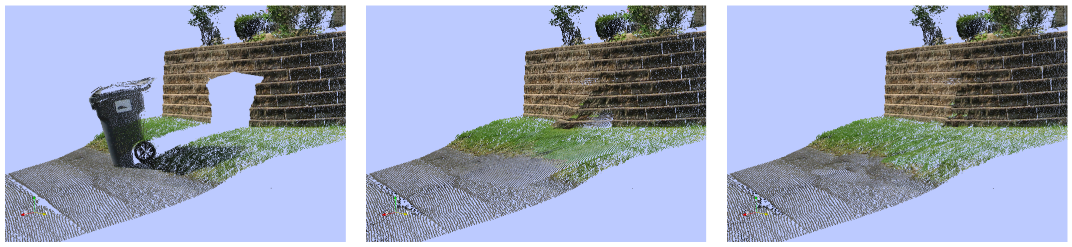
By looking closely at the region where the hole used to be, you can see that there are significant artifacts introduced, such as the uniform sampling of points which was not present in the original data. A close up inspection of these differences is shown in Figure 6.19, where the difference is made very clear.



(a) (b)

Figure 6.19: A zoomed in view of the filled region to highlight the erroneous sampling artifacts introduced by other methods. a) A zoomed in view of the resulting smooth surface completion. b) The resulting scene after filling the LiDAR shadow using a smooth completion.

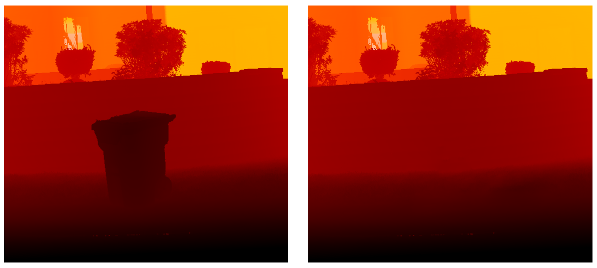
While the qualitative result of the two methods in the example above is not so different (i.e. both holes were "filled reasonably"), the region to be filled in the previous example was quite planar. If the region is non-planar, which is almost always the case in a real example, the smooth surface filling result is absolutely unacceptable with the smooth surface completion, as shown in Figure 6.20b.



(a) (b) (c)

Figure 6.20: Demonstration of filling a LiDAR shadow by finding a smooth surface compared to our gradient inpainting based method. a) 3D scan of an electric box. The scan is viewed from a different viewpoint than it was acquired from, so a long LiDAR shadow is cast behind the box on the left. b) The resulting scene after filling the LiDAR shadow using a smooth completion. c) The resulting scene after filling the LiDAR shadow using our proposed method.

The resulting depths after reconstruction from the inpainted gradient field look very much like what we would expect, as shown in Figure 6.21.



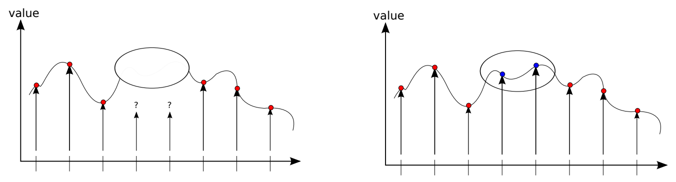
(a) Original depth image. (b) Reconstructed depth image.

Figure 6.21: The depth gradient magnitude before and after inpainting.

6.5 Replacing Depth Values To Obtain the 3D Hole Filling

Very recently, a similar approach to the one we have described in Section 6.4 has been taken to fill holes in 3D meshes [160, 158, 132]. The authors compute a tangent plane from points surrounding the hole, and then compute the projections of these points onto the plane, constructing a height field. Then then solve the Poisson equation over this height field to reconstruct a small portion of a mesh. However, we can avoid the problems mentioned by these researchers including the choice of sample spacing, and the computation of an appropriate tangent plane in which to perform the projections by taking advantage of our special case of single-viewpoint LiDAR data. This type of data allows us to work directly in the depth gradient domain, rather than needing to compute ambiguous local coordinate systems and introduce new sampling artifacts. In fact, by working in the depth gradient domain, the resulting hole completions we obtain behave exactly as if the points had been acquired by the LiDAR scanner. This makes the completions look very natural.

A demonstration of the height field approach is shown in Figure 6.22. Once the equation is solved, the resulting values can be naturally interpreted by setting the corresponding cell heights in the height field. That is, the function that is reconstructed, $Z = f(x, y)$ directly generates the height field.



(a) (b)

Figure 6.22: Filling a hole in a height field. a) A 1D height function. b) The holes in the height function naturally replaced, as in any interpolation problem.

In contrast to this, once we solve the depths in the hole, we must replace them by placing the points along the rays that were cast by the scanner. A 1-D demonstration of this is shown in Figure 6.23.

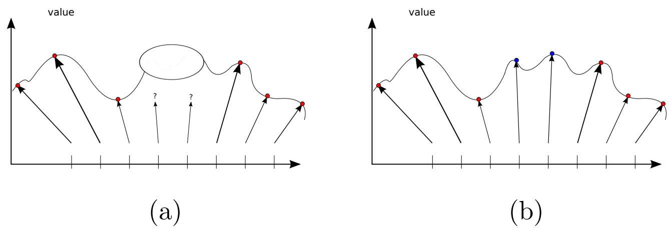


Figure 6.23: Replacing the depths, not the heights, in a 1D example. a) A function defined at distances along vectors. b) The hole in the function filled by placing points at the specified distances along the vectors.

We illustrate the full situation of replacing the depths from a scanner in Figure 6.24.

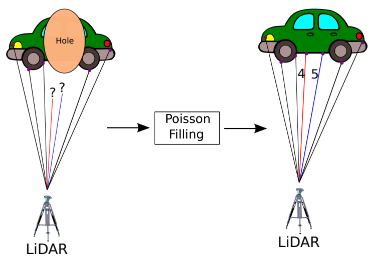
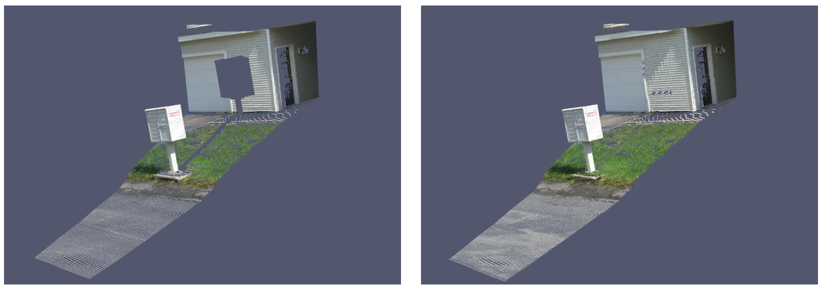


Figure 6.24: Filling depths in a LiDAR scan.

