Surface Reconstruction and Display from Range and Color Data

by

Kari Pulli

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

1997

Approved by____

(Chairperson of Supervisory Committee)

____

____

Program Authorized
to Offer Degree____

Date____

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with “fair use” as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted “the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform.”

Signature____________________________

Date________________________________

University of Washington

Abstract

Surface Reconstruction and Display from Range and Color Data

by Kari Pulli

Chairperson of Supervisory Committee
Professor Linda G. Shapiro
Computer Science and Engineering

This dissertation addresses the problem of scanning both the color and geometry of real objects and displaying realistic images of the scanned objects from arbitrary viewpoints. We present a complete system that uses a stereo camera system with active lighting to scan the object surface geometry and color as visible from one point of view. Scans expressed in sensor coordinates are registered into a single object-centered coordinate system by aligning both the color and geometry where the scans overlap. The range data are integrated into a surface model using a robust hierarchical space carving method. The fit of the resulting approximate mesh to data is improved and the mesh structure is simplified using mesh optimization methods. In addition, two methods are developed for view-dependent display of the reconstructed surfaces. The first method integrates the geometric data into a single model as described above and projects the color data from the input images onto the surface. The second method models the scans separately as textured triangle meshes, and integrates the data during display by rendering several partial models from the current viewpoint and combining the images pixel by pixel.

Table of Contents

List of Figures

List of Tables

Acknowledgments

There are many people I would like to thank for helping me in the research which lead to this dissertation. First, I would like to thank my advisor Linda Shapiro: she is the nicest advisor one could hope to have. The other members in my advisory committee, Tony DeRose, Tom Duchamp, and Werner Stuetzle, have been equally important for focusing and developing this dissertation. Other people attending our meetings include John McDonald, Hugues Hoppe, Michael Cohen, and Rick Szeliski; working with all these people has been a most pleasurable learning experience. Special thanks to Hugues Hoppe for code, advice on how to use it, and surface models.

I thank the staff in general for taking care of the paper work and for keeping the computer systems running, and Lorraine Evans in particular. I thank David Salesin for including me in the graphics group as the only student that wasn't supervised by him. I am grateful for all my fellow students in vision (Mauro, Habib, Pam, ...) and in graphics (Eric, Frederic, Daniel, ...) for giving me friendship and support during my studies. I especially want to thank Habib Abi-Rached for his hard work with our stereo scanner. I want to thank all my bug house, bridge, and hiking partners for the relaxation in the midst of hard work, and finally, Omid Madani for being such a good friend.

I wish to thank my committee, Hugues Hoppe, Eric Stollnitz and Brian Curless for proof-reading this dissertation and for providing valuable feedback.

Summer internships offered a chance for a much needed break from schoolwork and for learning useful new skills. I enjoyed working with Chas Boyd at Microsoft, Mark Segal at Silicon Graphics, and Michael Lounsbery at Alias|Wavefront.

My research career began at the University of Oulu in Finland. I would like to thank Matti Pietikäinen, Olli Silvén, Juha Röning, Tapani Westman, and Visa Koivunen who all in their own way helped me and taught me how to do research.

I am greatful to several foundations and institutions for direct and indirect financial support for my studies. I thank ASLA/Fulbright Foundation, Academy of Finland, Finnish Cultural Foundation, Emil Aaltonen Foundation, Seppo Säynäjäkangas Foundation, National Science Foundation and Human Interface Technology Laboratory.

Finally, I want to thank my parents and my wife Anne and daughter Kristiina for their constant support.

Chapter 1

Introduction

Computer Vision and Computer Graphics are like opposite sides of the same coin: in vision, a description of an object or scene is derived from images, while in graphics images are rendered from geometric descriptions. This dissertation addresses issues from both of these disciplines by developing methods for sampling both the surface geometry and color of individual objects, building surface representations, and finally displaying realistic color images of those objects from arbitrary viewpoints.

We begin this chapter by defining a set of subproblems within the greater framework of scanning and displaying objects. We then discuss some applications that benefit from techniques developed in this dissertation. Some relevant previous work is discussed, and the chapter concludes with an overview of the dissertation.

1.1 Problem statement

This dissertation presents a complete system for scanning and displaying textured objects. The tasks required to fulfill our goal can be organized into a sequence of stages that process the input data. Each of these stages presents us with its own problems. Specifically, the problems addressed in this dissertation are:

- How can we scan the object surface geometry using color cameras and structured light?

- Our scanner only allows us to digitize one view of an object at a time. Since the scan data is expressed in the sensor coordinate system, and the sensor moves with respect to the object between views, the data in different views are expressed in different coordinate systems. How can we accurately express all the data in a single object-centered coordinate system?

- How can we convert separate range views into a single surface description? How can we use our knowledge of the scanning process to make this process more robust and reliable?

- How can we manipulate the surface description so that it better approximates the input data, and how can we control the size and resolution of the surface representation?

- How can we combine the color information from the different views together with the surface geometry to render realistic images of our model from arbitrary viewpoints?

1.2 Motivating applications

Our research is motivated by numerous applications. The following two sections first discuss applications that require scanning surface geometry, followed by applications that additionally require colored and textured surface descriptions.

1.2.1 Surface reconstruction

Surface reconstruction applications take geometric scan data as input and produce surface descriptions that interpolate or approximate the data. The geometric scan data is typically expressed as a set of range maps called views. Each view is like a 2D image, except that at each pixel the coordinates of the closest surface point visible through the pixel are stored instead of a color value.

Reverse engineering

Today's CAM (Computer Aided Manufacturing) systems allows one to manufacture objects from their CAD (Computer Aided Design) specifications. However, there may not exist a CAD model for an old or a custom-made part. If a replica is needed, one could scan the object and create a geometric model that is then used to reproduce the object.

Design

Although CAD system user interfaces are becoming more natural, they still provide a poor interface for designing free-form surfaces. Dragging control points on a computer display is a far cry from the direct interaction an artist can have with clay or wood. Surface reconstruction allows designers to create initial models using materials of their choice and then scan the geometry and convert it to CAD models. One could also iterate between manufacturing prototypes, manually reshaping them, and constructing a new model of the modified version.

Inspection

Manufactured objects can be scanned and the data can be compared to the specifications. Detected deviations from the models can be used to calibrate a new manufacturing process, or it can used in quality control to detect and discard faulty parts.

Planning of medical treatment

Surface reconstruction has many applications in medicine. Scanning the shape of a patient's body can help a doctor decide the direction and magnitude of radiation for removing a tumor. In plastic surgery, scanning the patient can help a doctor to quantify how much fat to remove or how large an implant to insert to obtain the desired outcome.

Custom fitting

Surface reconstruction allows automatic custom fitting of generic products to a wide variety of body sizes and shapes. Good examples of customized products include prosthetics and clothes.

1.2.2 Textured objects

For displaying objects, rather than replicating or measuring them, one needs color information in addition to geometric information. Some scanners, including ours, indeed produce not only the 3D coordinates of visible surface points, but also the color of the surface at those points. The requirements for the accuracy of the geometric data are often much more lenient, as the color data can capture the *appearance* of fine surface detail.

Populating virtual worlds

Populating virtual worlds or games with everyday objects and characters can be a labor-intensive task if the models are to be created by artists using CAD software. Further, such objects tend to look distinctly artificial. This task can be made easier and the results more convincing by scanning both the geometry and appearance of real-world objects, or even people.

Displaying objects on internet

The influence of the internet is becoming more pervasive in our society. The world wide web used to contain only text and some images, but 3D applications have already begun to appear. Instead of looking at an object from some fixed viewpoint, one can now view objects from an arbitrary viewpoint. Obvious applications include building virtual museums and thus making historical artifacts more accessible both to scholars and to the general public, as well as commerce over the internet where the potential buyer can visualize the products before purchase.

Special effects for films

Computer graphics is increasingly used in films. Special effects that would be otherwise impossible, infeasible, or just expensive can be digitally combined with video sequences. The extra characters, objects, or backgrounds tend to look more realistic if they are scanned from real counterparts than if they were completely generated by a computer.

1.3 Previous work

In this section I briefly cover some previous work that most influenced the research described in this dissertation. Further descriptions of related work can be found in Chapters 2, 3, 4, and 5.

1.3.1 Surfaces from range

There has been a large number of PhD dissertations and other research in surface reconstruction from range data. The work that first motivated my research in range vision was Paul Besl's dissertation [Besl 86, Besl 88]. Besl did an excellent job of exploring low-level range image processing and segmentation. Also, his later work in registration of range data has been influential [Besl & McKay 92].

The greatest influence on this research was that of Hugues Hoppe at University of Washington with Professors Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle [Hoppe *et al.* 92, Hoppe *et al.* 93, Hoppe *et al.* 94, Hoppe 94]. Whereas Besl concentrated on low-level processing of range maps, Hoppe *et al.* developed a three-phase system for reconstructing arbitrary surfaces from unstructured 3D point data. The first phase creates an approximate triangle mesh from the point data by estimating a signed distance function from the data and then extracting the zero set of that function. The second phase takes the initial mesh and the point data and iteratively improves the fit of the mesh to the data and simplifies the mesh while keeping it close to the data. In the third phase the surface representation is changed to Loop's subdivision scheme [Loop 87] that can represent curved surfaces more accurately and more compactly than a triangle mesh. Hoppe *et al.* extended Loop's scheme to allow sharp surface features such as creases and corners.

Around the same time Yang Chen [1994] worked with Professor Gerard Medioni at USC on similar problems. Their research addressed registration of range maps into a single coordinate system and then integrating the data into surface representation.

The most recent work in surface reconstruction that had a large influence on my research was that of Brian Curless who worked with Professor Marc Levoy at Stanford [Curless 97]. Curless's dissertation has two parts. In the first part spacetime analysis, a more robust and accurate scheme for estimating depth using a light stripe scanning system was developed [Curless & Levoy 95]. In the second part a volumetric method for building complex models from range images was developed [Curless & Levoy 96].

1.3.2 Image-based rendering

My work on view-dependent texturing was heavily influenced by the image-based rendering papers published in SIGGRAPH 96. The Microsoft Lumigraph [Gortler *et al.* 96] and Stanford Light Field Rendering [Levoy & Hanrahan 96] address the same problem: given a collection of color images of an object from known viewpoints, how can one create new images of the same object from an arbitrary viewpoint? Their solution was to think of the pixels of the input images as half-rays ending at the camera image plane and encoding the apparent directional radiance of the object surface. From these samples a function that returns a color for a directed ray can be calculated. New images can then be created by evaluating that function over the rays associated with each pixel in the output image and painting the pixels with the returned colors. The approach is quite general and allows realistic rendering of objects that are hard to model and render using traditional graphics methods. However, a very large set of input images is required, and the 4D function mapping rays to colors requires a large storage.

Debevec *et al.* [1996] described a system that allows a user to interactively reconstruct architectural scenes from color images. After the user specifies the basic structure of the geometric models and marks correspondences between image features and model features, the system calculates the actual dimensions of the model elements. The models are then view-dependently textured using the original camera images.

1.4 Contributions

Here is a summary of the contributions in this dissertation:

- We have constructed a practical range scanner that uses inexpensive color cameras and a controlled light source. The range data is inferred from camera images using optical triangulation. The quality of the range data is increased by adapting spacetime analysis [Curless & Levoy 95] for our scanner.

- We have developed a new method for pairwise registration of range and color scans. Our method directly addresses the most difficult part of 3D registration: establishing reliable point correspondences between the two scans. We implement a robust optimization method for aligning the views using the established point pairs.

- We have developed a method for simultaneously registering multiple range maps into a single coordinate system. Our method can be used in conjunction with any pairwise registration method. The scans are first registered pairwise. The results of pairwise registration create constraints that can be then used to simultaneously find a global registration.

- We have developed a simple and efficient hierarchical space carving method for creating approximate meshes from registered range maps.

- We have developed two methods for view-dependent texturing of geometric models using color images. The first method is used to texture complete surface models, while the second method models each range scan separately and integrates the scans during display time in screen space.

1.5 Overview

The first three chapters in this dissertation discuss methods for reconstructing geometric surface models from range scans. Chapter 2 deals with data acquisition, Chapter 3 addresses registration of the scan data, and Chapter 4 discusses methods to reconstruct surfaces from registered range data. Chapter 5 concerns the use of color data and presents two methods for view-dependent texturing of scanned surfaces. Chapter 6 concludes the thesis. Some material presented in Chapters 4 and 5 has been published previously [Pulli *et al.* 97a, Pulli *et al.* 97b, Pulli *et al.* 97c].

Chapter 2

Data acquisition

2.1 Introduction

It is possible to develop algorithms for surface reconstruction and view-dependent texturing without having an actual range and color scanner. However, with the aid of an accurate scanner we can create new data sets of many different classes of objects and surface materials under various scanning configurations. We can get even more control over the input data by building a scanner instead of buying one as no data remains inaccessible inside the scanning system.

This chapter describes the system we built for scanning both range and color data. In Section 2.2 we describe the hardware configuration of our scanner as well as the calibration process. In Section 2.2.1 we discuss our simple but robust method for obtaining dense range data from stereo with active lighting. The chapter concludes with a description of how spacetime analysis, a method that was developed for another type of scanner, was adapted to our system for more reliable and accurate scanning.

2.2 Physical setup

Our scanning system consists of the following main parts (see Fig. 2-1). Four Sony 107-A color video cameras are mounted on an aluminum bar. Each camera is equipped with manually adjustable zoom and aperture. The cameras are connected to a Matrox Meteor digitizing board, which can switch between the four inputs under computer control, and produces images at 640×480 resolution. The digitizing board is attached to a Pentium PC.

Figure 2-1 The scanner hardware. Four video cameras are attached to an aluminum bar, which rests on a tripod. A slide projector is placed on a turntable under the cameras. Table lamps provide adjustable lighting for capturing color images.

Below the cameras, a slide projector sits on a computer-controlled turntable. The slide projector emits a vertical stripe of white light, which is manually focused to the working volume.

The object to be scanned is placed on a light table that is covered by translucent plastic. When the lamps under the light table are turned on, the background changes; thus we can easily detect background pixels by locating the pixels that change color.[1] Next to the scanner is a set of adjustable lamps that are used to control the object illumination when we capture color images.

[1] In order for this to work the cameras have to aim down so they don't see past the light table.

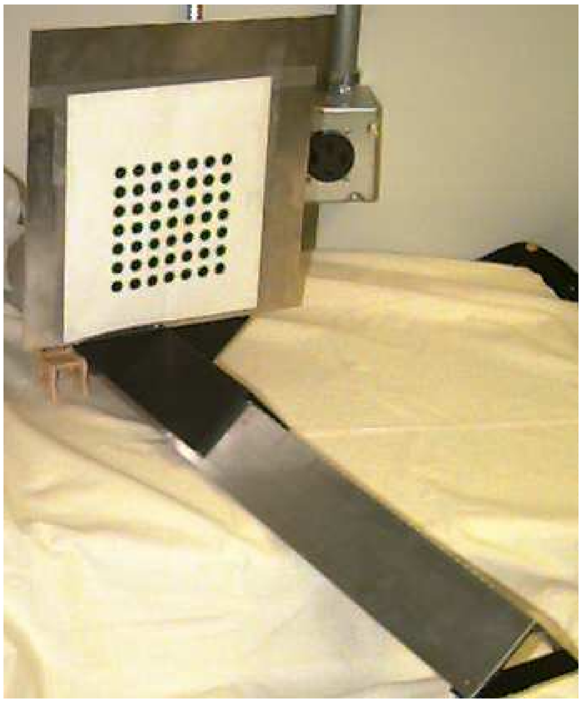


Figure 2-2 The calibration object is metal plate mounted on a rail. The plate is perpendicular to the rail, and it can be moved in 10 mm steps along the rail. A calibration pattern consisting of a 7×7 dot field is pasted on the plate.

2.2.1 Calibration

In order to make any measurements from camera images, we need to know the camera calibration parameters. Some of the internal parameters (size of the CCD array, aspect ratio of the pixels, etc.) are fixed, but others, such as focal length and distortion, vary. The external parameters define the pose of the camera, that is, the 3D rotation and translation of the optical center of the camera with respect to some fixed coordinate system.

We measure the calibration parameters for all four cameras simultaneously using Tsai's algorithm [1987] and the calibration object shown in Fig. 2-2. The calibration object consists of a metal plate that can be translated along a rail that is perpendicular to the plate. A calibration pattern made of a 7×7 block of circular black dots on a white background is pasted on the metal plate.

The calibration proceeds as follows. The calibration object is placed in front of the cameras so that all four cameras can see the whole pattern. The origin of the calibration coordinate system is attached to the center of the top left dot in the pattern, with the x-axis aligned with the dot rows, the y-axis with the dot columns, and the z-axis with the rail on which the calibration plate rests. The dots are located within the camera images using the Hough transform [Davies 90], and the 7×7 pattern is matched to those dot locations. The image coordinates of each dot are then paired with the corresponding 3D coordinates. The plate is moved by a fixed distance along the z-axis, and the process is repeated several times over the whole working volume. The calibration parameters for each camera are estimated from the entire set of 3D-2D point correspondences using Tsai's algorithm [1987].

2.3 Range from stereo

In traditional stereo vision one tries to match pixels in one camera image to pixels in the image of another camera [Barnard & Fischler 82]. The pixel matching relies on intensity variations due to surface texture. If the cameras have been calibrated and the matched pixels correspond to the same point on some surface, it is trivial to calculate an estimate of the 3D coordinates of that point.

There is an inherent tradeoff in the choice of the baseline (the distance between the two cameras): the longer the baseline, the more accurate the estimate, but the shorter the baseline, the more reliably the pixels can be matched. Okutomi and Kanade [1993] implemented a multiple-baseline stereo method that uses multiple stereo pairs with different baselines. The use of multiple stereo pairs provides improvement both in correctly matching pixels and in increasing the accuracy of the 3D coordinate estimation. Kang *et al.* [1995] later enhanced the "passive" method by projecting stripe patterns into the scene to help match images of untextured surfaces that do not exhibit intensity variations ("active" stereo).

We initially implemented the Kang *et al.* [1995] system for scanning range data. However, we had difficulties in reliably matching the camera images, especially if textures on the object surface interfered with the projected dense stripe pattern, and close to the silhou-

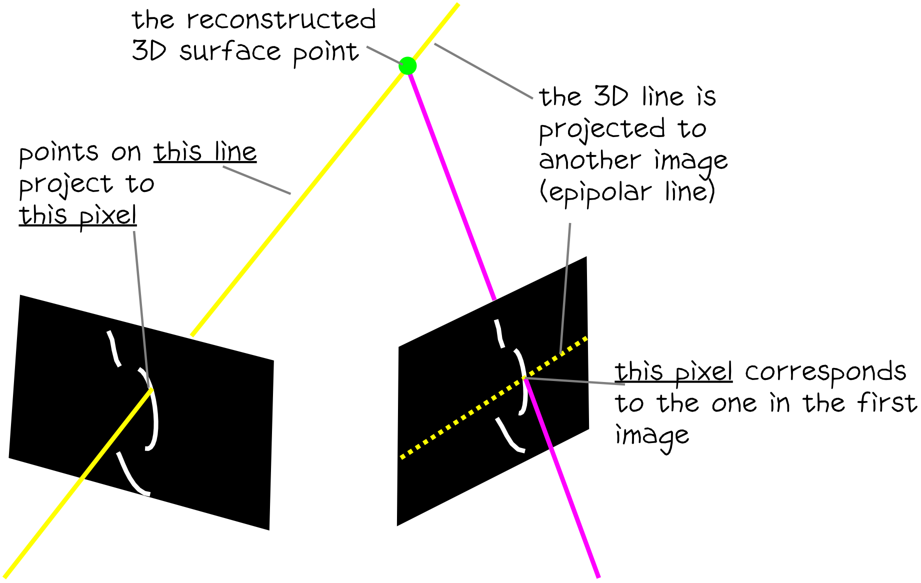


Figure 2-3 A vertical light beam is projected onto the scene resulting in stripes that generally appear curved and discontinuous when viewed from another direction. The stripes are located on both camera images. The 3D surface point visible through a pixel in the left image is found by projecting the 3D line associated with that pixel onto the right image, intersecting the projection with the detected stripe, and intersecting the 3D lines of the matched pixels.

ette of the object. As scanning speed was not important for us, we chose a much slower, but significantly more robust method that only requires two cameras and a single light stripe.

In our system we project only a single vertical light stripe onto the scene. The stripe appears in both camera images as shown in Fig. 2-3. The set of points projecting to a pixel forms a line in 3D, and we can calculate that line from calibration data. We match the stripe pixels by projecting the line corresponding to a pixel in the left image onto the right image, and then intersecting the projected line with the stripe in the right image. Once we have matched two pixels, we use the intersection of their corresponding 3D lines to determine the 3D coordinates of the surface point visible through the pixel in the left image. The whole scene is scanned by aiming the stripe at the left side of the working volume, reconstructing surface points illuminated by the beam, turning the beam to the right by a small fixed angle, and repeating the process until the beam has swept over the whole working volume.

The result is a dense range map, an image that stores the 3D coordinates of the first visible surface point through each pixel[2].

We can increase the reliability of our system by using more than two cameras. In the basic system we can only scan points that are visible from both cameras, as well as the light projector. If we use four cameras, we are satisfied if the surface point is visible from the first camera, the light projector, and any of the other cameras. In the case when the point is visible from several cameras, we can check whether the match was reliable by calculating the 3D coordinates of that point from several camera pairs and checking whether the answers agree. We can then either use the coordinates obtained from the stereo pair with the largest baseline, which is likely to give the most accurate estimate, or compute a weighted average of the answers.

2.4 Scanning color images

In order to detect the vertical beam projected onto the scene we have to turn all other lights off while scanning the object, and consequently we cannot capture reliable color information while scanning the geometry. Instead, we first scan the geometry, then turn the lights on (and turn the stripe projector off) and take a color image with the camera that is used as the base camera in range triangulation. This way the color and range data are automatically registered, i.e., each range measurement is associated with exactly one pixel in the color image.

2.5 Spacetime analysis

Our scanning method requires that we accurately locate the center of the stripe in a camera image. At first this seems quite easy; after all, the intensity distribution across the width of the stripe is approximately Gaussian, and simple techniques such as taking an average

[2] Actually, only for pixels whose surface points are visible from both of the cameras and the light source, and of those for ones over which the beam was actually centered.

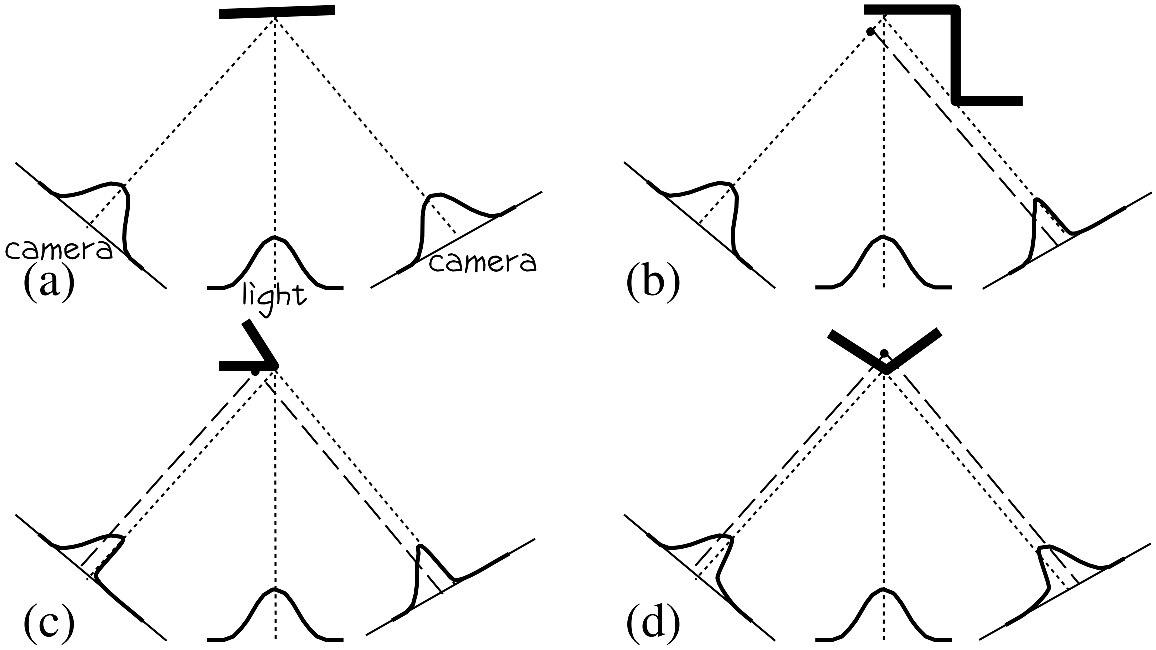


Figure 2-4 Sources of errors in range by triangulation. In all images the central vertical line is the light beam with Gaussian intensity distribution, on the left and right are two sensors. (a) Ideal situation: planar object. Both sensors see the stripe and the surface is scanned correctly. (b) The right sensor can't see the whole stripe because of self-occlusion, pulling the estimate forward. (c) The beam hits the surface only partially, pulling the surface estimate away from the corner, though not out of the surface. (d) The sensors see only half of the stripe well, resulting in too large a depth estimate.

weighted by pixel intensities across the beam should give good results. However, there is a hidden assumption that the surface illuminated by the beam is locally planar and that the whole width of the stripe is visible to the camera.

2.5.1 Problems in locating the stripe

Curless and Levoy [1995] noticed that when the assumptions of a planar surface and a fully visible stripe are violated, the range estimates become systematically distorted. They used a Cyberware scanner, which scans range data using a calibrated laser beam with a single calibrated camera. Figure 2-4 is adapted from Curless and Levoy's [1995] paper to a system of two calibrated cameras and an uncalibrated light beam, and it illustrates several config-

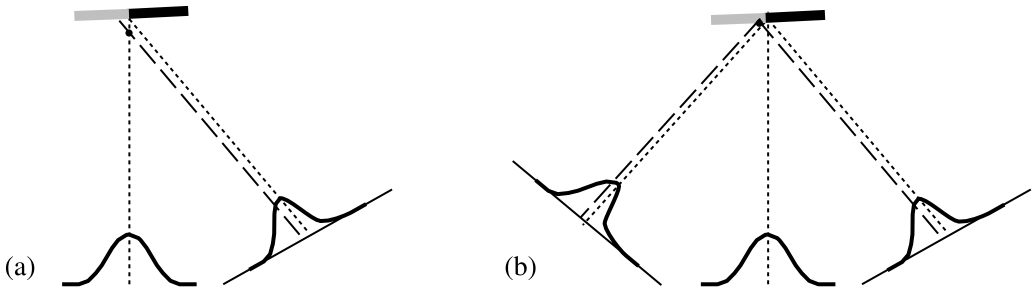


Figure 2-5 (a) Intensity changes lead to incorrect surface point estimate in a system using calibrated light source. (b) Intensity changes move the surface point estimate along the surface in a system using calibrated cameras and uncalibrated light source.

urations that lead to an incorrect estimate for the surface point coordinates. Figure 2-4(a) shows the ideal situation: a completely visible stripe on a flat surface, yielding an accurate range estimate. In Fig. 2-4(b) part of the beam is occluded in the right camera by a protuberance of the object. The estimate of the stripe center on the right camera is biased to the left, pulling the range estimate closer towards the left camera. In Fig. 2-4(c) the center of the beam is aimed at a sharp silhouette corner of the object, and only half of the beam hits the surface. The stripe center estimate of both cameras is biased to the left, giving incorrect 3D coordinates for the surface point on the silhouette edge. Because of this phenomenon, it is hard to scan the surface reliably all the way to the silhouette boundary. Figure 2-4(d) shows the beam pointing at a sharp crease on the surface. The left half of the stripe is fully visible to the left camera, whereas the right half of the stripe is foreshortened because the surface turns away. This results in a bias to the left in the stripe center estimate. By a similar argument, the stripe center is biased to the right in the right camera, resulting in error in scanning the surface point.

It is worth noting that Curless and Levoy presented a fourth case, where changes in surface intensity bias the stripe center estimate, leading to an error in the range estimate (see Fig. 2-5(a)). This is not really a problem in our system. Recall that while they use a calibrated light beam and a calibrated camera, we use an uncalibrated light beam and several calibrated cameras. In the case of a planar textured surface, the stripe center estimates would be similarly biased in both cameras. However, as shown in Fig. 2-5(b), the reconstructed point still lies on the surface.

2.5.2 Solution to the problem

The problems illustrated in Fig. 2-4 can be overcome by reformulating the task. Instead of trying to find *where* the center of the beam is in a single image, one can track the time-varying intensity profiles of the pixels to find out *when* the beam was centered at each pixel. Curless and Levoy [1995] call this approach spacetime analysis. Let's assume that the beam is wide enough to cover several pixels in the image, its intensity has Gaussian distribution, and we move the beam in steps that are small compared to the beam width. When the beam approaches the surface point visible through a pixel, the intensity of the pixel first increases, and then begins to decrease as the beam sweeps past the point. The shape of the time-varying intensity profile is always Gaussian, and the time when the beam was centered on the pixel can be reliably estimated from that profile.

Spacetime analysis produces a function describing when (if ever) the beam was centered on each image pixel. We can use this function in our triangulation method as follows. Choose a pixel in the left image and note the time when the beam was centered at that pixel. Find the epipolar line corresponding to that pixel on the right image as before; i.e., project the 3D line corresponding to the pixel onto the right image. The epipolar line defines a new function that maps a position on the line to a time when the beam was centered on the pixel under the line. To match the original pixel on the left image we simply search for a line position such that the pixel under it has the same time as the original pixel. The 3D coordinates of the surface point are found by triangulation as before. We implemented this method and thereby reduced the errors illustrated in Fig. 2-4.

2.6 Contributions

The contributions of this chapter are:

- Implementation of a working range and color scanning system that provides the data used in the following chapters.
- Adapting spacetime analysis to a scanner that uses calibrated cameras and an uncalibrated light beam.

Chapter 3

Registration

3.1 Introduction

Our scanner does not allow scanning all of the object without moving either the object or the scanner. In practise, the object is placed into the working volume in front of our stereo system, and the surfaces visible to the cameras are scanned. The object is then manually reoriented, so that other parts of the surface can be scanned, and this is repeated until most (ideally, all) of the object surface is scanned.

The range data in each view is expressed in the sensor coordinate system. Since the

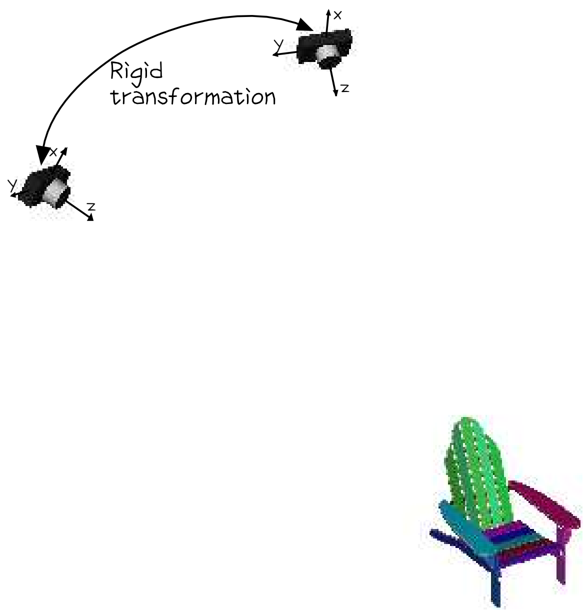


Figure 3-1 The range maps are registered by aligning the range data in the different views, each expressed in the sensor coordinate system. Registration of the data is equivalent to finding the rigid transformations of the sensor between the scans.

object was moved between scans, data points corresponding to the same surface point have different coordinates in different views. Clearly, if we want to estimate the object surface from the scans, we have to first align the data sets and express all data in a single, object-centered coordinate system. This process is called registering the range data. If we knew the relative motion of the scanner with respect to the object from view to view, we could register the data sets by applying that same transformation to them (see Fig. 3-1). However, even if we had a robotic system that moved the object (or the scanner) between the views, we could typically only expect to have an approximate transformation with accuracy much less than that of the scanner. But if we start from an approximate initial registration, and if there is sufficient overlap in the views, we can refine the registration by finding rigid 3D transformations that precisely align the overlapping range and color data.

This chapter presents new algorithms for registering range maps. We first describe our fast interactive method for obtaining a coarse initial registration. We then describe previous methods for pairwise registration of range maps, followed by our new method that makes the registration process more robust and uses color information to improve registration accuracy. We also present a new method for multiview registration that directly uses information obtained from pairwise registration, yet distributes the registration error evenly over all views. A short discussion concludes the chapter.

3.2 Traditional approaches to pairwise registration

If we have two range scans of the same object such that the range maps overlap, we can register the views by aligning their overlapping geometry. Figure 3-2 illustrates the general strategy in pairwise registration of range maps. In the upper left there are two scans of a couch, both expressed in the sensor coordinate system. First, we obtain an approximate alignment by interactively pairing a few (4) corresponding surface features. The initial registration is then refined by automatically pairing points in the surface scans and moving the data into better alignment using the paired points. These two steps are iterated until the pro-

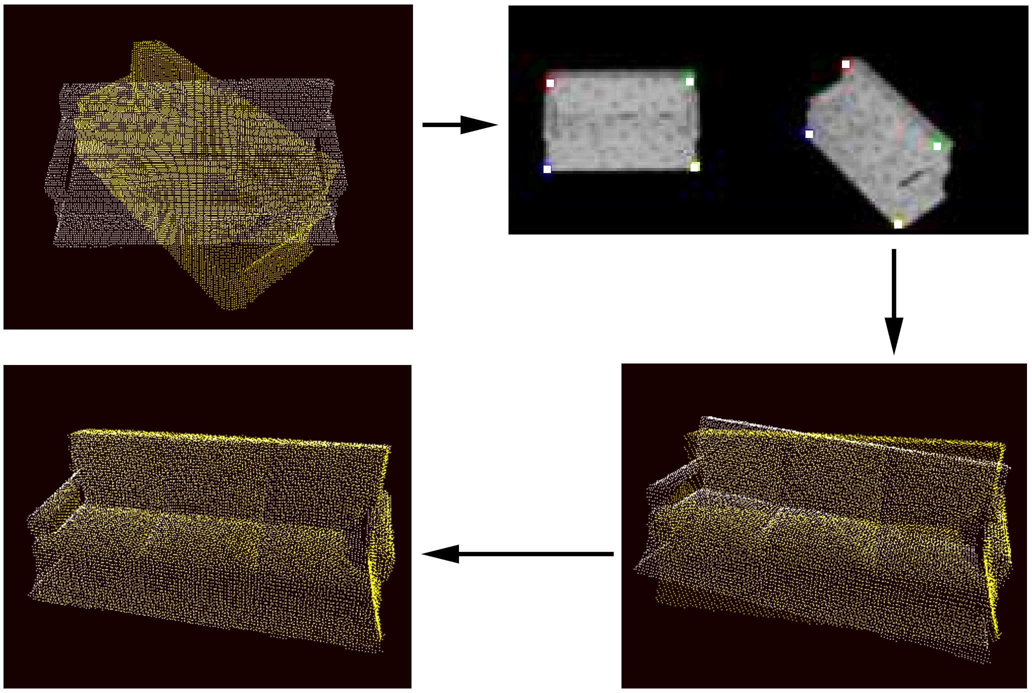


Figure 3-2 Initially the two views are expressed in the sensor coordinates rather than in an object-centered coordinate system. An initial registration is found by interactively matching small set of features and then aligning them. The initial registration is automatically refined.

cess converges.