6.6 Experiments

In this section we demonstrate our algorithm on several real world data sets.

A difficult case is shown in Figure 6.25. In this data set, a mailbox has been removed from the scene. Due to the harsh shadow on the



(a) Original scan of the mailbox. (b) The mailbox has been removed.

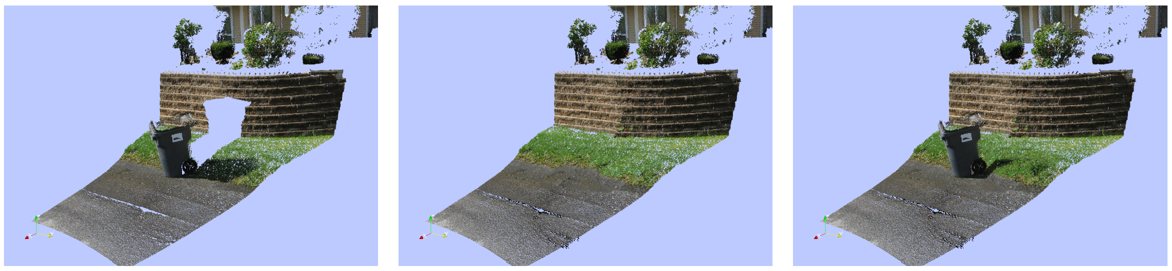
Figure 6.25: A demonstration of our approach on a data set of a mailbox in front of a complex background.

In the case of a bad completion, we could have the user specify additional constraints inside the hole as is done in [79].

6.7 Discussion

We postulate that a very similar technique could be used to, instead of fill holes by copying the gradient from elsewhere, rather smooth or otherwise alter the gradient of an existing region of a can to address some of the sampling issues that we discuss in 2.1.4.

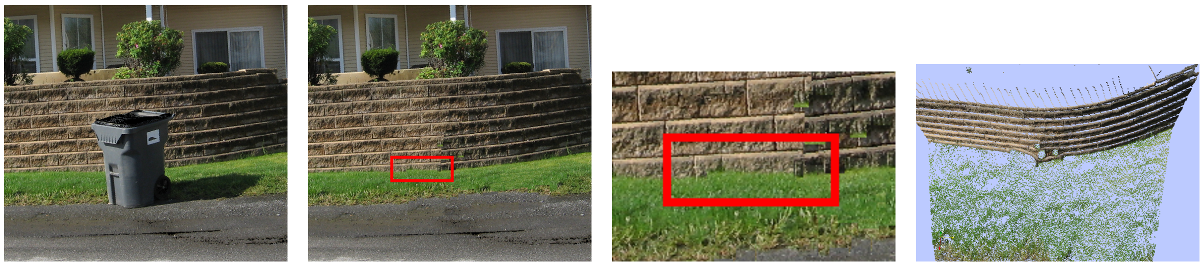
Once we have selected the object using the segmentation technique described in 5, we wish to complete the LiDAR data behind the object. This allows us to either remove the object entirely, or to make the data look feasible from any viewpoint, not just the viewpoint from which the original scan was acquired. An example of these two possibilities is shown in Figure 6.26.



(a) Original scan of the trashcan. (b) The trashcan has been removed. (c) The trashcan remains, with the scene behind it filled.

Figure 6.26: A demonstration of the two uses of LiDAR hole filling.

It is important to note that the accuracy of the image inpainting corresponding to a real scene is extremely important when using an inpainting-guided LiDAR hole filling technique such as the one we have proposed. That is, in image-only inpainting, though a completion may look good enough to seem convincing, it is not necessarily accurate enough to use to fill a hole in 3D. For example, in Figure



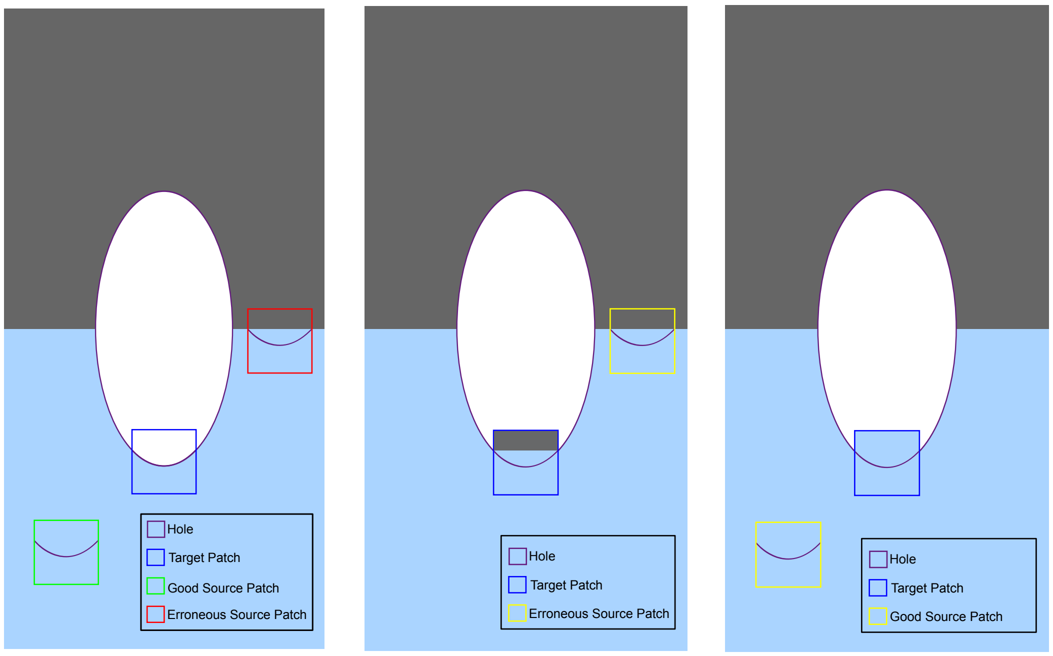
(a) Original image. (b) Inpainted image. (c) Zoomed in view. (d) 3D reconstruction.

Figure 6.27: A demonstration of the exacerbation of inpainting errors when reconstructing 3D geometry. 6.27a shows the original scene. We can see in 6.27b that he result looks feasible, but we note that the wall/grass boundary was not completed as directly as we would like in the red box. A zoomed in view of the problem is shown in 6.27b. Since grass was copied where brick should have been, the depth gradients indicate that the ground should continue for several more pixels, leading to a deformation of the wall shown in 6.27d.

It is exactly this problem that motivates our work in Chapter 7 to help reduce the number of errors similar to this that are made.

CHAPTER 7
Post-Candidacy Work

Through the work on LiDAR inpainting performed in this thesis thus far, we have used a greedy, patch-based method to fill holes in the images. At the present time, we consider the state of the art globally optimal methods too slow for this type of work. However, a major disadvantage of the greedy method that we are using is that if a single very bad patch is completed, the entire remaining inpainting result is unacceptable. We show an example of such a completion in Figure 7.1. After this bad completion shown in Figure 7.1b, you can image that now the algorithm will start matching more patches with gray in them, making the result completely unacceptable.



(a) Two potential source patches both which match the target patch in the source region exactly.

(b) The resulting inpainting using the wrong source patch.

(c) The resulting inpainting using a good source patch.

Figure 7.1: An example of the situation that occurs when patches match very well in the source region, but the resulting inpainting can be incorrect.

It is possible that the source region of the source patch and the source region of the target patch match extremely well, yet this is still a very bad patch to proceed with inpainting. This phenomenon was previously observed in [73]. In this chapter, we propose ideas for alleviating this problem to be performed post-candidacy.

Throughout our study of the exiting literature of inpainting techniques, to the best of our knowledge researchers have always trusted that the patch found during their search step of the algorithm is correct. There has been a significant amount of work to improve the patch matching process, but 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. Very similarly to our work in 4, we propose a “search-method independent check” of if the patch that was found to be the "best" is actually a good candidate. We have two ideas to alleviate this problem. The first is to attempt to detect when a bad patch is about to be copied, and ask the user if the patch is acceptable or not. 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 and 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. We have developed several new patch acceptability criteria (see Section 7.1), and intend to perform experiments to determine which are the most useful for accurately determining patch acceptability.

Second, using this notion of patch acceptability, we propose an algorithm to, instead of prompt the user for an action when a bad patch is detected, simply look elsewhere on the hole boundary to see if there is a better completion that can be performed. In initial experiments, even with a very small number of these candidate target patches, a dramatic increase in successful completions was observed.

7.1 Automatic and Interactive Patch Acceptance Criteria

Searching an entire image source patches with anything other than a simple pixel-wise comparison function like the sum of squared differences it prohibitively slow. Some attempts have been made to only search the region near the target patch, with the assumption and observation that quite often the best matching patch is indeed found in this region. However, restricting the search to this region sometimes misses the power of an exemplar-based method. For example, in Figure 7.2 we show a case where there is a repeated structure, a window, in the scene. This happens often in practice, as many man-made structures exhibit this property. In this case, we should be able to complete the missing region of the window exactly because the missing patches could be copied from the other window. Unfortunately if we had restricted the search to a pre-defined radius around the target patch, we would not have visited the patches that would have yielded the best completion.

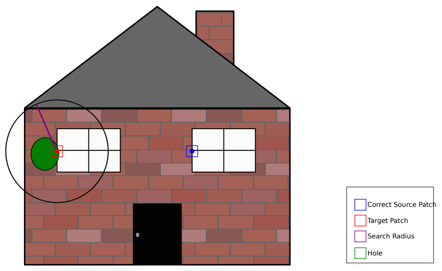


Figure 7.2: Searching only part of the image is fast, but can miss critical source patches.

To take advantage of both the speed-up of the local search as well as the repeated-structure property, we propose a two step search that first searches the local region, and if the acceptance criteria are not met, then a full image search is performed. Additionally, if the acceptance criteria are *still* not met, the user is prompted for action.

Acceptance Criteria

In post-candidacy work, we will experiment with several ideas for acceptance criteria. The first is comparing statistics of the valid region of the target patch to the hole region of the source patch, as shown in Figure 7.3.

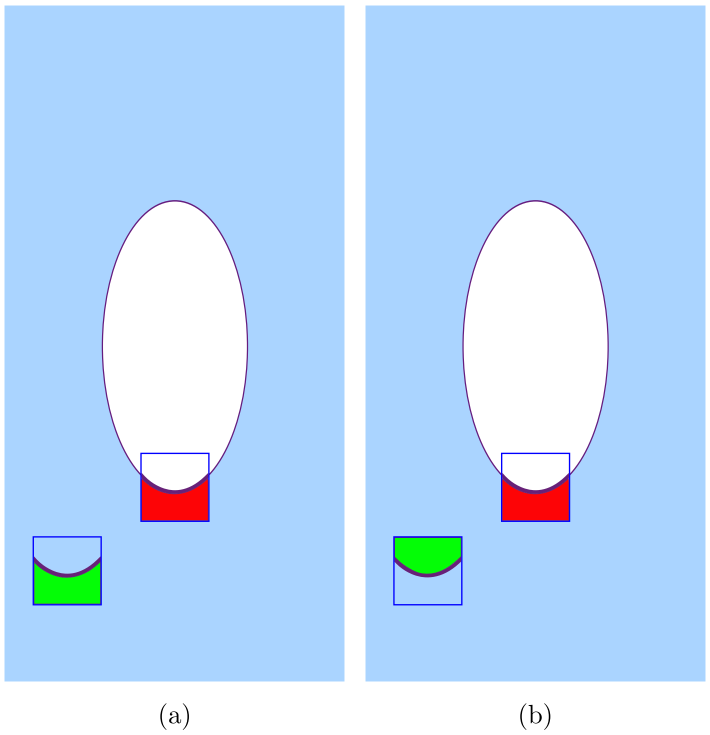
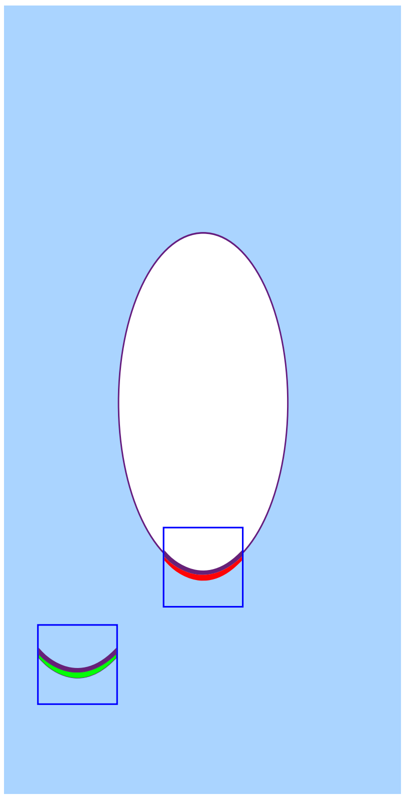


Figure 7.3: Different possibilities for determining how well a source patch matches a target patch. a) The method that most patch based inpainting algorithms use to compare patches. b) A method that can help ensure the region being copied is not too dissimilar from the region being compared.

This can also be interpreted as measuring the "energy" that is introduced into the image in the form of new strong edges. Minimizing this type of energy is common in recent algorithms ([11] for example).

After initial experimentation of such a method, we have realized that there are often situations in which the regions far away from the hole boundary erroneously contribute negatively to a matching criterion. To alleviate this, we can compare only the regions of the patches near the hole boundary, as shown in Figure 7.4.


(a) The resulting inpainting using a good source patch.