A good registration can be found by minimizing the following equation:

$$\sum_{i=1}^{N_p} \parallel \mathbf{T}\mathbf{p}_i - \mathbf{q}_i \parallel^2, \quad \mathbf{q}_i = \arg\min_{\mathbf{q}\in\mathbf{Q}} \parallel \mathbf{T}\mathbf{p}_i - \mathbf{q} \parallel . \tag{3.1}$$

Here we are looking for a rigid 3D transformation $\mathbf{T}$ that minimizes the distances of the scanned points $\mathbf{p}_i \in \mathbf{P}$ to a set $\mathbf{Q}$ that can be either another set of scanned points, a surface fitted to a point set, or a complete surface model of the scanned object. Equation 3.1 is difficult to solve directly as it tries to solve several problems simultaneously. First, there is the transformation $\mathbf{T}$ that aligns the overlapping geometry. Second, there is the question of pairing the $\mathbf{p}_i$'s. If the surface point corresponding to $\mathbf{p}_i$ is not represented in $\mathbf{Q}$, $\mathbf{p}_i$ should not even be included in the sum 3.1. Otherwise, it should be paired with $\mathbf{q_i} \in \mathbf{Q}$ that corresponds to the same point on the surface. However, since it is easy to solve the transformation if the correct point pairings are given, and vice versa, the idea of iterating over two simpler subproblems is used in most practical registration methods.

We first describe our approach to initial registration, and we then study the state-of-the-art in rigid 3D registration, concentrating on the iterative pairing-minimizing approach.

3.2.1 Initial registration

Registration is a nonconvex optimization problem and can therefore easily converge to an incorrect local minimum if not given a good starting point. Therefore most registration methods require a rough initial registration which is then refined by a local optimization process. There are many possible ways to obtain an approximate initial solution. In a fully automated scanning and reconstruction system, either the object or the scanner has to be moved under computer control between scans to obtain views that cover the whole surface. In such a system, the relative motion of the scanner and the object is likely to be directly available. If, on the other hand, a human operator chooses the viewpoints and manually re-orients the object (or the scanner) between scans, it is not unreasonable to ask the operator to provide an approximation of the registration, provided there is an easy-to-use interface that requires no more than the effort expended to reorient the object. This is the case in our system.

Figure 3-2 illustrates our initial registration method. The user is shown images that were taken along with the range scans. She then selects four point pairs so that each pair marks the same surface point in both images.[1] The program then finds the 3D points corresponding to the selected points. In terms of Eq. 3.1, $N_p = 4$ and the user indicates the pairs $(\mathbf{p}_i, \mathbf{q}_i)$. Now as the pairs have been fixed, we only have to find the transformation $\mathbf{T}$ that aligns them. Slightly differing closed form solutions for finding the rigid Euclidean motion that minimizes the sum of squared distances between the paired points were published independently by Horn [1987] and Faugeras and Hebert [1986]. There is also a more recent formulation

[1] Three point pairs would be enough to determine the relative orientation and position, but with more points small errors in pairing have less influence on the solution.

by Wang and Jepson [1994] that according to the authors provides more robust solutions when the data are contaminated by anisotropic noise.

There have been many suggestions on how to obtain the initial registration without user interactions. Unfortunately, they are not robust, general purpose solutions that could be expected to work with more than a limited class of objects. We defer their discussion to Section 3.5.

3.2.2 Iterative solution

Besl and McKay [1992], Chen and Medioni [1992], and Zhang [1994] proposed minimizing Eq. 3.1 by iterating between pairing points and solving the transformation. Besl and McKay called their method the Iterative Closest Point (ICP) algorithm, and we use that name to describe all methods with this common approach. The objective function is similar to Eq. 3.1:

$$\sum_{i=1}^{N_p} \| \mathbf{T}_k \mathbf{p}_i - \mathbf{q}_i \|^2, \quad \mathbf{q}_i = \mathrm{Proj}(\mathbf{T}_{k-1}\mathbf{p}_i, \mathbf{Q}) \tag{3.2}$$

The estimate of the registration transformation $\mathbf{T}$ is refined in small steps. First, while holding the transformation fixed, the points $\mathbf{T}_0\mathbf{p}_i$ are projected onto surface $\mathbf{Q}$ and paired with their projections $\mathbf{q}_i$. Ideally, $\mathbf{p}_i$ and its pair $\mathbf{q}_i$ should correspond to the same point on the object surface. Now the pairs are held fixed, and a transformation $\mathbf{T}_1$ that brings $\mathbf{P}$ closer to $\mathbf{Q}$ is calculated. As the surfaces move closer, finding better point pairs becomes easier and more reliable, which enables finding better transformations, so projections and minimizations can be iterated until the process converges.

Pairing

It is difficult to find for each point $\mathbf{p}_i \in \mathbf{P}$ the point $\mathbf{q}_i \in \mathbf{Q}$ corresponding to the same 3D surface point. Furthermore, the corresponding point does not even exist for some $\mathbf{p}_i$.

Besl and McKay [1992] proposed a simple way of projecting points to another surface or point set: for each point, find the closest point in the other view. This is illustrated in

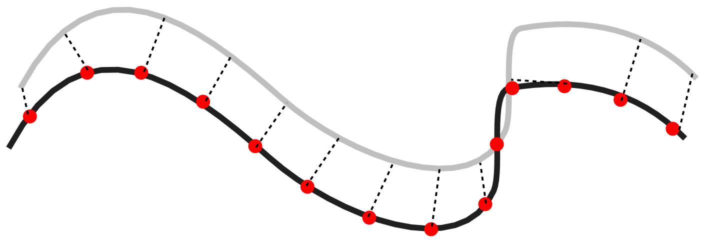


Figure 3-3 Pair each control point with the closest point in the other view.

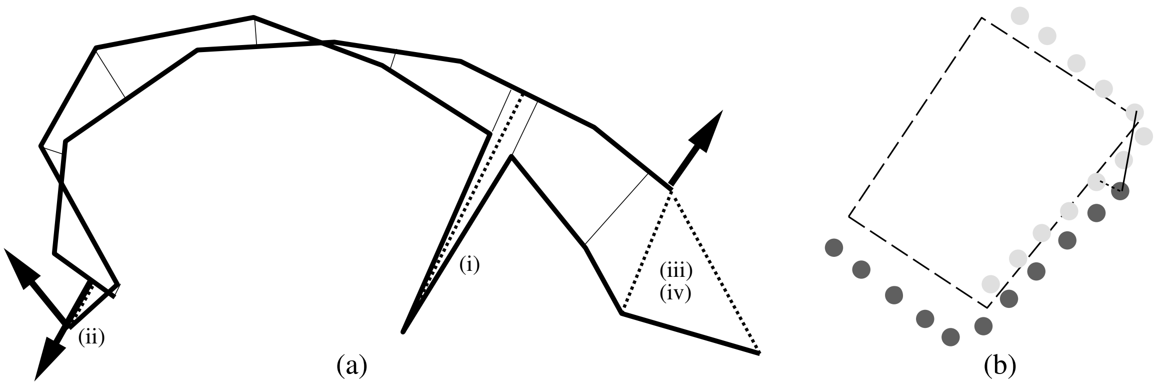


Figure 3-4 Point pairing heuristics. (a) Heuristics for discarding point pairs: (i) distance is too large, (ii) normals differ too much, (iii) pairing a point with a boundary vertex pulls the solution in the wrong direction, (iv) the point-to-point vector differs from the normal vector. (b) A point on an occluding boundary is paired with the closest point on a hidden surface (solid line), not with the closest point (dotted line).

Fig. 3-3, and it works best when the surfaces are close together and are relatively smooth. Chen and Medioni [1992] preferred a different approach that was originally proposed by Potmesil [1983]: extend the normal vector at the point $\mathbf{p}_i$ and locate the intersection of the normal and the surface $\mathbf{Q}$.

Both of those methods are only heuristics that often fail. Many researchers have proposed additional heuristics that either remove some of the pairs that are clearly wrong or help in finding pairs that are more likely to correspond to the same surface point. Figure 3-4 illustrates some of these heuristics. Gagnon *et al.* [1994] do not try to pair points with normal vectors pointing away from the viewpoint of the other view as the points cannot be visible. Turk and Levoy [1994] eliminate pairs where the points are too far apart. They also eliminate pairs where one of the points is on the silhouette of the range map, as many points in the other view that are not visible in the first view are paired with that point. A similar effect, however, can be achieved by Besl and McKay's heuristic of removing pairs in which the vector from one point to another does not agree with the local normal vector. Koivunen and Vezien [1994] first perform one registration step and then discard pairs if the points are far apart or if the local normals don't agree. Dorai *et al.* [1996] discard pairs if they are not compatible with the neighboring pairs. Take two pairs, $(\mathbf{p}_i, \mathbf{q}_i)$ and $(\mathbf{p}_j, \mathbf{q}_j)$. Unless the distance $\| \mathbf{p}_i - \mathbf{p}_j \|$ equals the distance $\| \mathbf{q}_i - \mathbf{q}_j \|$, it is not possible that both pairs are valid at the same time, since a rigid registration transformation preserves 3D distances. Dorai *et al.* filter the pairs and discard those that are not compatible with most of the neighboring pairs.

Godin *et al.* [1994] used a different approach. Instead of first pairing points and then culling away bad pairs, they define a compatibility function that is used to find better pairs for the points. In addition to the range data, they also had color data, and their compatibility function compares local surface reflectance properties. The compatibility of two points is based on their colors, and each point is paired with the closest compatible point. Though Godin *et al.* used only reflectance information, the compatibility function can be easily extended to take into account other features such as normal vectors, curvatures, etc.

Eggert *et al.* [1996] proposed a clever heuristic for finding good pairs for points on an occluding boundary. Each boundary point is paired with nearest point on the surfaces of the other views that is hidden from the current viewpoint. Figure 3-4(b) illustrates this with two views of a box. The dark points were scanned from the direction of the bottom corner, while the light points were scanned from the right. The rightmost dark point is an occluding boundary point, so it corresponds to the corner of the box. Most methods would pair that point with closest point in the other scan (dotted line), but this heuristic pairs the point with the closest point with a normal pointing away from the viewpoint of the current view (solid line). This pairing pulls the corners efficiently together. Other points are paired with the closest points, where the closeness is defined as a sum of the physical distance and a scaled difference in normal vector orientation. The influence of the normal vectors is decreased as the registration proceeds, because although using normals helps in the beginning, noise in

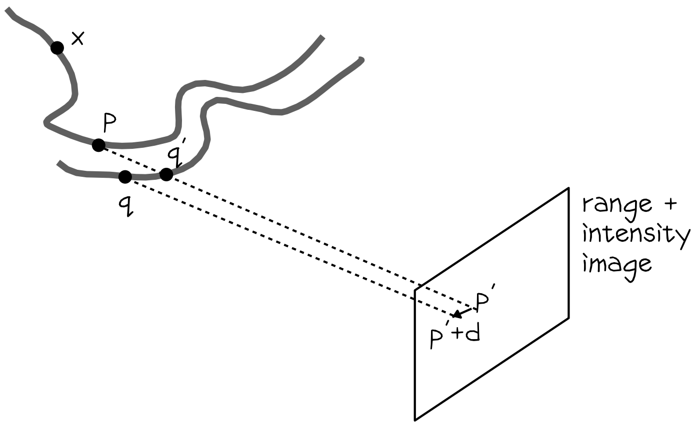


Figure 3-5 The point $\mathbf{p}$ is projected onto the image plane of the other view. The intensities of $\mathbf{p}$ and its projection $\mathbf{p}'$ are compared, and a correction displacement $\mathbf{d}$ is calculated using the intensity gradient at the projection. $\mathbf{p}$ is paired with $\mathbf{q}$, the point that projects to the corrected image location $\mathbf{p}' + \mathbf{d}$. Points that are not visible to the other scanner due to self-occlusions, such as point $\mathbf{x}$, are not paired with any other point.

the normal estimates hinders the final convergence.

The methods described above typically find a partner for a point by searching for the closest point in another set of points or small triangles. Although the search can be accelerated by organizing the data into a k-D tree [Friedman *et al.* 77] or into spatial bins, it can still take a large percentage of the processing time, especially since each range map can easily contain hundreds of thousands of points. Weik [1997] developed a method that avoids the closest point search by projecting points from one range map to another range map. As illustrated in Fig. 3-5, a point $\mathbf{p}$ is paired by projecting it to $\mathbf{p}'$ on the image plane of the scanner of the other view, and finding the point $\mathbf{q}'$ in the other view that projects to the same location. However, they also used image intensity information to find a slightly better pair for a point. The difference ΔI of the intensities $I_0(\mathbf{p})$ and $I_1(\mathbf{p}')$, as well as the intensity gradient $\nabla I_1(\mathbf{p}')$, are calculated. A displacement $\mathbf{d}$ is calculated such that $\Delta I = \nabla I_1(\mathbf{p}') \cdot \mathbf{d}$, and the original point is paired with the point $\mathbf{q}$ that projects to the new corrected image location $\mathbf{p}' + \mathbf{d}$. Weik also prevents some points from being paired if they cannot possibly be visible, due to self-occlusion in the direction of the other view point (point $\mathbf{x}$ in Fig. 3-5).

A few people [Zhang 94, Godin *et al.* 94] have noted that the point pairing should be

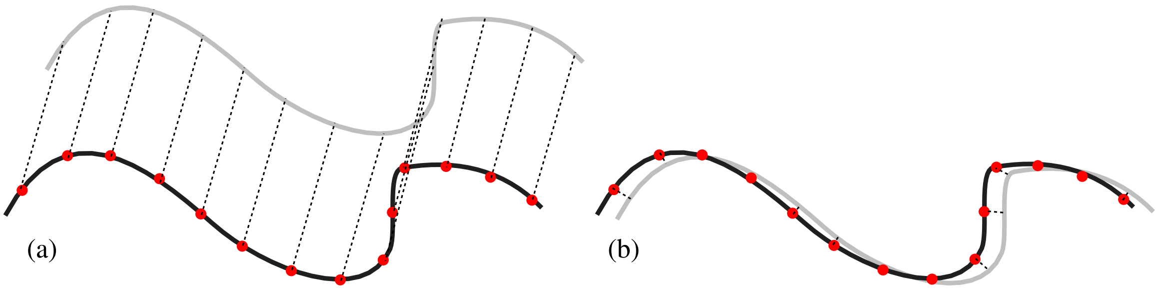


Figure 3-6 Fixed ideal springs. (a) Horn's [1987] method pulls surfaces together in one step, provided the paired points correspond to same surface points. (b) If many of the paired points don't correspond to the same surface points, many seemingly good pairs (short distances) resist motion in all directions.

symmetric. Unless it is known that one view is a proper subset of the other, there is no reason why one view should prevail over the other. The pairing can be made easily symmetric by first projecting points in **P** to **Q**, then points in **Q** to **P**, and finally merging the two sets of pairs.

Minimization

Figure 3-6(a) illustrates how Besl and McKay [1992] and Zhang [1994] bring the surfaces closer together: the sum of the squared distances of the paired points is minimized using Horn's algorithm. One can try to visualize the method by imagining that the paired points are connected by ideal springs that pull the surfaces together. The approach works well if the paired points really correspond to the same points on the surface; in this case the correct registration is obtained in one iteration. Typically, however, even a few bad point pairs pull the solution away from the perfect registration, producing a situation like the one illustrated in Fig. 3-6(b). The surfaces are in a fair registration, but they still need to slide a bit past each other. When the points are reprojected to the other surface and the pairs are connected with conceptual springs, several pairs pull the solution to the correct direction. However, most paired points are very close to each other and therefore resist motion in any direction, preventing the surfaces from moving more than a very small step, which slows the convergence near the solution.

Chen and Medioni [1992] minimize something other than the distances between the paired points. The first surface is moved so that points on it move closer to the tangent planes of their counterparts on the other surface. In terms of the ideal spring analogy, only the first end of the spring is fixed to a point, while the other end is free to slide along the other surface. No penalty is associated with surfaces sliding past each other, so larger steps are possible, and the registration is likely to converge faster.

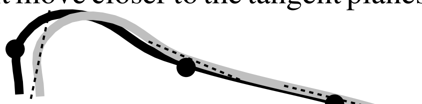


Figure 3-7: Sliding springs. Minimizing the distance from a point to a plane allows the left pair to pull the surfaces into alignment without the other pairs resisting tangential sliding.

Although the sliding spring approach typically converges faster, there is a situation where the fixed spring approach works better. If the point pairs are good, the fixed springs pull the surfaces to alignment immediately, while with sliding springs a good pairing may not move the surfaces at all if the points already lie in the tangent plane of their counterparts.

Masuda and Yokoya [1995] attempted to make Horn's minimization step of ICP more robust by using the *least median of squares* (LMS) approach [Rousseeuw & Leroy 87]. Horn's method minimizes the sum of squared distances between all given pairs, and it therefore has a breakdown point of 0%, meaning that even one wrong pairing can create an arbitrarily bad answer. The LMS estimator, on the other hand, minimizes the median of the squared residuals, and it can give the correct answer even if almost 50% of the points have been paired incorrectly. Full implementation of LMS requires selecting subsets of data points, finding the registration that aligns the subsets, calculating the median of the squared residuals for each view, and storing the answer that produces the smallest squared median. In practice, only a relatively small number of subsets are randomly chosen and evaluated. For each iteration of ICP, Yokoya and Masuda selected five points, paired them with the closest point in the other view, calculated the registration transformation using Horn's method, evaluated the result, repeated the process 200 times, and selected the best result. The approach still requires the same number of ICP iterations as other methods, typically around 50. The main benefit of the method is that reliable results can be obtained without actively culling bad point pairs. To speed up the otherwise expensive registration function, one data set is projected to the image plane of the other much like in Fig. 3-5 and the residuals are calculated along the projection direction. The authors claimed a reduction in computational costs as long as the size of the subsets (5) times the number of trials (200) does not exceed the size of the data set (n). However, this was based on the flawed assumption that finding the closest points has $O(n^2)$ cost, whereas with preprocessing the cost is only $O(n \ln n)$, so the described implementation is actually slower than traditional ICP. Further, as the method was described, a successful registration could be expected only if the two views share at least 50% of the points.