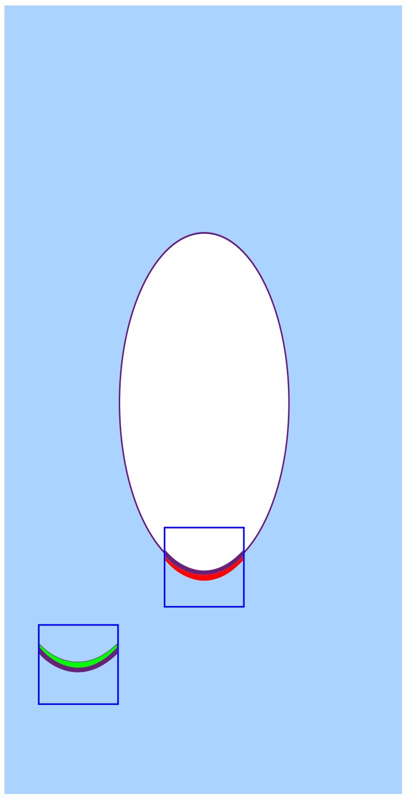

(b) The resulting inpainting using a good source patch.

Figure 7.4: Using thin regions near the hole boundary for comparison.

The reason for this is made clear in 7.5. One can see that the selected source patch leads to exactly the filling we want, even though the full source regions match very poorly.

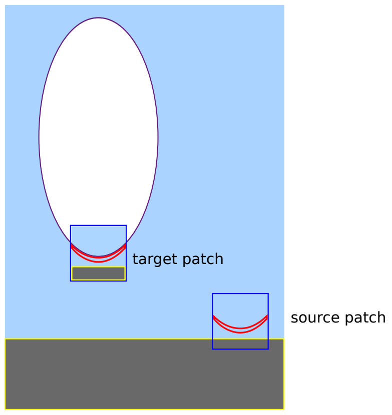


Figure 7.5: An example of when using a thin region prevents false negative matches.

To this source-valid-to-target-hole region comparison, along with the original source-valid-to-target-valid region comparison, we can apply many different types of statistical comparisons. For example, we can perform histogram comparisons of many different types (standard histogram difference as in Figure 7.6, compressed histogram difference as in Figure 7.7, quadrant-wise histogram comparisons as in Figure 7.8, etc.) average value comparisons, representative pixel comparisons, etc. Furthermore, other metrics for histogram comparison such as the EarthMovers distance [130] , the histogram intersection score [143], and the diffusion distance [105] could also be applied to any of the histogram region comparisons. As an even further extension, rather than naively collect points 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 segmentation algorithms such as [70, 163]. We could also use ideas like “Representative color” as described in the MPEG7 standard.

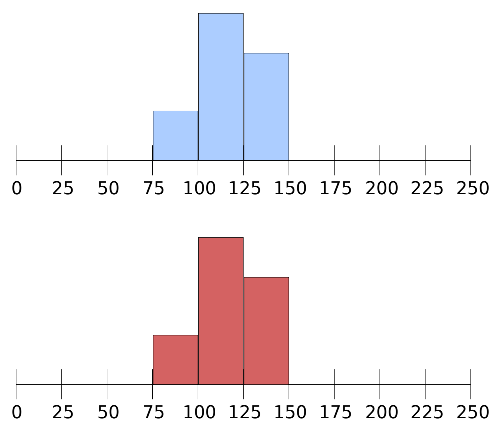


Figure 7.6: Full range histogram.

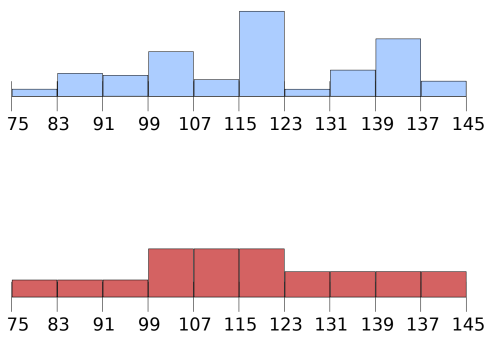


Figure 7.7: Compressed range histogram.

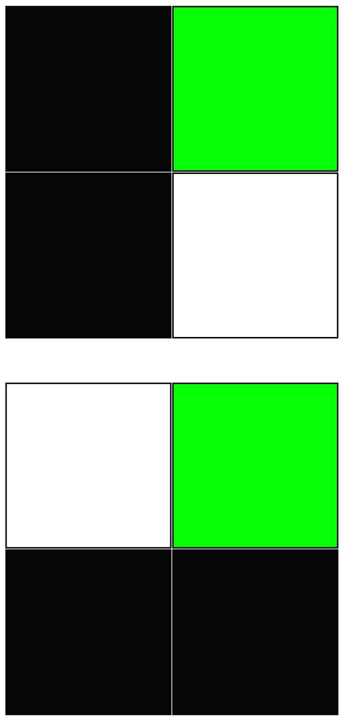


Figure 7.8: An example of when comparing the entire source region does not work well, but comparing a quadrant at a time works better.

Catching Erroneous Matches to Smooth Patches

A common occurrence that is a very bad case during greedy inpainting is when a smooth patch matches well to a textured patch. An example of such a setup is shown in Figure 7.9.

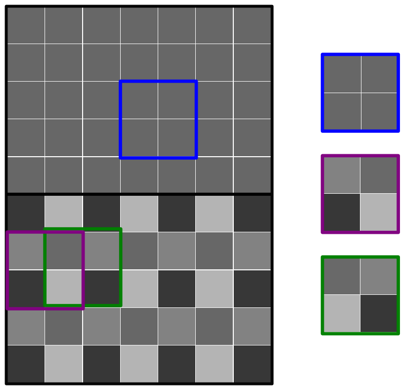


Figure 7.9: An example of how patches that "look like" each other can actually have worse SSD scores than any patch to a smooth patch.

This phenomenon can be described theoretically by the following. Consider two independent, normally distributed random variables $X = N(\mu_x, \sigma_x^2)$ and $Y = N(\mu_y, \sigma_y^2)$. The difference of these variables, $X - Y$ is distributed as $Z = N(\mu_x - \mu_y, \sigma_x^2 + \sigma_y^2)$. That is, the variance of the resulting variable is the sum of the variances of the original variables. 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 $u_{texture}$ and variance $\sigma_{texture}^2$. Now if we subtract two pixels from this region, we obtain an error distributed as $(0, 2\sigma_{texture}^2$. Now consider patch C to consist of pixels all with value c. The difference between a pixel from A and C is distributed as $(u_{texture} - c, \sigma_{texture}^2$. If c is near $u_{texture}$, then this difference is statistically smaller (variance $\sigma_{texture}^2$ versus $2\sigma_{texture}^2$) 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.

Combining Acceptance Tests

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.