Some other alternative approaches for minimization are discussed in Section 3.5.

3.3 Projective pairwise registration

We now consider how the point pairing heuristics presented in the previous section fare in practice. Figure 3-8 shows some typical results. In Fig. 3-8(a) points from the dark curve are paired with the closest point on the light curve such that the local normals agree. The pairings in the ideal situation shown in Fig. 3-8(b) are consistently aligned, whereas in Fig. 3-8(a) some of the points are paired with a point above them, others have pairs below, to the left, or even to the right of them. In Fig. 3-8(c) points are paired with the closest point with matching color, pairing most points with a point directly above it, though some pairings slant to the left and others to the right. Not only are the pairings produced using local heuristics inconsistent, but almost none of the paired points actually correspond to the same point on the surface. It is clearly not enough to just throw away bad pairs; we should instead concentrate on finding good ones.

Before suggesting an alternative point pairing method, we list some desirable properties of good pairings:

- Each point and its partner should correspond to the same location on the surface, and only points from surfaces visible in both views should be paired.

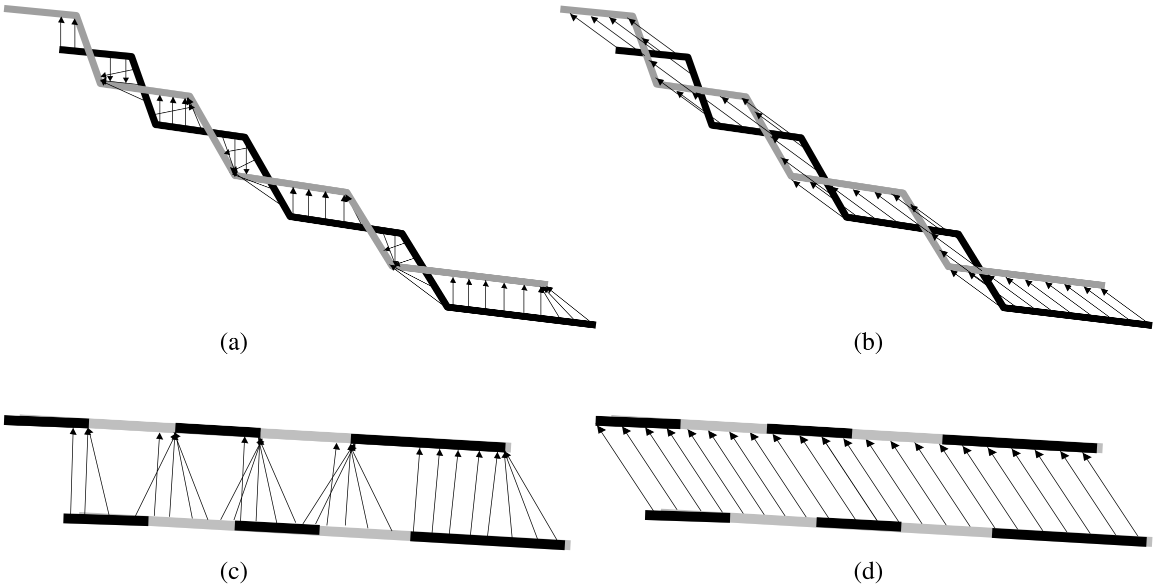


Figure 3-8 Heuristics lead to inconsistent pairings. (a) Points from the dark curve are projected to the closest point with compatible normal vector on the light curve. Some of the pairing vectors point up, some down, some to the left, and at least one to the right. (b) Points are projected to their true corresponding locations, and the pairs follow consistently a single rigid transformation, in this case translation. (c) Points on the lower two-colored curve are paired with the closest point with compatible color on the upper curve. Most pairing vectors point up, about as many are slanted to the right as to the left, while (d) shows that all should slant to the left.

- The mapping within the overlapping surface should be bijective, i.e., one-to-one (a point can be paired with only a single point) and onto (every point in the overlapping region should have a partner).

- The pairs should be consistently related with a single rigid 3D motion.

- The color data should be aligned as well as the geometry. In fact, surfaces that have rotational or translational symmetries cannot be unambiguously registered using geometric data alone [Gagnon *et al.* 94, Weik 97].

It is hard to create a pairing with the above properties using only local heuristics that myopically look for a closest compatible point without considering how the other points are paired. We propose a new, more global metric for simultaneously pairing the points on overlapping surfaces. The basic idea is to use 2D image registration to align the projections of the data sets to a plane and pair surface points projecting to the same image location.

Consider two textured surfaces that are already in close alignment. If you render the scanned surfaces as they would be seen from an arbitrary viewpoint (from which the overlap of the two surfaces is visible) the resulting 2D color images are also in alignment. Each point on surface A projects to the same pixel as its corresponding point on surface B. Assuming parallel projection, we could even translate one of the surfaces along the projection axis, and the corresponding points would project to the same pixels.

Now let's go back to our problem, where we have two views of a textured surface in rough alignment and would like to improve their registration. If we could move one of the views (partial surface) such that its projected image aligns well with the image of the other surface, we could be fairly confident that visible surface points projecting to the same pixel correspond to the same point on the object surface. We can then find good point pairs by pairing points that project to the same pixel. A brief examination confirms that all the properties that we listed as desirable are indeed satisfied with this pairing.

Figure 3-9 illustrates the basic idea in our point-pairing mechanism. In Fig. 3-9(a) one data set is rendered over the color image of the other data set, while in Fig. 3-9(b) the second data set is rendered over the color image of the first. In Figs. 3-9(c) and (d) the rendered images have been automatically aligned with the original color images. From the alignment we determine which image pixels in one image correspond to the pixels in the other image, and because we can find out which point on the mesh projects to a given pixel, we obtain a dense set of 3D point pairs. The point pairs are used to align the data sets using a robust version of Horn's [1987] method. The whole process can then be iterated until the data sets are closely aligned.

In the next section we describe how we use 2D registration of the color images to obtain better point pairs. We then discuss our robust approach for calculating the 3D registration of the data sets using those pairs. We conclude with some results and comparisons with a previous method.

Figure 3-9 Consistent pairing of points. (a) and (b): The data sets are roughly aligned. Each data set is rendered over the color image of the other data set. (c) and (d): The images are manipulated so they align better. Now points that project to the same pixel can be paired, and a better 3D registration can be calculated from the paired points.

3.3.1 2D Image registration

We want to align the rendered image of a range and color scan over the color image of another scan. We already mentioned one possible way to do this: translate and rotate the data in 3D so that when viewed from the viewpoint of the other camera it aligns with the color image taken from that point. However, if the data sets are already in rough alignment we don't need to search the 3D motions and repeatedly reproject the data. Instead, we can use 2D image registration techniques to register the rendered image directly with the color image.

Szeliski and Shum [1997] described a method for registering color images using planar perspective warping[2]. The planar perspective transform is described by

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \mathbf{M}\mathbf{x} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{3.3}$$

which maps image point (x, y) to $(x'/w', y'/w')$. The parameters m_i are found by iteratively updating the transform matrix using

$$\mathbf{M} \leftarrow (\mathbf{I} + \mathbf{D})\mathbf{M} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} d_0 & d_1 & d_2 \\ d_3 & d_4 & d_5 \\ d_6 & d_7 & 0 \end{bmatrix} \right) \mathbf{M}. \tag{3.4}$$

A new transformation $\mathbf{x}'' = (\mathbf{I} + \mathbf{D})\mathbf{M}\mathbf{x}$, is determined by minimizing

$$E(\mathbf{d}) = \sum_i \left[\mathbf{w}_i \left(I_1(\mathbf{x}''_i) - I_0(\mathbf{x}_i)\right)\right]^2, \tag{3.5}$$

where I_0 and I_1 are the two images, $\mathbf{w}_i$ is a per-pixel weight, and $\mathbf{d} = (d_0, \ldots, d_7)$ is the incremental update parameter. Truncation of the Taylor series expansion around zero gives the following approximation:

$$E(\mathbf{d}) \approx \sum_i \left[\mathbf{w}_i \left(\mathbf{g}_i^T \mathbf{J}_i^T \mathbf{d} + \mathbf{e}_i\right)\right]^2, \tag{3.6}$$

[2] A planar perspective warping can precisely align images of a surface if the surface is planar. However, when the surfaces are far from the camera and only small rotations are involved, the approximation can be good.

where $\mathbf{g}_i^T = \nabla I_1(\mathbf{x}_i')$ is the image gradient of I_1 at $\mathbf{x}_i'$, and $\mathbf{e}_i = I_1(\mathbf{x}_i') - I_0(\mathbf{x}_i)$ is the current color error. $\mathbf{J}_i = \mathbf{J}_d(\mathbf{x}_i)$ is the Jacobian of the warped point $\mathbf{x}''$ with respect to $\mathbf{d}$

$$\mathbf{J}_d(\mathbf{x}) = \frac{\partial \mathbf{x}''}{\partial \mathbf{d}} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix}^T . \tag{3.7}$$

The least-squares minimization problem Eq. 3.6 can be solved for example by completing

$$\mathbf{w}_i \mathbf{g}_i^T \mathbf{J}_i^T \mathbf{d} = -\mathbf{w}_i \mathbf{e}_i \tag{3.8}$$

to normal equations $\mathbf{Ad} = -\mathbf{b}$ by multiplying both sides of Eq. 3.8 with $\mathbf{J}_i \mathbf{g}_i$, where

$$\mathbf{A} = \sum_i \mathbf{w}_i \mathbf{J}_i \mathbf{g}_i \mathbf{g}_i^T \mathbf{J}_i^T \tag{3.9}$$

is an 8×8 matrix and

$$\mathbf{b} = \sum_i \mathbf{w}_i \mathbf{e}_i \mathbf{J}_i \mathbf{g}_i . \tag{3.10}$$

$\mathbf{d}$ is solved best by applying Cholesky decomposition for positive semidefinite matrices to $\mathbf{A}$ (p. 147 [Golub & Van Loan 96]).

The previous derivation was for a single color channel. In order to calculate a weighted least-squares solution where each color channel has weight $\mathbf{w}_j$, we have

$$\mathbf{A} = \sum_i \mathbf{w}_i \mathbf{J}_i \left(\sum_j \mathbf{w}_j \mathbf{g}_{i,j} \mathbf{g}_{i,j}^T \right) \mathbf{J}_i^T \tag{3.11}$$

and

$$\mathbf{b} = \sum_i \mathbf{w}_i \mathbf{J}_i \sum_j \mathbf{w}_j \mathbf{e}_{i,j} \mathbf{g}_{i,j} , \tag{3.12}$$

where j is the color channel index.

Hierarchical search

The 2D image registration method described above can find only a local minimum for $\mathbf{M}$ minimizing the image registration error (Eq. 3.6). The optimal registration transformation is unlikely to be found if the transformation moves pixels of the image more than the extent of the gradient operator (the gradients are estimated from the input images by convolving the image with, e.g., Sobel operator [Klette & Zamperoni 96]). We can make the implementation more robust by solving the problem in a hierarchical fashion, which makes this typically very nonconvex problem more convex.

We begin the hierarchical image registration by building image pyramids for the input images. At the top of this inverted pyramid is the original image, and below it are successively coarser versions. In our implementation the image pyramid is five levels deep and a pixel in a coarse image corresponds to four pixels in the image directly above it. We first register a low resolution version of the image, increase the resolution, and continue the process until the finest level of resolution is reached. Note that when moving to a finer resolution, a pixel with coordinates (x, y) gets new coordinates $(2x, 2y)$, and the transformation $\mathbf{M}$ has to be modified by multiplying m_2 and m_5 by 2 and dividing m_6 and m_7 by 2. That way we obtain from Eq. 3.3

$$\begin{bmatrix} 2x' \\ 2y' \\ w' \end{bmatrix} = \begin{bmatrix} m_0 & m_1 & 2m_2 \\ m_3 & m_4 & 2m_5 \\ m_6/2 & m_7/2 & 1 \end{bmatrix} \begin{bmatrix} 2x \\ 2y \\ 1 \end{bmatrix}. \tag{3.13}$$

Dense textured meshes

The images that we register are obtained by rendering two approximately registered dense textured meshes from the same viewpoint. We get those meshes from the original data using the following procedure. First, we label the images as background and foreground pixels based on known background color. Treating each pixel with valid foreground data point as a vertex, we perform a Delaunay triangulation [Guibas & Stolfi 85]. Triangles containing background pixels are removed, along with triangles that span step edges, i.e., connect an occluding surface with the occluded one. To detect step edges, we use a heuristic that takes into account both triangle edge length and triangle orientation. Finally, we organize the mesh into triangle strips for efficient rendering, and texture map it with an associated color image.

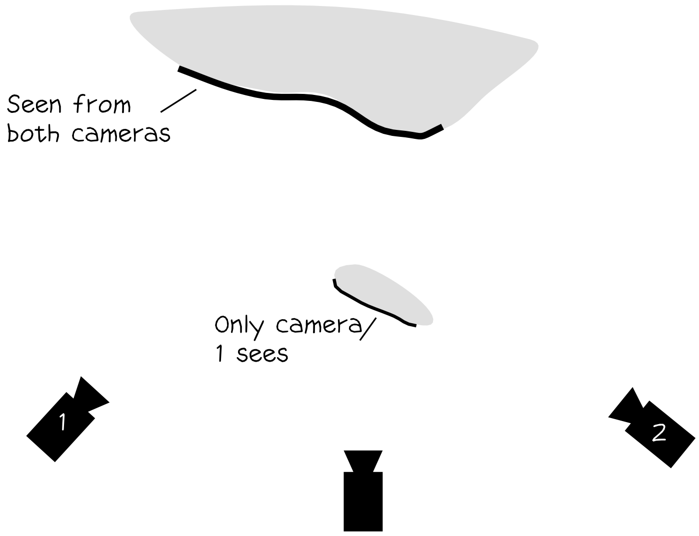


Figure 3-10 Part of the object only visible from camera 1 hides some of the surface visible from both cameras when viewed from a third direction.

Projection directions

We want to render the textured meshes from a viewpoint where as much of the overlap, i.e., surface that is scanned in both views, is visible. By definition, all of the overlap is visible to both scanners, which gives us the scanner positions of both of the views as good candidates for the rendering viewpoint. No other viewpoint can be any better in the sense that more of the overlap would be visible. However, many other viewpoints, such as the point between the two scanner viewpoints (see Fig. 3-10), are typically worse, as parts of the overlapping surface may be occluded by some other part of the surface.

We render the views twice, once from each scanner position. As illustrated in Fig. 3-9, the two sets of images are registered separately, producing two sets of point pairs. This way both views are treated symmetrically, and neither view prevails over the other.

Projecting to original images

With our choice of the rendering viewpoints, we don't need to render both of the meshes. Instead, we can project one of the textured meshes (I_0) directly to the color image of the other view (I_1). This leads to a speedier and more robust implementation. The most important improvement is more accurate image gradients. Recall that the image gradient $\mathbf{g}_i^T$ is calculated on the image I_1. If we calculate the gradients directly from an image of a mesh, the gradients close to mesh boundaries become suspect as part of the gradient convolution kernel includes background pixels. When we calculate the gradients directly from a complete color image, every pixel has a meaningful color value, even if the scanner failed to return valid range data, leading to correct gradient values.

Choosing the pixel weights

As shown in Fig. 3-9, the rendered meshes represent only parts of the visible object surface. We only want to align the parts of the images that display portions of the surface that are represented in both meshes. Using the notation of Section 3.3.1, let I_1 be the image of the mesh that was scanned from the current viewpoint, and I_0 be the image of the other mesh rendered from the same viewpoint. Ideally, I_0 is a subset of I_1. We align only that subset with image I_1 by setting the weights w_i to zero at pixels in image I_0 that do not contain data. In order to make the weight function smooth, the pixels well inside the valid data in I_0 have weight one, while the pixels closer to boundaries have a smaller weight that linearly diminishes towards the boundary.

Choice of color space

It is not obvious which color space one should use for registering the color images. Szeliski and Shum [1997] used the RGB space and didn't discuss any alternatives. In our scanning system the lighting is fixed while the object is reoriented between the scans, which means that different scans see the same object in different lighting conditions. Typically the colors remain approximately the same, while the intensity varies, especially when different parts of the object are shadowed. It would therefore seem to be desirable to work in a color space that separates these components, in order to downweight the relative importance of the intensity component.

We experimented with two alternatives to RGB. Our first choice was HSV (Hue Saturation Value) [Foley *et al.* 90]. The first component, hue, maps colors with one of the RGB components to a circle. The second component, saturation, tells how pure the color is, or how much apart it is from white, while the third component, value, codes the intensity of the color, or the distance of the color from black. The color space was designed for intuitive determination of the color, but it didn't prove to be suitable for color image registration. The main problem was that at low levels of intensity, slight changes in RGB values cause a large change in both hue and saturation. Also, if the object has any shades of gray, the hue component becomes indeterminate. We attempted to address that problem by scaling the saturation by value (intensity), but we clearly cannot do the same with hue as that changes the color. Experiments with HSV produced much poorer registration results than with RGB.