User Action

When the combined acceptance tests fail, we must ask the user what to do. An interface is being developed 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 hundred 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 patch 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 very manually select which patch should be copied. This allows for a reasonable completion even in extremely hard data sets that would typically fail outright.

7.2 Forward-Looking Greedy Patch Based Inpainting

Throughout our work in this thesis, we have concluded that even a single bad patch can severely corrupt the result of a greedy image completion algorithm. However, in many cases, the human visual system is capable of viewing the completed image and not noticing that anything is "too wrong." Unfortunately, this is not the case in LiDAR hole filling - a single bad patch almost always makes the configuration of the resulting 3D points obviously unrealistic. Because of this, it is extremely important to never allow a bad patch to be copied. Although approaching this problem by necessary for LiDAR inpainting, the techniques we propose can also improve the state of the art of standard image inpainting.

To extend the ideas that we presented in Section 7.1, we can further automate the process. We propose to do this by identifying many, rather than just one, potential target patches. Then, we actually fill the one that is determined to be the best by the acceptance criteria. This situation is shown in Figure 7.10.

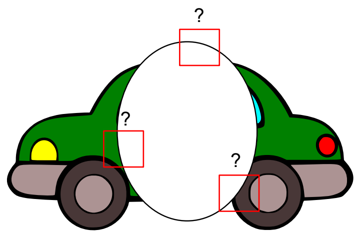


Figure 7.10: An example of inspecting multiple target patches simultaneously.

This algorithm leads to a slightly less greedy version of an exemplar based algorithm. Effectively, this allows us to defer filling a target patch for which we have declared a good match has not been found. This is beneficial in two ways. First, in the best case the hole will be filled from different, easier directions, eliminating the need to fill the difficult patch entirely. Second, by deferring the hard target patches as long as possible, the user interaction can be performed as a batch process near the end of the inpainting, rather than requiring the user to potentially interact at many points during the completion. We hope to show through several experiments that though only a few bad patches may be found out of hundreds in each hole filling, by deferring these patches until later in the operation the results can be greatly improved.

LITERATURE CITED

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] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine S. SLIC Superpixels. Technical Report June, 2010.

[3] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M Seitz, and Richard Szeliski. Building Rome in a day. In *2009 IEEE 12th International Conference on Computer Vision*, volume 54, pages 72–79. Ieee, September 2009. ISBN 978-1-4244-4420-5. doi: 10.1109/ICCV.2009.5459148. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5459148`.

[4] a. Agathos and R.B. Fisher. Colour texture fusion of multiple range images. In *Proceedings of the 4th International Conference on 3-D Digital Imaging and Modeling, 2003*, pages 139–146. Ieee, 2003. ISBN 0-7695-1991-1. doi: 10.1109/IM.2003.1240243. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1240243`.

[5] A. Agrawal, R. Chellappa, and R. Raskar. An algebraic approach to surface reconstruction from gradient fields. In *Proceedings of the 10th IEEE International Conference on Computer Vision*, pages 174–181 v. 1. Ieee, 2005. ISBN 0-7695-2334-X. doi: 10.1109/ICCV.2005.31. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1541254`.

[6] Amit Agrawal, Ramesh Raskar, and Rama Chellappa. What is the Range of Surface Reconstructions from a Gradient Field ? *Computer Science*, 3951: 578–591, 2006.

[7] Karteek Alahari, Pushmeet Kohli, and Philip H. S. Torr. Reduce, reuse & recycle: Efficiently solving multi-label MRFs. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, volume 1, pages 1–8. Ieee, June 2008. ISBN 978-1-4244-2242-5. doi: 10.1109/CVPR.2008.4587402. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4587402`.

[8] Yahya Alshawabkeh, Norbert Haala, and Dieter 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.

[9] 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. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.133. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1467438`.

[10] Michael Ashikhmin. Synthesizing Natural Textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, 2001.

[11] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Transactions on Graphics*, 26(3), July 2007. ISSN 07300301. doi: 10.1145/1276377.1276390. URL `http://portal.acm.org/citation.cfm?doid=1276377.1276390`.

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

[13] Alireza Bab-hadiashar and David Suter. Robust Range Segmentation. In *Proceedings of the 14th International Conference on Pattern Recognition, 1998*, pages 969–971 v.2, 1998.

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

[15] Connelly Barnes, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. The Generalized PatchMatch Correspondence Algorithm. *Computer Vision-ECCV 2010*, 6313:29–43, 2010.

[16] Marc Bartels and Hong Wei. Segmentation of lidar data using measures of distribution. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(7):426–431, 2006.

[17] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110 (3):346–359, 2008.

[18] Jacob Becker, Charles Stewart, and Richard J. Radke. LiDAR inpainting from a single image. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1441–1448. Ieee, September 2009. ISBN 978-1-4244-4442-7. doi: 10.1109/ICCVW.2009.5457441. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5457441`.

[19] S. Belongie and J. Malik. Matching with shape contexts. *Statistics and Analysis of Shapes*, pages 81–105, 2006. doi: 10.1109/IVL.2000.853834. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=853834`.

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

[21] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424, New York, New York, USA, 2000. ACM Press. ISBN 1581132085. doi: 10.1145/344779.344972. URL `http://portal.acm.org/citation.cfm?doid=344779.344972`.

[22] Marcelo Bertalmio, Luminita Vese, Guillermo Sapiro, and Stanley Osher. Simultaneous Structure and Texture Image Inpainting. *IEEE Transactions on Image Processing*, 12(8):882–889, 2003.

[23] Arnav V Bhavsar and A.N. Rajagopalan. Inpainting Large Missing Regions in Range Images. In *2010 20th International Conference on Pattern Recognition*, pages 3464–3467. Ieee, August 2010. ISBN 978-1-4244-7542-1. doi: 10.1109/ICPR.2010.846. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5597544`.

[24] Josep Miquel Biosca and José Luis 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. doi: 10.1016/j.isprsjprs.2007.07.010.

[25] Raphael Bornard, Emmanuelle Lecan, Louis Laborelli, and Jean-Hughes Chenot. Missing Data Correction in Still Images and Image Sequences. In *Proceedings of the 10th ACM International Conference on Multimedia*, number December, pages 355–361, 2002.

[26] Yuri Boykov and Gareth Funka-Lea. Graph Cuts and Efficient N-D Image Segmentation. *International Journal of Computer Vision*, 70(2):109–131, November 2006. ISSN 0920-5691. doi: 10.1007/s11263-006-7934-5. URL `http://www.springerlink.com/index/10.1007/s11263-006-7934-5`.

[27] Yuri Boykov and Vladimir Kolmogorov. Computing Geodesics and Minimal Surfaces via Graph Cuts. In *Proceedings on the International Conference on Computer Vision*, volume 2003, pages 1–8, 2003.