Our second attempt was rgI, which relates to RGB through the following simple equations: $r = R/(R+G+B)$, $g = G/(R+G+B)$, $I = (R+G+B)/3$. The r and g components contain the color information, while I is directly the intensity. This color model, although better, exhibited similar problems to HSV: the r and g components become unstable as I becomes small. This instability can be alleviated by scaling the color components towards gray with low intensity levels. Nevertheless, the RGB model fared better in practice.

Pairing the points

The 2D image registration provides us with a transformation matrix $\mathbf{M}$ that warps image I_1 to I_0; alternatively, we can think of image I_0 being warped to I_1 by the inverse transformation $\mathbf{M}^{-1}$. Also, we can determine the 3D coordinates for a mesh point projecting to a given pixel from the z-buffer and rendering parameters. Given the z-buffers of two meshes rendered from the same viewpoint, and the transformation $\mathbf{M}^{-1}$, we create 3D point pairs as follows. Consider pixel $\mathbf{x}$ in the image I_0. That pixel is associated with pixel $\mathbf{M}^{-1}\mathbf{x}$ in I_1. If according to the z-buffer information both $I_0(\mathbf{x})$ and $I_1(\mathbf{M}^{-1}\mathbf{x})$ contain valid data (not background), pair the 3D mesh points projecting to those pixels. As explained before, to ensure symmetry in registration we repeat the process of rendering data over a color image, registering the images, and pairing surface points from the viewpoint of the second camera.

3.3.2 Minimization

After 2D image registration most of the point pairs correspond to almost the same surface point, but we can't expect that to be the case with all pairs. With some pairs, the distance between the paired points can be large. An example of such a situation is close to a step edge, where a point from the occluding surface may project to the same pixel as a point from the occluded surface. If we proceed to minimize the sum of squared pairwise distances, the bad pairings can have an arbitrarily large influence on the result.

We could try to deal with bad point pairs as is typically done in many ICP implementations and use various compatibility thresholds to filter out the bad pairs. However, it is far from obvious how to choose the thresholds. Instead of thresholding,we combine Horn's least squares point set alignment method with a robust statistical method similar to the LMS (least median of squares) described earlier. LTS (least trimmed squares) [Rousseeuw & van Zomeren 90] has the same breakdown point as LMS, and can correctly fit a function when up to 50% of the data is contaminated by outliers, but it has a better convergence rate and a smoother objective function than LMS. In LTS we try to find a 3D rigid transformation such that the sum

$$\sum_{i=1}^{h} (r^2)_{i:n} \tag{3.14}$$

is minimized, where $(r^2)_{1:n} \leq \cdots \leq (r^2)_{n:n}$ are the ordered squared distances of points projecting to the same pixel after the transformation, n is the number of pixels, $h = \lfloor n/2 \rfloor + \lceil p/2 \rceil$, and p is the number of data points required to determine a solution (in this case 3). LTS is implemented as LMS by using stochastic sampling. We randomly select N_S point pairs and using Horn's method find the 3D rigid transformation that best aligns the chosen points. The transformation is evaluated by applying the transformation to the views, projecting the data to the same image plane, pairing points projecting to the same pixel, and evaluating the sum of Eq. 3.14. This process is repeated N_I times, and the transformation producing the smallest truncated sum is chosen as the answer. The probability that at least one sampling consists only of inliers is

$$1 - (1 - (1 - \epsilon)^{N_S})^{N_I}, \tag{3.15}$$

where ϵ is the rate of outliers. We evaluate each trial as follows. Again, let I_1 be the image of the mesh that was scanned from the current viewpoint and I_0 be the image of the other mesh rendered from the same viewpoint; ideally, I_0 is the subset of I_1. We first rerender I_0 using the registration transformation of the current trial. Then, we evaluate the pairwise distances over all the pixels in I_0 that contain data by pairing points from the two meshes that project to the same pixel. If there is no data for that pixel in I_1, we set the distance to infinity. We finally calculate the sum in Eq. 3.14, and if the sum turns out to be the smallest so far, we store the transformation of the current trial. After N_I trials the one with the smallest sum is applied to the data.

Having obtained an estimate for the registration using LTS, we try to improve the result by finding a least-squares solution using all good point pairs. We pair the points that project to the same pixel, and keep only the pairs that are within 2.5 times the standard deviation of the current fit. Rousseeuw and Leroy [1987] give a formula for estimating the standard deviation from the median of squared residuals:

$$s^0 = 1.4826 \left(1 + \frac{5}{n-p}\right) \sqrt{\mathrm{med}_i r_i^2}. \tag{3.16}$$

The factor 1.4826 is used to make s^0 a consistent estimate of the standard deviation when the r_i are normally distributed. The second term, which approaches 1 when the sample size n grows, is a correction term for small samples.

If, for example, 80% of the data are inliers, it is far better to sum the 80% smallest squared residuals when calculating the quality of an LTS trial, rather than only 50% as the basic method advises. At the same time we estimate the standard deviation, we also calculate which percentage of the pairs are inliers, that is, how many of them are within the threshold of $2.5s^0$. We use this estimate in the next round of LTS trial evaluations.

Using a robust method such as LMS or LTS to register two range maps is based on the premise that many of the points for which a counterpart in the other view exists are paired with the correct or almost correct counterpart. In our case, after 2D registration of the images this assumption holds, and we often obtain good registration in a single step. In Masuda and Yokoya's [1995] system the points are paired using the closest point heuristic, so in early iterations almost no point is paired with its true counterpart. Thus finding a good registration becomes extremely unlikely, and one can only hope to improve the current registration by a small amount. Whereas our method typically converges in a few (3-5) iterations of pairing and minimization, their method often requires on the order of 50 iterations.

3.3.3 Summary of the algorithm

Figure 3-11 gives pseudocode for the complete registration algorithm. The algorithm loops between finding point pairs and minimizing the distance between the data sets as long as the result seems to improve. The points are paired by rendering the textured mesh of one of the views over the color image of the other view, registering the 2D images, and pairing the points that project to the same pixels. The process is repeated by interchanging the roles of the views. The minimization part consists of two parts. First, a robust estimate of the 3D registration is found using the least trimmed squares method (LTS). The LTS part is repeated `N_i` times and the best result of all tries is stored. In each try, `N_s` of the pairs are randomly selected, a least squares (LSQ) solution is found using the pairs and Horn's algorithm, and the solution is evaluated by rerendering the meshes, pairing points, and summing up the `P` percentage of the smallest distances. We used `N_i` and `N_s` 20 and 8, respectively, yielding .975 probability that at least one try contains only inliers, with the assumption that 80% of the paired points are inliers. `P` is initially 50%, but the estimate is updated in the final part of the minimization algorithm. After the best LTS transformation `T_lts` is found, it is applied to the data, and the result is refined by doing a least squares minimization on the inliers. For this purpose a new set of point pairs is found, again by pairing points projecting to the same pixels. The median residual is found, the standard deviation is estimated using the

```
WHILE (LTS error reduces OR P grows)

    PAIR POINTS
        FOR viewpoints 1 and 2
            render textured mesh of the other view
            2D registration of color images
            pair points at same pixel

    MINIMIZE
        LTS
            repeat N_i times
                choose N_s pairs
                LSQ estimate of T_lts
                evaluate
                    render geometry
                    pair points at same pixel
                    sum h = P*n smallest squared distances
                save T_lts if best so far
        LSQ
            apply T_lts
            render geometry
            pair points at same pixel
                estimate st.dev. (Eq. 3.16)
                accept if closer than 2.5 st.dev.
                calculate new P = #inliers/n
            LSQ estimate of T_lsq
            apply T_lsq
```

Figure 3-11 Registration algorithm pseudocode.

Eq. 3.16, and the pairs within 2.5 standard deviations are treated as inliers. At the same time the percentage of the inliers is estimated, and `P` is updated. Finally, an LSQ transformation estimate `T_lsq` is found using the inlier point pairs and applied to the data.

3.3.4 Experiments

We have tested our registration method with a large number of range scans[3], and all the data used in the later chapters of this dissertation were registered using this method. In this section we present the results of registering two range scans using both our new method and

[3] 14 scans in a husky dog data set, 17 scans in a flower data set, 8 scans in a chair data set, and some additional experiments with both real and synthetic data.

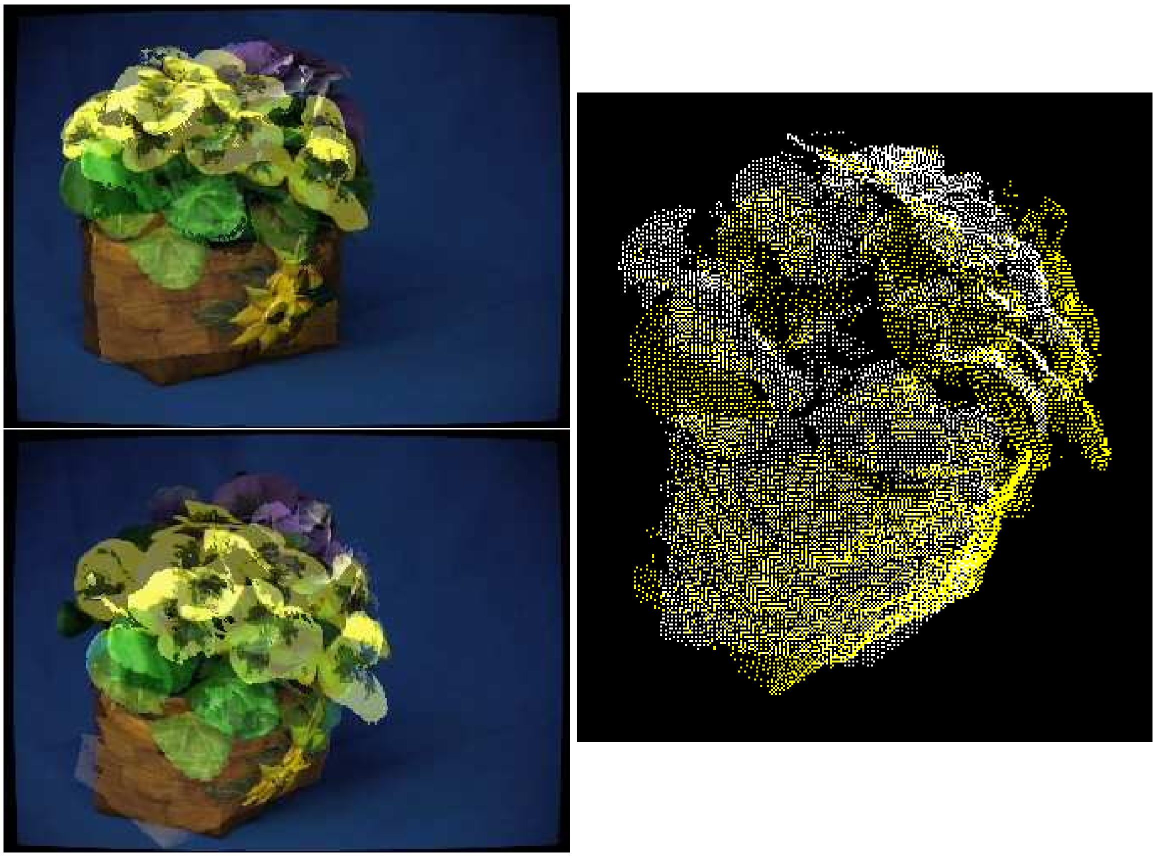


Figure 3-12 The initial registration configuration. On the left, the textured mesh of one scan is superimposed on the camera image of the other scan. The raw point data is displayed on the right.

using the modified ICP method that we used previously.

It is difficult to perform quantitative comparisons of two registration methods if the ground truth, the true registration transformation, is not known. Reporting the objective function values is not necessary meaningful since those values only indirectly reflect the quality of the registration. For example, with two views of a planar surface, the ICP metric of finding for each point the closest point in the other set or on the other surface could evaluate to zero even when the data sets are far from perfect registration. We first report some qualitative results on our experiments with real data sets, and finally some quantitative results from experiments with synthetic data sets where the ground truth is known.

Qualitative comparisons with real data

Figure 3-12 shows the starting configuration. The images on the left show a textured mesh of one scan superimposed on the camera image of the other scan, while the raw point data is displayed on the right.

Figure 3-13 illustrates the new registration method. Images (a), (b), (c), (g) form a sequence as seen from one camera, (d), (e), (f), (h) as seen from the other camera. The first image pair ((a) and (d)) shows the result when the images in Fig. 3-12 have been registered using the 2D color image registration. The second image pair ((b) and (e)) shows the result of 3D registration using the point pairs found by matching surface point projecting to the same pixel in Figs. 3-13(a) and (d). The second iteration of the 3D registration method begins with the 2D registration of the color images ((c) and (f)). On the third iteration the results do not seem to improve, and the final results are displayed in Figs. 3-13(g) and (h).

We also tried registering the same data sets from the same starting configuration using a modified ICP. The points are paired with the closest matching point in the other data set, where the matching criterion is that the points' normal vectors differ at most 45 degrees. Figs. 3-14 (a) & (b) show the final result when only pairs that are shorter than 25mm (the diameter of the flower basket is about 300mm) are used. A quite obvious misalignment remains when the method converges after 60 iterations (one iteration registers first one set to another and vice versa). Figs. 3-14 (c) & (d) show the final result when we use a schedule of diminishing thresholds. We begin by accepting pairs shorter than 30mm. After a few iterations the threshold is lowered first to 25mm, then to 10mm, to 5mm, and finally to 2.5mm. The results are now much better than in Figs. 3-14 (a) & (b), but the data sets are still in worse alignment than Figs. 3-13 (g) and (h). We finally tried to start from the same initial configuration and only accept pairs shorter than 2.5mm, but then the algorithm got stuck in a local minimum close to the starting configuration.

Figure 3-13 Registration results of the new method. Images (a), (b), (c), (g) form a sequence as seen from one camera, (d), (e), (f), (h) as seen from the other camera. (a) & (d) The result of 2D image registration from Fig. 3-12. (b) & (e) The result after a 3D registration is found using the point pairs found through 2D registration. (c) & (f) The result of 2D registration on images (b) and (e). (g) & (h) The final result after 3 iterations.

Quantitative comparisons with synthetic data

It is not obvious how to evaluate the resulting registration transformation. Many researchers simply evaluate the ICP function, that is, they pair each point with the closest point on the other surface, remove some of the pairs as outliers, and calculate a root mean square error. If the result is within the accuracy of the scanner, it is concluded that the correct registration was found. This is unwarranted for two reasons. First, the error is easy to bring down by discarding the longer pairs as outliers. Second, with many objects one can slide one data set along the other one with little penalty as most surface locations remain close to the other surface.

If one knows the correct registration the evaluation becomes easier. Now it is possible to directly compare the registration transformations. There is a problem, however. While it is possible to compare the rotations, the error in the translation component depends on both the rotation error and the choice of the origin of the coordinate system. Depending on that choice, the translation error can vary arbitrarily.

Our choice is to calculate the average deviation of the data points from their true positions after registration. For each point, we calculate the 3D coordinates after the true registration transformation and the recovered registration transformation, and calculate the average distance between the two positions.[4] This way of expressing registration quality is intuitive, especially if the data and error are scaled so that the scanned geometry just fits to a unit box.

Figure 3-15 shows the results on three synthetic data sets. For that purpose we created a viewer that can view and store color and range scans of any OpenInventor models, along with a registration transformation that aligns the data with those of the other scans. The results of our new projection-based registration method are compared to an ICP method. As before, in the ICP method points are paired with the closest point in the other set with a matching normal vector. Pairs with length exceeding an interactively determined threshold

[4] With nonuniform range sample distributions one could weight the individual errors with the approximate surface area that the sample represents and divide the sum by the total area.

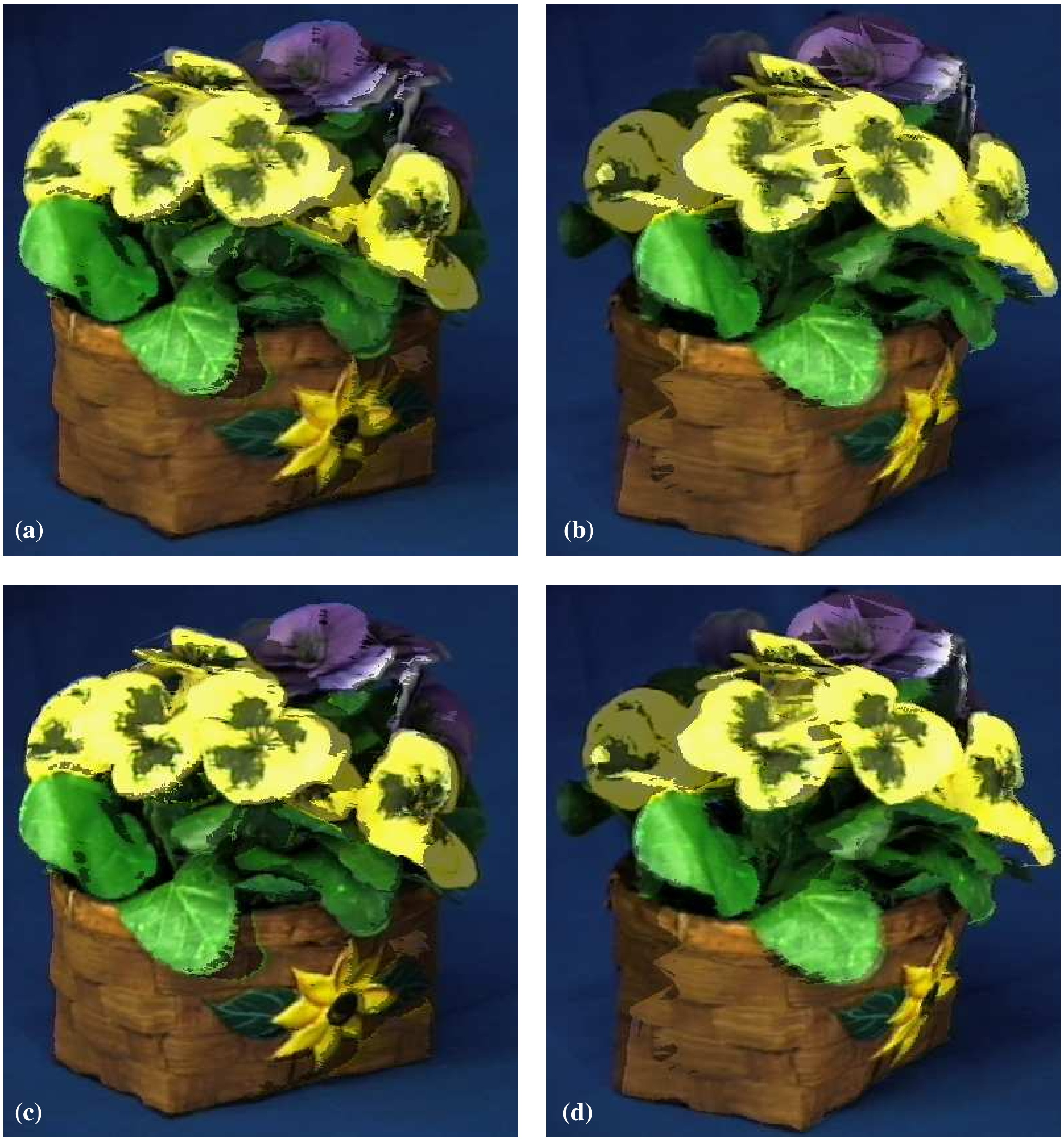


Figure 3-14 Registration results of ICP. (a) & (b) The result of ICP when accepting all point pairs shorter than 25mm. (c) & (d) The result of ICP that first accepts all the point pairs shorter than 30mm, then only shorter than 25mm, then 10mm, 5mm, and finally 2.5mm.

are removed. The threshold begins large, but is reduced as the registration progresses.

The horizontal axis in the diagrams of Fig. 3-15 denotes the number of iterations, while

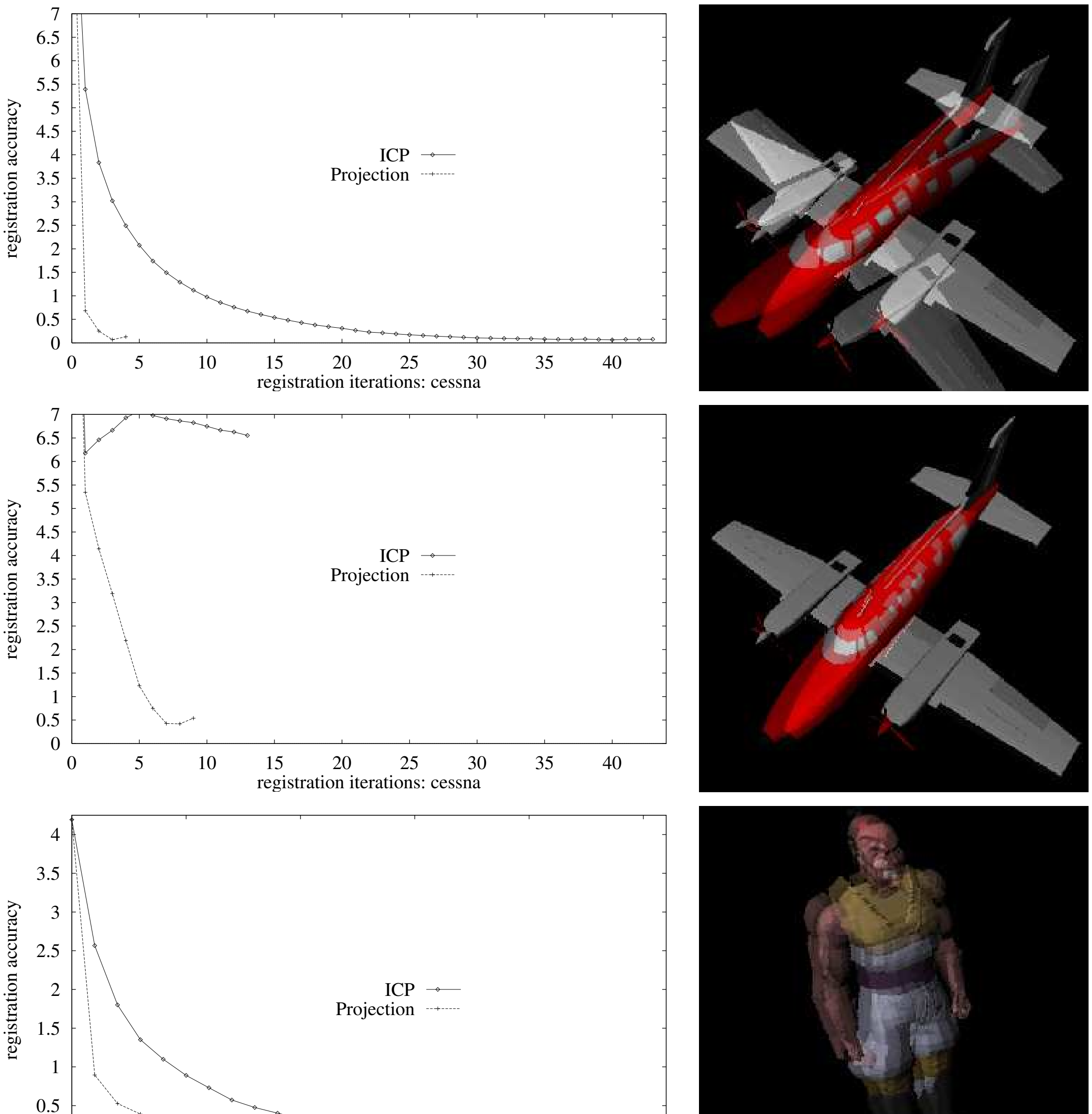


Figure 3-15 Experiments with synthetic data sets. The images on the right show the starting positions.

the vertical axis denotes the registration accuracy.[5] The accuracy is expressed as percentages of an approximation of the diameter of the data sets. Overall, the result show that the

[5] For ICP, one iteration actually includes two iterations: first set A is registered with respect to set B, and then vice versa.

Figure 3-16 (a) A scanned object. (b) Initial registration of the views. (c) Errors in pairwise registrations accumulate.

new projective registration method converges much faster and gives comparable or more accurate registration. Especially noteworthy is the second experiment in Fig. 3-15. In this case, only a subset of the range data, most of the fuselage but no wings, was used, while the color images of the whole plane were used in 2D registration. The fuselage is roughly cigar-shaped, but not so symmetrical that geometry-based registration would be impossible. However, ICP failed to find the correct answer, while the color variations helped the projective method to reach the correct solution.

3.4 Multi-view registration

Extending pairwise registration to simultaneously registering several views is not straightforward. We first describe a simple approach for generalizing pairwise registration, followed by some proposed improvements. We then describe our approach to multi-view registration.

3.4.1 Generalize pairwise registration

A simple method for registering several views is to register views sequentially: the unregistered views are registered, one at a time, to a view that has already been registered. The problem with this approach is illustrated in Fig. 3-16. When the views are registered pairwise in the scanning order, small registration errors begin to accumulate. The accumulated registration error is manifested as a large gap between two views that should overlap in Fig. 3-16(c).

Turk and Levoy's [1994] solution to the compounding registration error relies on their scanner's ability to take cylindrical scans. A single cylindrical scan of the object is taken and is used as an anchor; all the other scans are then registered with the cylindrical scan. This solution limits the possible viewing directions of the linear scans by requiring there to be a significant overlap with the anchor scan. Further, it can only be used if the scanner can obtain a cylindrical scan. However, there is a way to develop this idea, so that these two limitations are overcome. One can register the views one at a time, merging the registered view into a single "metaview" [Chen & Medioni 92]. All the new views are then registered with the metaview rather than just a single neighboring or anchor view. This approach was also taken by Masuda *et al.* [1996] , as well as by Levoy's research group [Curless 97].

Gagnon *et al.* [1994] point out that when more views are added, it is possible that they bring information that could improve the registration of the previously registered views. Gagnon *et al.* propose to solve this problem as follows. A set of points in view i is projected to all the views $j \neq i$, and a transformation that registers view i using all the paired points is calculated. This process is iterated repeatedly over all the views until the registration slowly converges. Jin *et al.* [1995] tried to solve the same problem by incrementally building a surface model, against which new views can be registered and already registered views can be reregistered. The method proved quite slow as a new surface estimate was created after every registration step.

Eggert *et al.* [1996a] attempt to overcome the thresholding difficulties of pairwise correspondences (maximum allowed difference in position, normals, color, etc.) that most previous methods, including that of Gagnon *et al.*, have to deal with. The distance metric is modified to take into account normals and view directions, as described in Section 3.2.2, but no threshold is used. The original point is projected to the tangent plane of the point in the other views that is closest to it. They justify always pairing a point with exactly one other point with their basic assumption that all of the object has been observed at least twice, meaning a proper corresponding location does exist, it just needs to be found. On the other hand, most other views cannot see the corresponding surface location. However, we have observed that finding a mate for a point in only one other view can prevent the algorithm from converging to a correct solution. Sometimes the views form cliques such that all the views belonging to a clique are in a good registration, but the cliques themselves are not well registered to each other. As a result, the closest point is usually found in another view of the same clique, and no further progress is made, even though the global registration may be far from perfect. We adopted Gagnon *et al.*'s approach of pairing each point with the closest points in several other views in our earlier method in order to avoid clique formation. If a point is paired with many points, some of them in other cliques, the process is more likely to converge into a good registration.

3.4.2 New method for multi-view registration

All the previous approaches to multi-view registration described above use the basic ICP idea of iterating the process of pairing points and moving the views closer. The difficulty with such an approach is pairing points with corresponding points in the other views. First, it is a lot of work. Assuming we use the simple heuristic of pairing a point with the closest point in each other view, we have m views, each with n points, and we find pairs for only k points in each view: $O(m^2 k \log n)$ work is required each round just to find the pairs. Second, it is hard to get the pairs right. Even with compatibility heuristics, most pairs do not correspond to the same points on the object surface, and a large number of iterations is required before the registration converges. Additionally, due to unreliable pairs, the registration often converges to a local, instead of the global, minimum.

The main idea of our new multi-view registration method is to separate the projection and minimization steps. First, we attempt to find a set of *reliable* point pairs between each view and several neighboring views. The set of pairs is then kept unchanged, and a global registration is found that simultaneously aligns all the pairs.

We begin by finding an initial registration via interactive sequential pairwise registration as described in Section 3.2.1. The user is given visual feedback of the initial registration by showing the subsampled registered range data, surrounded by thumbnails of the color im-

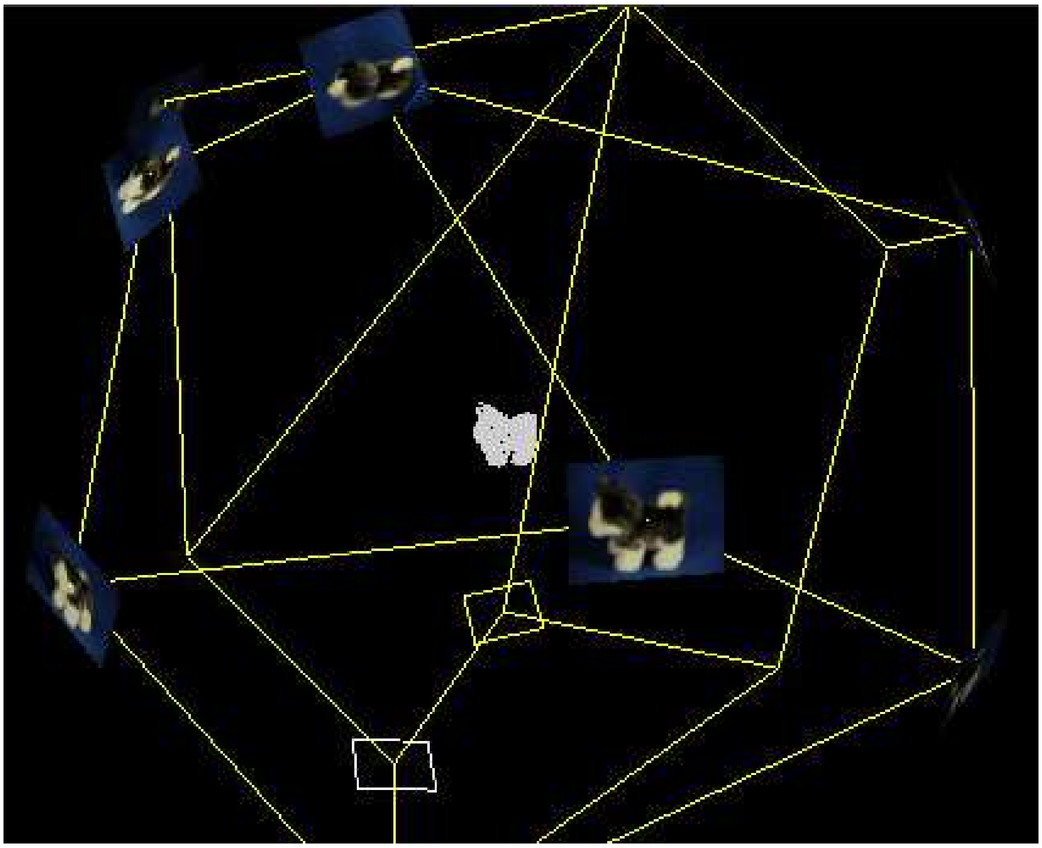


Figure 3-17 Multiview registration. Initially, views are registered pairwise and a collection of reliable point pairs is stored. Using fixed point pairs a global registration is found.

ages at the viewpoint of each sensor, as shown in Fig. 3-17. Each view is then registered pairwise with a few neighboring views. It is of no particular importance which pairwise registration method is used, as long as the results are accurate; we use the method described in Section 3.3. As shown in Fig. 3-17, a yellow line segment is added between each registered pair of views, and a collection of point pairs between those views is stored. Once the views are in good registration, reliable point pairs are found simply by using the same pairing method as was used in pairwise registration, and accepting only pairs that are very close to each other. The pairing is done symmetrically from each view to the other.

With the pairwise registrations forming a graph such that every view is connected to every other view through several paths, the global registration is found using the stored point pairs. There are several possible ways to solve this problem. Shum *et al.* [1995] proposed to solve it using principal component analysis with missing data. Stoddart and Hilton [1996] and Eggert *et al.* [1996b] both developed a method that attaches imaginary springs to the paired points and simulates a simplified version of the resulting dynamic system. We experimented with two different approaches to minimize the sum of squared distances of paired points.

We first tried the conjugate gradients method. The solution of registering n views consists of a $6n$ state vector that encodes three rotation and three translation parameters for each view.[6] We implemented routines that evaluate both the registration function and its gradient at a given point of the state vector, and solved the global registration using the implementation given in [Press *et al.* 92]. The method worked well, giving accurate results in a matter of minutes. Our previous approach, similar to the one in [Gagnon *et al.* 94], took hours, because of the costly projection step has to be repeated at every iteration, and typically dozens if not hundreds of iterations are required, depending on the object and the accuracy of the initial registration.

We then tried another approach that is simple to implement, is much faster, and produces results of equal accuracy. The idea is to use successive relaxation using Horn's [1987] algorithm for pairwise registration of paired points. The algorithm is initialized as follows. The initial set of views consists of one of the views. The other views have valid point pairs only if any of the views it was registered with is already in that set. Each of the remaining views is added to the set one at a time, but only if it has valid pairs. Its pose (orientation and location) is determined by applying Horn's algorithm to the valid pairs. Once all the views have been added, the relaxation process begins. The pose of a view is simultaneously registered with all the views with which it has point pairs. This process is repeatedly iterated over all the views until the views stop moving.

Where the conjugate gradients method takes minutes to complete, results of similar or

[6] Strictly speaking there are only $6(n-1)$ *independent* variables because the views can be registered with respect to any one particular view, but there is little difference if we solve for $6n$ variables.

better accuracy can be obtained by the relaxation method typically in a few seconds. The proof for convergence is simple: at each step, the total sum of squared distances is reduced, and as that sum is bound below (the sum is always nonnegative), the algorithm must converge.

All the data sets used in the rest of this dissertation were registered using this method.

3.5 Discussion

3.5.1 Automatic initial registration

Most registration algorithms require a rough initial registration as a starting point. In our opinion the best practical solution is to either obtain the initial estimate directly from the scanning system, or if the system is not automated, from the operator.

There have been suggestions for other means of obtaining an initial registration. One could use techniques from feature-based vision. Faugeras and Hebert [1986] located geometric features such as corners, creases and edges, and planes from the data, and tried to match them with similar features extracted from another data set. Typically three matched features are needed to constrain a rigid 3D transformation. The matches are mutually consistent with respect to the rigidity constraint if features in one set can be aligned with their matched counterparts in the other set using a rigid 3D transformation. The more mutually consistent matches can be found, the greater is the confidence of the correctness of the matches, and the greater the accuracy of the initial registration. However, not all objects even have detectable corners or edges. Besl [1990] described numerous features for registering free-form surfaces, such as elliptic, hyperbolic, and parabolic surface patches, contours of constant curvature, umbilic points, and curvature discontinuities. These, however, are often difficult to locate reliably, especially when the data contains some noise. Other work along these lines include [Stein & Medioni 90, Chua & Jarvis 96].