[28] Yuri Boykov and Vladimir 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. ISSN 0162-8828. doi: 10.1109/TPAMI.2004.60. URL `http://www.ncbi.nlm.nih.gov/pubmed/15742889`.

[29] Y.Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings on 8th IEEE International Conference on Computer Vision, 2001*, number July,

pages 105–112 v.1. IEEE Comput. Soc, 2001. ISBN 0-7695-1143-0. doi: 10.1109/ICCV.2001.937505. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=937505`.

[30] E Braverman, M Israeli, A Averbuch, and L Vozovoi. A Fast 3-D Poisson Solver of Arbitrary Order Accuracy. *Computational Physics*, 144(1):109–136, 1998.

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

[32] Alan Brunton, Stefanie Wuhrer, and Chang Shu. Image-based Model Completion. In *Proceedings of the 6th International Conference on 3-D Digital Imaging and Modeling*, pages 305–311. Ieee, August 2007. ISBN 0-7695-2939-9. doi: 10.1109/3DIM.2007.29. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4296769`.

[33] Nicola Brusco, Marco Andreetto, Andrea Giorgi, and Guido M Cortelazzo. 3D registration by textured spin-images. In *Fifth International Conference on 3-D Digital Imaging and Modeling, 2005*, pages 262–269, 2005.

[34] Tony Chan and Jianhong Shen. Non-Texture Inpainting by Curvature-Driven Diffusions (CDD). *J. Visual Comm. Image Rep*, 12:436–449, 2000.

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

[36] Yang Chen and Gerard 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.

[37] Tomas Chevalier, Pierre Andersson, Christina Grönwall, and Gustav Tolt. Methods for ground target detection and recognition in 3-D laser data. Technical Report December, 2006.

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

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

[40] Carlo Dal Mutto, Pietro Zanuttigh, and Guido M Cortelazzo. Scene Segmentation by Color and Depth Information and its Applications. Technical report, 2010.

[41] Paulo Dias, Vitor Sequeira, Francisco Vaz, and Joao Goncalves. Registration and Fusion of Intensity and Range Data for 3D Modelling of Real World Scenes. In *Proceedings of the 4th International Conference on 3-D Digital Imaging and Modeling, 2003*, pages 418–425, 2003.

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

[43] David Doria. A Synthetic LiDAR Scanner for VTK. *Kitware Source Magazine*, 2010. URL `http://www.insight-journal.org/browse/publication/695`.

[44] David Doria. A Greedy Patch-based Image Inpainting Framework. *Kitware Source Magazine*, 2011. URL `http://www.kitware.com/source/home/post/49`.

[45] David Doria. Poisson Editing in ITK. *Kitware Source Magazine*, 2011. URL `http://www.kitware.com/source/home/post/50`.

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

[47] David L Doria. A Conditional Mesh Front Iterator for VTK. Technical report, 2010. URL `http://www.midasjournal.org/browse/publication/724`.

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

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

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

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

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

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

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

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

[56] David L. Doria. Poisson Editing in ITK. Technical report, 2011. URL http://www.insight-journal.org/browse/publication/791.

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

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

[59] David L Doria and Richard J Radke. Consistency and Confidence : A Dual Metric for Verifying 3D Object Detections in Multiple LiDAR Scans. In *Proceedings of 3-D Digital Imaging and Modeling (3DIM) 2009, in conjunction with the International Conference on Computer Vision (ICCV)*, pages 1481–1488, 2009. ISBN 0011071001.

[60] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. Fragment-based image completion. *ACM Transactions on Graphics*, 22(3):303–312, July 2003. ISSN 07300301. doi: 10.1145/882262.882267. URL http://portal.acm.org/citation.cfm?doid=882262.882267.

[61] Alexei A Efros and William T Freeman. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 341–346, 2001.

[62] Alexei A Efros and Thomas 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.

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

[64] J. Fernandez and J. Aranda. Image segmentation combining region depth and object features. *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, pages 618–621, 2000. doi: 10.1109/ICPR.2000.905413. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=905413.

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

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

[67] R.T. Frankot and R. Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):439–451, July 1988. ISSN 01628828. doi: 10.1109/34.3909. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=3909`.

[68] Christian Frueh, Russell Sammon, and Avideh 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.

[69] Christian Früh and Avideh 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.

[70] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. Class Segmentation and Object Localization with Superpixel Neighborhoods. In *IEEE 12th International Conference on Computer Vision, 2009*, pages 670–677, 2009.

[71] Josselin Gautier, Olivier Le Meur, Christine Guillemot, Irisa Universit, De Rennes Inria, and Campus De Beaulieu Rennes. Depth-based image completion for view synthesis. In *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, 2011. ISBN 9781612841625.

[72] Aleksey Golovinskiy and Thomas Funk. Min-cut based segmentation of point clouds. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 39–46. Ieee, September 2009. ISBN 978-1-4244-4442-7. doi: 10.1109/ICCVW.2009.5457721. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5457721`.

[73] Pulkit Goyal and Sapan Diwakar. Fast and Enhanced Algorithm for Exemplar Based Image Inpainting. In *2010 Fourth Pacific-Rim Symposium on Image and Video Technology (PSIVT)*, pages 325–330, 2010.

[74] Matthew Harker and Paul O'Leary. Least squares surface reconstruction from measured gradient fields. In *Proceedings on the IEEE Conference on Computer Vision and Pattern Recognition, 2008*, pages 1–7. Ieee, June 2008. ISBN 978-1-4244-2242-5. doi: 10.1109/CVPR.2008.4587414. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4587414`.

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

[76] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. Image Analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 327–340, 2001.

[77] Berthold K P Horn and Michael J Brooks. The Variational Approach to Shape from Shading *. *Computer Vision, Graphics, and Image Processing*, 33 (2):174–208, 1986.

[78] B.K.P. Horn. Extended Gaussian Images. *Proceedings of the IEEE*, 72(12):1671–1686, 1984. ISSN 0018-9219. doi: 10.1109/PROC.1984.13073. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1457341`.

[79] Itsik Horovitz, Nahum Kiryati, and Tel Aviv. Depth from Gradient Fields and Control Points : Bias Correction in Photometric Stereo. *Image and Vision Computing*, 22(9):681–694, 2004.

[80] D Huber. Fully automatic registration of multiple 3D data sets. *Image and Vision Computing*, 21(7):637–650, July 2003. ISSN 02628856. doi: 10.1016/S0262-8856(03)00060-X. URL `http://linkinghub.elsevier.com/retrieve/pii/S026288560300060X`.

[81] Daniel 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.

[82] Alexander Patterson Iv, Philippos Mordohai, and Kostas Daniilidis. Object Detection from Large-Scale 3D Datasets using Bottom-up and Top-down Descriptors. *Computer Vision-ECCV 2008*, 5305(2008):553–566, 2008.

[83] Anupama Jagannathan, Eric L Miller, and Senior 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.

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

[85] Andrew E Johnson and Martial 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.

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

[87] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3D mesh segmentation and labeling. *ACM Transactions on Graphics*, 29(4):102:1–102:12, July 2010. ISSN 07300301. doi: 10.1145/1778765.1778839. URL `http://portal.acm.org/citation.cfm?doid=1778765.1778839`.

[88] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

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

[90] Klaas Klasing and Dirk Wollherr. Realtime segmentation of range data using continuous nearest neighbors. In *IEEE International Conference on Robotics and Automation, 2009*, pages 2431–2436, 2009. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5152498`.

[91] Klaas Klasing, Dirk Wollherr, and Martin Buss. A clustering method for efficient segmentation of 3D laser data. *2008 IEEE International Conference on Robotics and Automation*, pages 4043–4048, May 2008. doi: 10.1109/ROBOT.2008.4543832. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4543832`.

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

[93] N. Komodakis. Image Completion Using Global Optimization. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR'06)*, pages 442–452. Ieee, 2006. ISBN 0-7695-2597-0. doi: 10.1109/CVPR.2006.141. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1640791`.

[94] Nikos Komodakis and Georgios Tziritas. Image completion using efficient belief propagation via priority scheduling and dynamic pruning. *IEEE transactions on image processing*, 16(11):2649–2661, November 2007. ISSN 1057-7149. URL `http://www.ncbi.nlm.nih.gov/pubmed/17990742`.

[95] Marcel Kortgen, Gil-Joo Park, Marcin Novotni, and Reinhard Klein. 3D Shape Matching with 3D Shape Contexts. In *Proceedings of The 7th Central European Seminar on Computer Graphics*, 2003.

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

[97] Longin Jan Latecki and Rolf Lakaemper. Polygonal Approximation of Laser Range Data Based on Perceptual Grouping and EM. In *Proceedings of the 2006 International Conference on Robotics and Automation*, volume 1, pages 790–796, 2006.

[98] Longin Jan Latecki, Rolf Lakaemper, Xinyu Sun, and Diedrich Wolter. Building Polygonal Maps from Laser Range Data. In *ECAI International Cognitive Robotics Workshop*, volume 1, pages 1–7, 2004.

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

[100] Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. Seamless Image Stitching in the Gradient Domain. *Computer Science*, 3024:377–389, 2004.

[101] Alex Levinshtein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, and Sven J Dickinson. TurboPixels : Fast Superpixels Using Geometric Flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12): 2290–2297, 2009.

[102] Marc 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.

[103] Yin Li, Jian Sun, Chi-keung Tang, and Heung-yeung Shum. Lazy Snapping. *ACM Transactions on Graphics*, 23(3):303–308, 2004.

[104] Bart Liefers and Supervision Robby T Tan. Gradient Space Manipulation and Shadow Removal. Technical report, Utrecht University, 2011.

[105] Haibin Ling and Kazunori Okada. Diffusion Distance for Histogram Comparison. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 246–253. Ieee, 2006. ISBN 0-7695-2597-0. doi: 10.1109/CVPR.2006.99. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1640766`.

[106] Lars Linsen. Point Cloud Representation. Technical report, 2001.

[107] Rong Liu and Hao Zhang. Segmentation of 3D Meshes through Spectral Clustering. In *Proceedings. 12th Pacific Conference on Computer Graphics and Applications, 2004*, pages 298–305, 2004.

[108] 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. doi: http://dx.doi.org/10.1109/ICCV.1999.790410. URL `http://dx.doi.org/10.1109/ICCV.1999.790410`.

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

[110] A. Makadia, A.I. Patterson, and K. Daniilidis. Fully Automatic Registration of 3D Point Clouds. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1297–1304. Ieee, 2006. ISBN 0-7695-2597-0. doi: 10.1109/CVPR.2006.122. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1640899`.

[111] David Marshall, Gabor Lukacs, and Ralph Martin. Robust Segmentation of Primitives from Range Data in the Presence of Geometric Degeneracy æ. *Analysis*, 23(3):304–314, 2001.

[112] Andrew Mastin, Jeremy Kepner, and John 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.

[113] Bogdan Matei, Ying Shan, Senior Member, Rakesh Kumar, Daniel Huber, and Martial 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.

[114] Ajmal S Mian, Mohammed Bennamoun, and Robyn 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.

[115] Krystian Mikolajczyk and Cordelia Schmid. Performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, October 2005. ISSN 0162-8828. doi: 10.1109/TPAMI.2005.188. URL `http://www.ncbi.nlm.nih.gov/pubmed/16237996`.

[116] Niloy J Mitra, Natasha Gelfand, Helmut Pottmann, and Leonidas Guibas. Registration of Point Cloud Data from a Geometric Optimization Perspective. In *Eurographics Symposium on Geomtetry Processing*, 2004.

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

[118] Heung-Sun Ng, Tai-Pang Wu, and Chi-Keung Tang. Surface-from-gradients without discrete integrability enforcement: A Gaussian kernel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(11):2085–2099, November 2010. ISSN 1939-3539. doi: 10.1109/TPAMI.2009.183. URL `http://www.ncbi.nlm.nih.gov/pubmed/20847395`.

[119] Kwan-jung Oh, Sehoon Yea, and Yo-sung Ho. Hole-Filling Method Using Depth Based In-Painting For View Synthesis in Free Viewpoint Television (FTV) and 3D Video. In *Picture Coding Symposium, 2009*, pages 1–4, 2009.

[120] Manuel M Oliveira, Brian Bowen, Richard McKenna, and Yu-Sung Chang. Fast Digital Image Inpainting. In *Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001)*, number Viip, 2001.

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

[122] Seyoun Park, Xiaohu Guo, Hayong Shin, and Hong Qin. Shape and Appearance Repair for Incomplete Point Surfaces. In *Proceedings of 10th IEEE International Conference on Computer Vision, 2005*, pages 1260–1267 v. 2, 2005.

[123] P. Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Transactions on Graphics*, 22(3):313–318, 2003. URL `http://portal.acm.org/citation.cfm?id=882269`.

[124] N Petrovic, Ira Cohen, Brendan J Frey, R Koetter, and Thomas S Huang. Enforcing Integrability for Surface Reconstruction Algorithms Using Belief Propagation in Graphical Models. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001*, pages I–743–I–748 v.1, 2001.

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

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

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

[128] Patil Sandeep Ramsing and Sachin 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] Carsten Rother and Andrew Blake. GrabCut Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics*, 23(3): 309–314, 2004.

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

[131] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152. IEEE Comput. Soc, 2001. ISBN 0-7695-0984-3. doi: 10.1109/IM.2001.924423. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=924423`.