Johnson and Hebert [1997] proposed an interesting approach for automatically creating geometric surface feature templates directly from the data. A template called the spin image is calculated for a surface point as follows. A local 2D coordinate system is created centered at the point, and the two coordinates of another 3D point in this frame are its distance from the surface normal vector and the tangent plane at the original point. The 2D spin coordinate system is discretized, and all nearby points increment the value of the bin corresponding to their spin coordinates. Figure 3-18 shows three spin images for different parts of the same object. Different spin images can then be compared using image correlation, and a search similar to the one described in [Faugeras & Hebert 86] is used to find a set of mutually consistent matches.

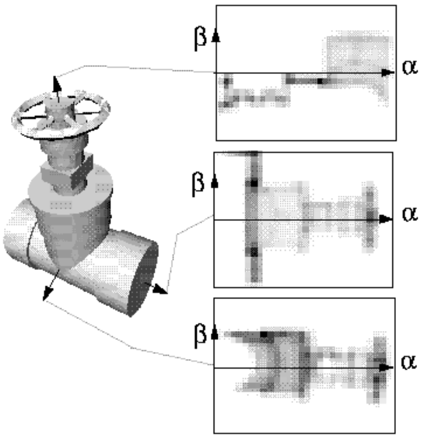


Figure 3-18: A spin image.

3.5.2 Other approaches to registration

Bergevin *et al.* [1995] implemented a heuristic search in the 6-parameter space of 3D registration (3D rotation + 3D translation). An initial rough transformation was placed at the root of the search tree. A node is expanded by perturbing each of the 6 parameters to either direction, the amount of perturbation depending on the depth of the node in the tree. The expanded nodes are evaluated, and *n* best of them are kept in the search list. The authors noted problems in the parametrization (the parameters are not independent) and in the control of the search, and actually ended up using the Chen and Medioni's method with a slight modification.

Higuchi *et al.* [1994] use a representation that decouples rotation and translation and note that solving the translation is easy once the relative rotation has been solved. First, a mesh that is the dual of a regular triangulation of a sphere is fitted to the data. While fitting, the mesh is kept locally regular by forcing the three edges joining in a vertex to have approximately the same length. At each vertex, a discrete estimation of the Gaussian curvature (called the simplex angle) of the underlying surface is calculated. When the curvature distribution is mapped on a sphere it results in a translation and scale invariant description of the surface shape called the Simplex Angle Image (SAI). Before registering partial views of the same surface, the number of nodes corresponding to the visible surface needs to be adjusted since the relative sizes of the visible and hidden areas vary depending on the viewing direction. The rotation can be solved by searching for the rotation of a sphere that minimizes the sum of squared differences of the spherical angles, using either a coarse-to-fine search strategy [Higuchi *et al.* 94], or a full search at the resolution of the SAI mesh [Hebert *et al.* 95]. The greatest drawback of the method is that it can be applied only to objects topologically equivalent to a sphere.

Champleboux *et al.* [1992] developed a registration method that directly minimizes the distance between two surfaces. Their method is iterative, but unlike ICP-type algorithms, it does not pair points from one surface with points in the other. The idea is to build a function that can be quickly evaluated at any 3D location and which returns the distance from that point to the surface. The function representation allows speedy numerical estimation of its gradients, and a surface is moved closer to the other using a gradient descent type of optimization. The distance function was encoded in an octree-spline. The space is divided into an octree where the nodes get progressively smaller the closer they are to the surface. The distances from the corners of the octree nodes to the surface are calculated and stored in advance, and the distance from any point inside a node is interpolated from the values stored in corners. The Levenberg-Marquardt algorithm was used for finding the registration transformation. Szeliski and Lavallée [1994] later extended the method to register non-rigid objects. The idea is to find a combination of a rigid transformation and a space deformation that aligns the two scans. The method first performs a rigid registration, following a sequence of increasingly general space warpings.

Blais and Levine [1993, 1995] seem to be the first ones to evaluate the quality of registration by projecting one range map to the image plane of the other. This technique was later used by Masuda and Yokoya [1995], Weik [1997] and us. Like Masuda and Yokoya, but unlike Weik's and our work, the projection is used only for evaluation, not for registration. The optimization is done using very fast simulated reannealing (VSFR), a stochastic global optimization method. The advantage of such an approach is that the registration is less likely to get stuck in a local minimum, the disadvantage is that the method requires a large number of evaluations of the registration function. Each function evaluation takes $O(n)$ time, which is significantly less than the $O(n \ln n)$ time required in traditional ICP. However, if the method requires $O(\ln n)$ iterations, the net effort is greater than that of ICP, and certainly greater than that required by combining the fast evaluation with iterative pairing-minimization methods.

Laurendeau *et al.* [1996] performed a comparative study of five general purpose optimization algorithms for range image registration. The algorithms that were considered were a simple evolutionary algorithm, a genetic algorithm, a dynamic hill climbing algorithm, and two flavors of simulated annealing. The conclusion was that although the dynamic hill climbing and simulated annealing methods were better than the other two methods, none of them performed as well as most other published registration methods. Regarding these search-based methods [Blais & Levine 95, Bergevin *et al.* 95, Laurendeau *et al.* 96] we share Chen's [1994] opinion: *we feel that searching through the huge parameter space, even with some heuristics, is neither computationally tractable nor necessary.*

3.5.3 Contributions

The contributions of this chapter are:

- A new robust method for pairwise registration of views that exploits color information. The method finds a consistent set of point pairs between two range maps by projecting the colored range data to an image plane, aligning the images using a planar projective transformation, and pairing points projecting to the same pixel. The 3D registration transformation is found by a robust LTS version of a well known least-squares point set alignment method.

- A new robust and fast approach for multi-view registration that separates a difficult problem into two manageable parts. The first part registers the views pairwise using any 3D registration method and then finds reliable point pairs. The point pairs define constraints for a global registration that is found in a separate optimization step.

Chapter 4

Surface reconstruction

4.1 Introduction

There are two important steps in surface reconstruction from range data. The data must be *integrated* into a single representation, and surfaces must be *inferred* either from that representation or directly from the original data. There does not seem to be a clear concensus in which order these two steps should be taken.

Several researchers have decided to first approximate each data set as a polygonal mesh [Rutishauser *et al.* 94, Turk & Levoy 94, Pito 96]; the individual meshes are then connected to form a single mesh covering the whole object. Soucy and Laurendeau [1995] first divide the range data into subsets based on which surface regions are visible within each view. In each subset, the redundancy of the views is used to improve the surface approximation. Finally, the triangulated non-overlapping subsets are connected into a single mesh.

Other authors [Hoppe *et al.* 92, Curless & Levoy 96, Hilton *et al.* 96] chose first to integrate the data into a signed distance function and then extract a polygonal mesh using the marching cubes algorithm [Lorensen & Cline 87]. Hoppe *et al.*'s method was designed to work with arbitrary point clouds. They first estimate local tangent planes to each data point, and then propagate the tangent plane orientations over the whole data set using the minimum spanning tree of the points[1]. The distance of a 3D point is evaluated as the distance to the closest tangent plane, where the distance is positive if the point is above the plane, negative otherwise. Ideally the zero set of the distance function would follow the object surface.

[1] The distance metric used for calculating the minimum spanning tree combines Euclidean distance and an estimate of the surface curvature, so the tree tends to connect flat regions rather than connect across creases.

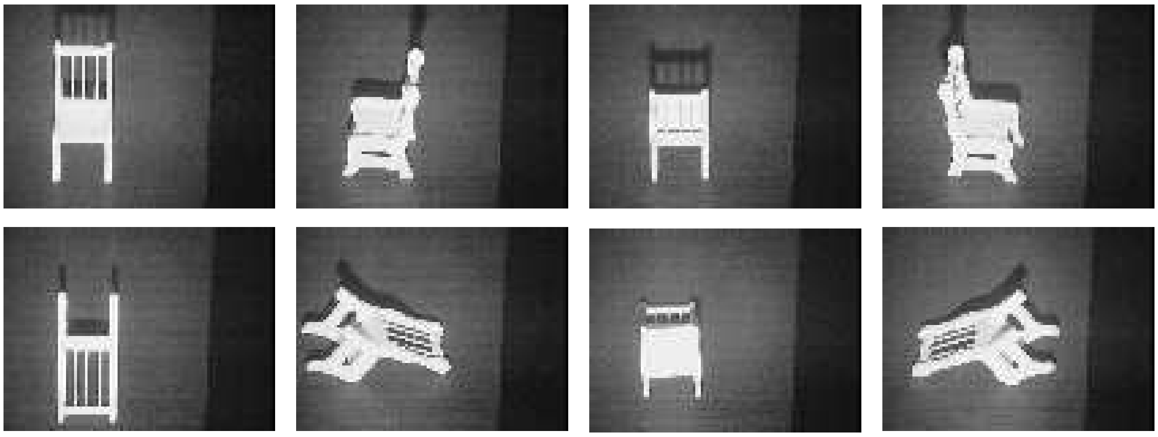
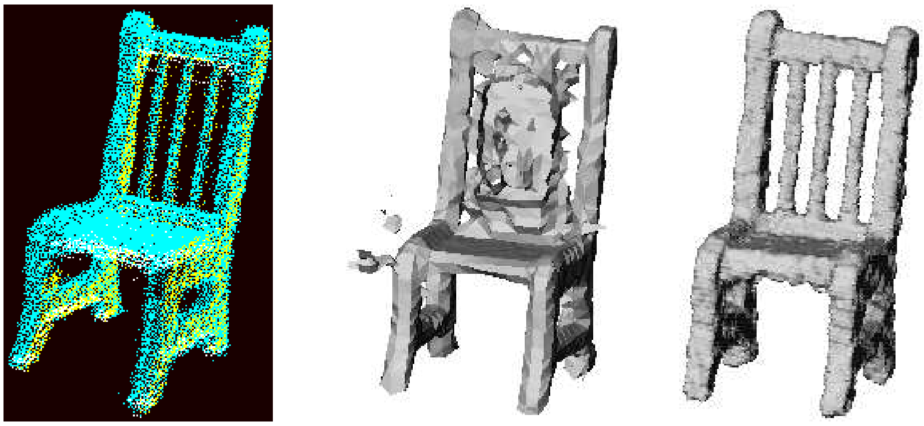


Figure 4-1 Eight intensity images corresponding to the views of the miniature chair.

However, the local tangent plane estimation phase is likely to smooth the zero set. Hoppe *et al.* later improved the results by fitting the result of the marching cubes approximation better to the original data [Hoppe *et al.* 93].

Curless and Levoy improved the distance function estimation for the case that the input is structured in the form of a set of range maps [Curless & Levoy 96]. They define a volumetric function based on a weighted average of signed distances to each range image. Their scheme evaluates this volumetric function at discrete points on a uniform 3D grid, and uses these discrete samples to determine a surface that approximates the zero set of the volumetric function. Like Hoppe *et al.*, their scheme may fail to detect features smaller than the grid spacing and has difficulties with thin features. Also, it requires a significant amount of storage space, although this can be alleviated through run-length encoding techniques. However, their use of space carving (throwing away regions known to lie outside the object) makes the approach much more robust. Since their approach is less susceptible to smoothing, they evaluate the signed distance function at very fine resolution, and use the output of the marching cubes algorithm as their final result.

We originally used the method of [Hoppe *et al.* 92] to obtain an initial mesh, which we then simplified and improved the fit to the original data using the methods of [Hoppe *et al.* 93]. The initial mesh algorithm does not require any knowledge (such as viewing directions)



(a) (b) (c)

Figure 4-2 An example where the old method for obtaining initial surface with correct topology fails. (a) The registered point set. (b) The result from using the method by Hoppe *et al.* [1992]. (c) The result from our method.

aside from the data points themselves. It works quite nicely if the data does not contain outliers and uniformly samples the underlying surface. Unfortunately real data often violate both of those requirements. Figure 4-1 shows eight views of a miniature chair (courtesy of Prof. Patrick Flynn of WSU), and Fig. 4-2(a) shows the range maps after registration. Although most of the outliers and background data points were interactively removed from the data sets prior to initial mesh estimation, many outliers remain, especially around the spokes of the back support, some data was missing, and the data wasn't uniform enough for the algorithm to produce a topologically correct result. Figure 4-2(b) shows the incorrect result.

We decided to abandon Hoppe *et al.*'s initial mesh method and create a more robust method that produced the result shown in Fig. 4-2(c). Like Curless and Levoy, we use the idea of space carving. However, we use a hierarchical approach that saves storage space and execution time, and we do not attempt to directly obtain the final result. Instead, we concentrate on capturing the object topology as correctly as possible given the input data. Additionally, we use only robust methods such as interval analysis [Snyder 92] that enable us to recover thin objects that are typically difficult to model using the signed distance function approach. We kept the mesh optimization method of [Hoppe *et al.* 93] since for our purposes we typically want a mesh that is both accurate and concise. Typically one-pass methods such as [Curless & Levoy 96] can only produce meshes that are either dense or not very accurate. Hoppe *et al.*'s mesh optimization first improves the accurace, and can also simplify the mesh drastically with little sacrifice in accuracy.

Section 4.2 describes our hierarchical space carving method for creating initial meshes, while Section 4.3 briefly describes the mesh optimization algorithm of Hoppe *et al.*

4.2 Hierarchical space carving

This section describes our hierarchical space carving method for generating approximate meshes and determining the object topology. First we introduce space carving and show how space can be carved a cube at a time. Then we describe an improvement that concentrates computation close to the actual surface by using a hierarchical octree-based scheme. We discuss how the mesh is extracted from the octree description, and then conclude the section with a discussion of results and various properties of the method.

Our method has several assumptions. The data is assumed to be a collection of registered range maps acquired from various viewpoints around the object. The data has to be obtained using some line-of-sight method, so that every straight line between a data point and the sensor lies entirely outside the object. Finally, the calibration parameters of the sensor must be known in the sense that any 3D point can be projected to the sensor's image plane, or equivalently, the correspondences between 3D lines and pixels on the image plane are known. If the calibration parameters are not known before hand, it is often possible to estimate them directly from the known correspondences between 3D points and their 2D image coordinates. For example, in the case of the dataset in Fig. 4-1 we did not have the calibration parameters but estimated them using Tsai's method of camera calibration [Tsai 87].

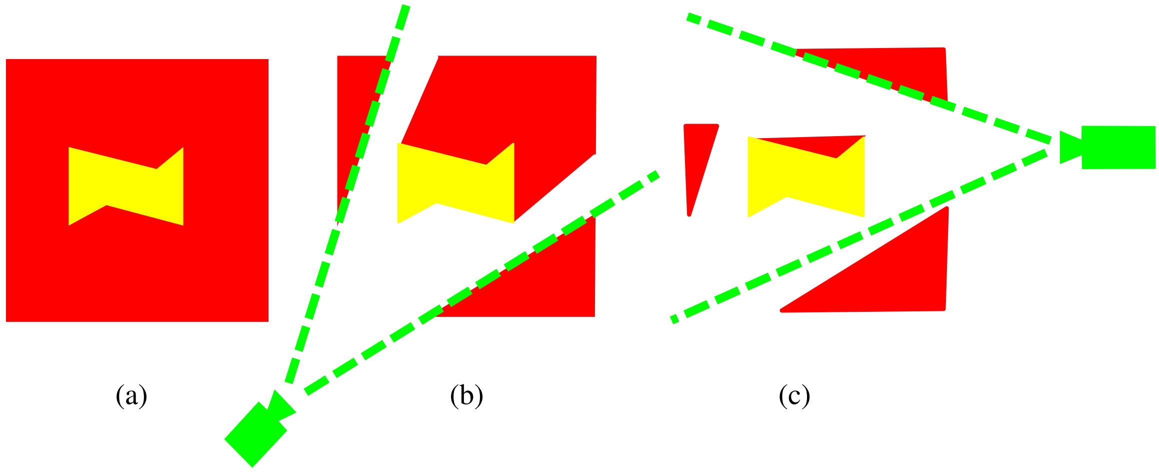
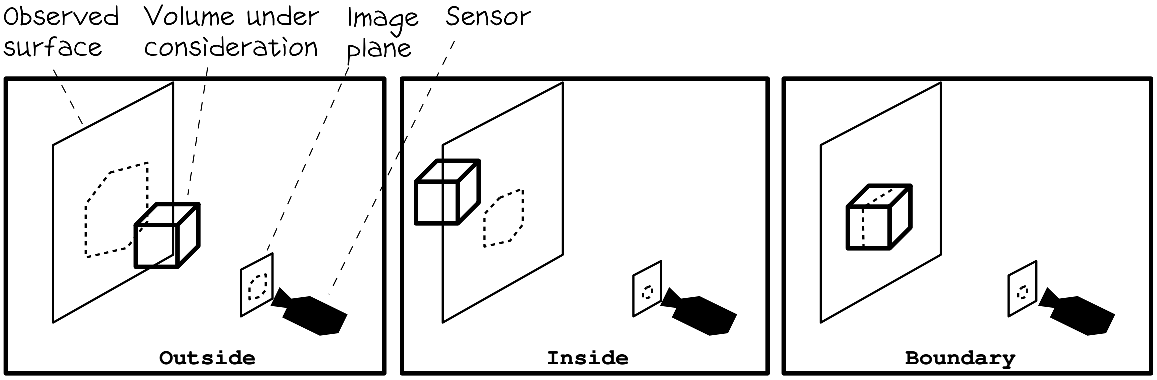


Figure 4-3 The idea of space carving. (a) The working volume contains an object. (b) A scan can prove some of the volume lies outside the object. (c) More scans remove more space, but the object remains.

4.2.1 Space carving

The idea of space carving is illustrated in Fig. 4-3. In Fig. 4-3(a) there is a working volume, and the object we are trying to reconstruct is only known to be somewhere inside of it. Figure 4-3(b) shows a scanner and its viewing cone. The left and bottom sides of the object are visible to the scanner, and the volume between the scanned surface and the sensor can be removed. If there is data from the background in addition to the object, even more of the unknown volume can be removed. Figure 4-3(c) shows how another view carves more space away.