[132] Santiago Salamanca, Pilar Merch, Antonio Ad, Carlos Cerrada, and E T S De Ingeniera Inform. Filling Holes in 3D Meshes using Image Restoration Algorithms. In *Proceedings of 3DPVT'08 - the 4th International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 1–8, 2008.

[133] Hanan Samet. Implementing ray tracing with octrees and neighbor finding. *Computers & Graphics*, 13(4):445–460, 1989. ISSN 00978493. doi: 10.1016/0097-8493(89)90006-X. URL `http://linkinghub.elsevier.com/retrieve/pii/009784938990006X`.

[134] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Transactions on Graphics*, 23(3):878–887, August 2004. ISSN 07300301. doi: 10.1145/1015706.1015814. URL `http://portal.acm.org/citation.cfm?doid=1015706.1015814`.

[135] 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. ISSN 00978493. doi: 10.1016/j.cag.2006.10.004. URL `http://linkinghub.elsevier.com/retrieve/pii/S009784930600224X`.

[136] Jianbing Shen, Xiaogang Jin, Chuan Zhou, and Charlie C L Wang. Gradient based image completion by solving. *Computers & Graphics*, 31(1):119–126, 2007.

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

[138] Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. Summarizing Visual Data Using Bidirectional Similarity. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008*, number ii, pages 1–8, 2008.

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

[140] a Specht, A Sappa, and M Devy. Edge registration versus triangular mesh registration, a comparative study. *Signal Processing: Image Communication*, 20(9-10):853–868, October 2005. ISSN 09235965. doi: 10.1016/j.image.2005.05.010. URL `http://linkinghub.elsevier.com/retrieve/pii/S0923596505000688`.

[141] Pavlos Stavrou, Pavlos Mavridis, Georgios Papaioannou, and Georgios Passalis. 3D Object Repair Using 2D Algorithms. *Computational Science–ICCS 2006*, 3992:271–278, 2006.

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

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

[144] Zinovi Tauber, Ze-nian Li, and Mark S Drew. Review and Preview : Disocclusion by Inpainting for Image-based Rendering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(4): 527–540, 2007.

[145] Alexandru Telea. An image inpainting technique based on nonparametric kernel estimation. *Journal of Graphics Tools*, 9(1):23–24, May 2004. doi: 10.1109/ICCCAS.2008.4657916. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4657916`.

[146] Kalaivani Thangamani, Takeshi Kurata, and Tomoya Ishikawa. Superpixel Based Inpainting for Interactive 3D Indoor Modeler. In *MVA 2011 IAPR Conference on Machine Vision Applications*, pages 156–159, 2011.

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

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

[149] Olga Veksler. Graph Cut Based Optimization for MRFs with Truncated Convex Priors. In *IEEE Conference on Computer Vision and Pattern Recognition, 2007.*, pages 1–8. Ieee, June 2007. ISBN 1-4244-1179-3. doi: 10.1109/CVPR.2007.383249. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4270274`.

[150] 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. ISBN 0-7803-7750-8. doi: 10.1109/ICIP.2003.1246828. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1246828`.

[151] Paul Viola and Michael Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001*, pages I–511–I–518 v.1, 2001.

[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. ISSN 02628856. doi: 10.1016/j.imavis.2005.12.006. URL `http://linkinghub.elsevier.com/retrieve/pii/S0262885606000199`.

[153] J. Wang and M.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. IEEE Comput. Soc, 2003. ISBN 0-7695-2032-4. doi: 10.1109/SIBGRA.2003.1240986. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1240986`.

[154] Jianning Wang and Manuel M Oliveira. Improved Scene Reconstruction from Range Images. *EUROGRAPHICS*, 21(3):521–530, 2002.

[155] Jun Wang and Jie Shan. Segmentation of LiDAR point clouds for building extraction. In *American Society for Photogrammetry and Remote Sensing Annual Conference*, 2009. URL `http://cobweb.ecn.purdue.edu/~jshan/publications/2009/ASPRS_2009_Lidar.pdf`.

[156] Liang Wang, Jin Hailin, Ruigang Yang, and Minglun Gong. Stereoscopic Inpainting : Joint Color and Depth Completion from Stereo Images. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008*, pages 1–8, 2008.

[157] Wei Wang, Longshe Huo, Wei Zeng, Qingming Huang, and Wen Gao. Depth image segmentation for improved virtual view image quality in 3-DTV. In *Proceedings of the 2007 International Symposium on Intelligent Signal Processing and Communication Systems*, pages 300–303, 2007.

[158] Hans Weghorn, Hans Weghorn Ed, Monika Kochanowski, Philipp Jenke, and Wolfgang Straß er. Analysis of Texture Synthesis Algorithms with Respect to Usage for Hole-Filling in 3D Geometry. In *Proceedings of the 4th Annual Meeting on Information Technology and Computer Science-ITCS*, volume 4th, pages 1–6, 2008.

[159] Tiangong Wei and Reinhard Klette. Regularization Method for Depth from Noisy Gradient Vector Fields. Technical report, 2002.

[160] Tiangong Wei and Reinhard Klette. On Depth Recovery from Gradient Vector Fields. In *Algorithms, architectures and information systems security*, pages 75–95. 2009.

[161] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-Time Completion of Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):463–476, 2007.

[162] Chunxia Xiao, Meng Liu, Yongwei Nie, and Zhao Dong. Fast exact nearest patch matching for patch-based image editing and processing. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1122–34, August 2011. ISSN 1077-2626. doi: 10.1109/TVCG.2010.226. URL `http://www.ncbi.nlm.nih.gov/pubmed/21659679`.

[163] Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. Image Smoothing via L0 Gradient Minimization. *ACM Transactions on Graphics*, 30(6):1–12, 2011. doi: 10.1145/2024156.2024208.

[164] Ning Xu, Ravi Bansal, and Narendra 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.

[165] Songhua Xu, Athinodoros Georghiades, Holly Rushmeier, Julie Dorsey, and Leonard McMillan. Image Guided Geometry Inference 1 Introduction 2 Previous Work. In *Proceedings of the 3rd International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 310–317, 2006.

[166] Zongben Xu and Jian Sun. Image inpainting by patch propagation using patch sparsity. *IEEE Transactions on Image Processing*, 19(5):1153–1165, May 2010. ISSN 1941-0042. doi: 10.1109/TIP.2010.2042098. URL `http://www.ncbi.nlm.nih.gov/pubmed/20129864`.

[167] Michael Ying Yang and Wolfgang F. Plane Detection in Point Cloud Data. Technical Report 1, 2010.

[168] Gene Yu, Michael Grossberg, George Wolberg, and Ioannis Stamos. Think Globally , Cluster Locally : A Unified Framework for Range Segmentation. In *Proceedings of the 4th International Symposium on 3D Data Processing, Visualization and Transmission*, 2008.

[169] Yun Zeng, Wei Chen, Dimitris Samaras, and Qunsheng 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.

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