In the end we are left with a connected volume that closely approximates the object. An approximation of the object surface can be obtained by extracting the boundary between the space that was carved away and the volume that remains. The resulting surface estimate can have arbitrary topology, i.e., the object can have any number of holes, yet the surface itself is without boundaries or holes, even if not all of the object surface was scanned. In general, the resulting surface is the one with maximum volume such that the model is still compatible with all of the scanned data. Figure 4-3(c) shows a situation where the whole top side of the object remained unseen by the scanner. Nevertheless, the gap in the input is



(a) (b) (c)

Figure 4-4 The three cases of the algorithm. (a) The cube is in front of the range data. (b) The cube is entirely behind the surface with respect to the sensor. (c) The cube intersects the range data.

filled by a plausible solution that is not far from the true surface.

4.2.2 Carve a cube at a time

Our algorithm discretizes space into a collection of cubes or voxels and processes them one at a time. Figure 4-4 illustrates how the status of a single cube with respect to a single view is determined:

- In case (a) the cube lies between the range data and the sensor. Therefore the cube must lie outside of the object and can be discarded.

- In case (b) the whole cube is behind the range data. As far as the sensor is concerned, the cube is assumed to lie inside of the object.

- In case (c) the cube is neither fully in front of the data or behind the data, in most cases because it intersects the range map. The cube is assumed to intersect the object surface.

The cubes are labeled as follows. The eight corners of the cube are projected to the sensor's image plane, where their convex hull forms a hexagon. The rays from the sensor to the hexagon form a cone, which is truncated so that it just encloses the cube (see Fig. 4-5). If all the data points projecting onto the hexagon are behind the truncated cone (i.e., are farther than the farthest corner of the cube from the sensor), the cube is outside the object. If all those points are closer than the closest cube corner, the cube is inside the object. Otherwise, we have the boundary case. Areas for which range data is missing are treated as points that are close to the sensor. If the data has been preprocessed so that parts of the range maps were labeled as background, the background points are treated as being infinitely far away from the sensor.

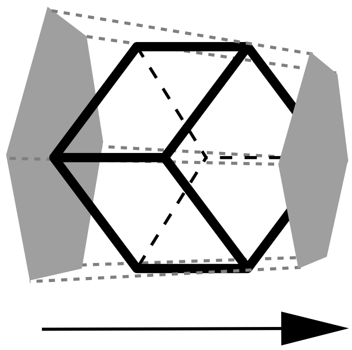


Figure 4-5: An octree cube and its truncated cone. The arrow points to the sensor.

Figure 4-6 illustrates the situation in which the whole working volume has been tesselated into cubes and the scene has been scanned from several viewpoints. The cubes are processed independently from each other. Each cube is labeled with respect to all the views using the following rules. Determining that a cube is outside of the object is conclusive evidence; therefore it is enough for a single view to label a cube as outside to remove it. We can only indirectly infer that a cube is inside of the object, and then only if every view agrees. If one more view were available, that view might indicate that the cube is really outside the object or on its boundary, but without that view we can only assume the cube is inside. Again, if a cube is neither in nor out, it is labeled as part of the object boundary. The implicit surface corresponding only to the observed data will then be the surface of minimum volume. That is, it biases toward range points that are behind others. This is corrected during mesh optimization.

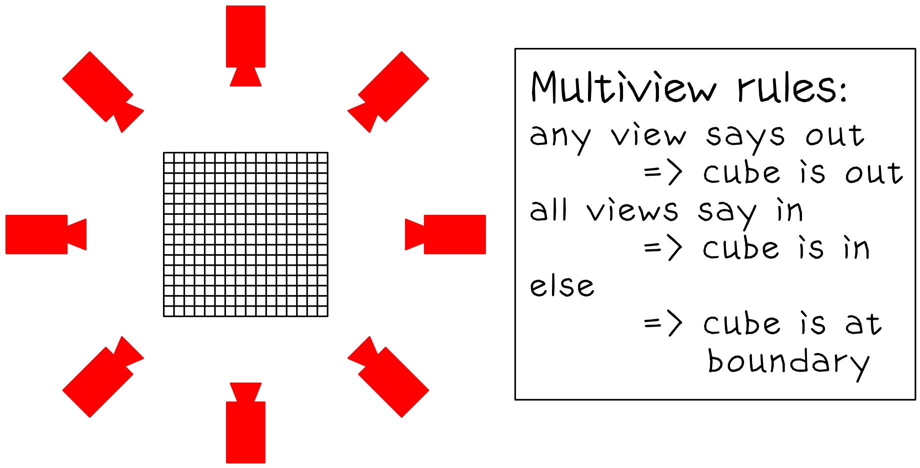


Figure 4-6 Multiple cubes and sensors.

4.2.3 Hierarchical approach

A fixed partitioning of space suffers from a tradeoff between accuracy and efficiency. With large cubes the object cannot be modeled accurately. Using small cubes gives better results, but requires much more work. However, if we take a hierarchical approach using octrees, large chunks of space can be quickly ruled out and our efforts can be concentrated close to the surface.

Figure 4-7 illustrates the hierarchical space carving approach for the chair data set. Initially a large cube surrounds the data. Since by definition it intersects the data, it is immediately subdivided into eight smaller cubes, which the algorithm tries to label as being outside of the object. In Fig. 4-7 none of these eight smaller cubes were carved away. At the next level a number of cubes are discarded. The remaining volume shrinks closer and closer to the actual surface, until finally, in this case after seven subdivisions, we are left with a relatively accurate approximation to the chair, with the correct topological structure.

In this *simultaneous processing* order, the whole octree is traversed once, and the consensus of all the views is used to determine the labeling for each cube. Another possibility is *sequential processing*, where the algorithm hierarchically processes one view at a time, building on the results of the previously processed views. Sequential processing can be used

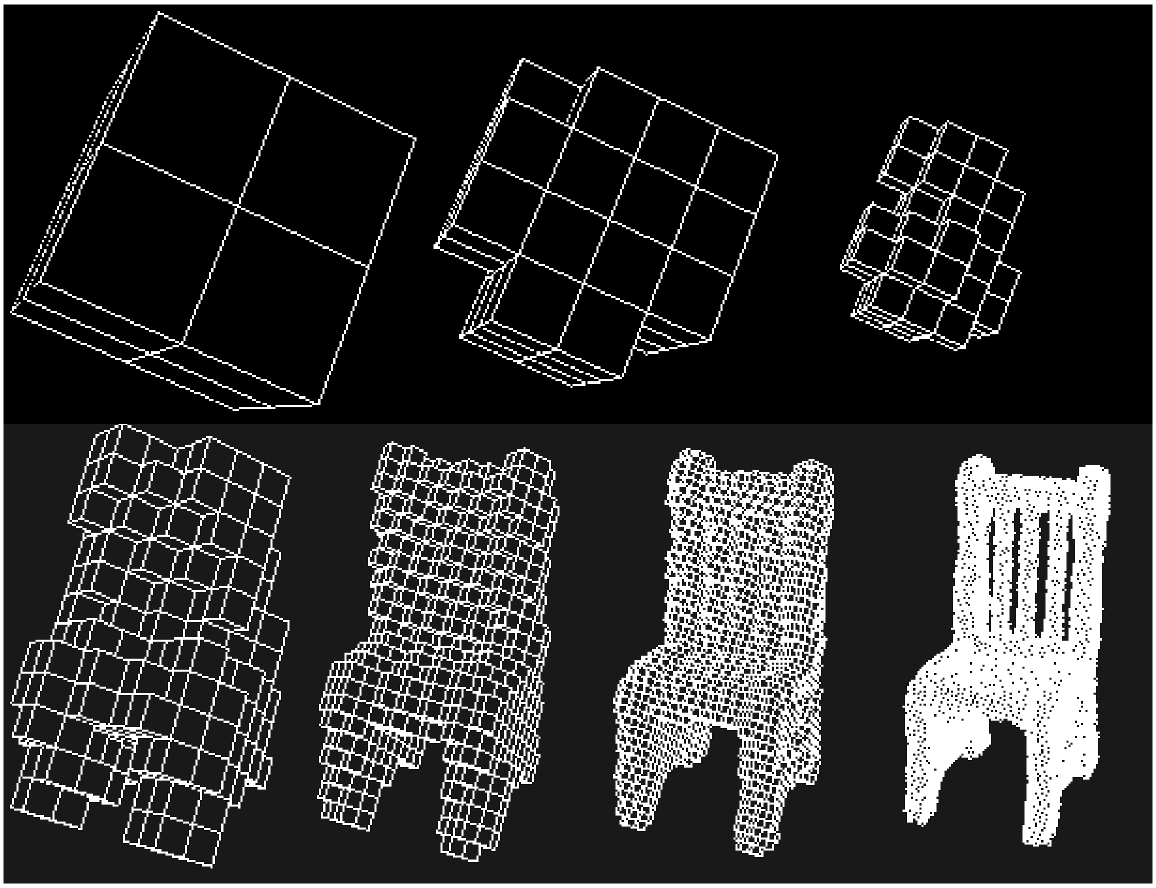


Figure 4-7 Hierarchical carving of cubes.

to integrate a new view into a previously processed octree. The algorithm recursively descends the octree and performs the occlusion test for each cube that has not been determined to lie outside of the object. If the new view indicates that a cube is outside, the cube is relabeled and the subtrees below it are removed. Similarly, a boundary label overrides a previous inside label, in which case the cube's descendants have to be recursively tested, potentially to the maximum subdivision level.

Although both processing orders produce the same result, the simultaneous processing order is in general faster [Szeliski 93]. In sequential processing the silhouette of the object often creates a visual cone (centered at the sensor) that separates volumes known to be outside from those speculated to be inside. The algorithm would then have to recurse to the finest subdivision level to accurately determine this boundary. In simultaneous processing, however, another view could determine at a rather coarse level of subdivision that at least part of that boundary is actually outside of the object, and the finer levels of the octree for that subvolume need never be processed.

The sequential processing approach has a different kind of advantage. Since we only need to be store the data from a single view at a time, this approach scales better when there are a large number of views of the same object. If we have dozens or even more views, we can use a hybrid between the sequential and simultaneous approaches. First, choose a small subset of the views (8-12) from different sides of the object. The subset is processed using the simultaneous approach, and most of the excess space is carved away. The first set of views is discarded, a new subset is selected, and it is processed using the current model, until all the views have been processed.

In both the sequential and simultaneous processing approaches, we may occasionally determine that all the children of a cube lie in free space. Whenever this happens the eight sibling cubes are removed and their parent is labeled as lying outside the object.

4.2.4 Mesh extraction

The labeling of the octree nodes divides the space into two sets: the cubes known to lie outside of the object and the cubes that are assumed to be part of the object. Our surface estimate will be the closed boundary between these sets. This definition allows us to create a plausible surface even at locations where no data was obtained. The boundary is represented as a collection of vertices and triangles that can easily be combined to form a mesh.

The octree generated by the algorithm has the following structure: outside cubes and inside cubes do not have any children, while the boundary cubes have a sequence of descendants down to the finest subdivision level. To generate a surface representation, we traverse the octree starting from the root. At an outside cube we don't need to do anything. At a boundary cube that is not at the finest level, we recursively descend to the children. If we reach the maximum subdivision level and the cube is either at the boundary or inside, we check the labelings of the cube's six neighbors. If a neighboring cube is an outside cube, two triangles are created for the face they share. In an inside cube that is not at the maximum subdivision level, the algorithm checks whether it abuts an outside cube, and if so creates enough triangles (of the same size as the ones created at the finest level) to cover the shared part of the face.

Finally, we combine the triangles into a closed triangle mesh. For the most part, connecting the triangles into a mesh is straightforward; triangles that share an edge become neighbors. However, if there are two boundary cubes sharing only an edge, there is a choice in the mesh connectivity. Since the octree cubes contain all of the object and possibly some more, the volumes of two cubes can only connect across faces, and not across edges or corners. Therefore such cubes should be separated. Similar processing is needed when two cubes share only a single vertex. Note that for most mesh representations this requires replicating the shared vertices.

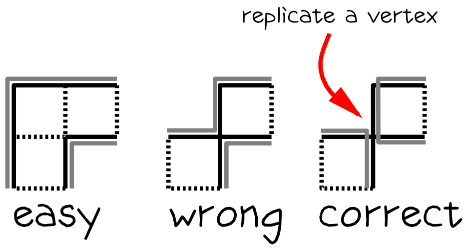


Figure 4-8: Only cubes that share a face should be connected in the mesh.

4.2.5 Discussion

Our work is not the first to use octrees for hierarchical processing of the working volume. Szeliski [1993] constructed octree bounding volumes of objects from silhouettes extracted from intensity images. Each silhouette forms a cone centered at the corresponding camera, and the resulting model is the intersection of these visual cones. In the limit (with infinite number of input images) the resulting model is called the line hull. The objects do not have to be convex for this technique to work; some holes can be recovered. However, only surface locations that are part of the silhouette from some viewpoint can be modeled accurately, so indentations and locations where both the principle curvatures are negative cannot be recovered. Connolly [1984] created octree models from range maps taken from arbitrary viewpoints. Each range map is converted to a quadtree, and the quadtrees are transformed into the octree coordinate system, and then assimilated into an octree model of the object. The algorithm projects four rays for each quadtree node through the octree, clearing octree nodes along the way. Since the level of octree nodes is the same as the level of the quadtree node, the hole that is carved is jaggy and larger than it should be. The carving could be made more accurate by performing the carving at a finer level of resolution, but then the processing becomes more costly, and it becomes more difficult to guarantee that all the cubes that should be removed are removed. Chien *et al.* [1988] tried to address these problems by creating range quadtrees from six orthogonal viewing directions and intersecting them into a volumetric octree model. Although the quadtrees conform much better to the structure of the octree, there is a large class of objects that cannot be modeled using this approach, including objects with at least two holes that are not perpendicular to each other. Our approach of projecting subvolumes onto range images is a much more robust method; it eliminates the deficiencies of Connolly's without introducing the restrictions of Chien *et al.*'s approach. Other work that use similar principle to ours include [Li & Crebbin 94, Tarbox & Gottschlich 94].

In the rest of this section we discuss some execution statistics, the connection between our approach and interval analysis, how our method deals with thin objects, and how the method implicitly removes many outliers in the input data.

levels	2	3	4	5	6	7	8
nodes	73	217	649	2377	9393	41201	190441
faces	74	128	422	1372	5152	18446	71038
seconds	0.1	1.2	0.8	1.7	6.0	26.1	121.1

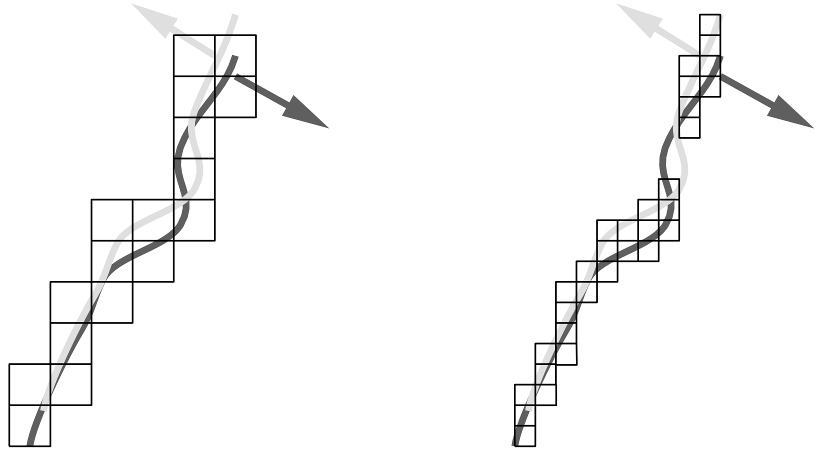
Table 4-1 Statistics for the chair data set. For example, at 5 levels of subdivision the octree contained 2377 nodes, the boundary between the object and free space was 1372 faces (twice as many triangles), and it took an additional 1.7 seconds to process from the level 4 octree.

Execution statistics

Table 4-1 summarizes some execution statistics for the chair data set. It shows the number of octree nodes, the number of square faces between the outside cubes and the others, and execution times for subdivision levels 2 through 8. The timings were obtained using an SGI O2 with a 175 MHz R10000 processor, and they show the incremental extra time it takes to process a level given that all the data was already processed up to the previous level. From these results we can make several observations. First, the execution times are fast enough to allow interactive selection of a suitable processing resolution, balancing the competing goals of correct recovery of object topology and conciseness of the representation. Second, the size of internal data structures, output, and execution times all grow exponentially as powers of (approximately) four. The theoretical reason for this is quite obvious. Each increase in level causes the octree cells to be bisected in three orthogonal directions, so the number of cells grows by a factor of $2^3 = 8$. However, the object surface is a 2D manifold embedded in 3D, so the number of cells that intersect the surface grows only by a factor of $2^2 = 4$. The fact that time and space requirements grow approximately by factors of four demonstrates that the algorithm automatically concentrates its efforts close to the surface. There is a seeming anomaly in the timing for level 3. The probable reason is that at the coarse levels the cubes' projections onto the sensors' image planes cover a large area, so the inside/outside labeling of the cubes takes a long time.

Interval analysis

Our idea in surface reconstruction was to avoid nonrobust techniques such as averaging as much as possible. Instead, we propagate and manipulate constraints into more useful forms that allow robust estimation of the object surface topology, while we leave the accurate fitting of a concise surface representation to the data to a later stage. The space carving method that we use to robustly determine surface topology is a particular instance of a general technique called interval analysis.



(a) (b)

Figure 4-9 A thin sheet seen from the left (light gray) and right (dark gray) is reconstructed correctly in (a). In (b) registration error and small cube size combine to cause a hole.

Interval analysis [Snyder 92] takes a conservative approach for determining function values. Let's assume we want to plot a function $f(x)$, and we don't know the actual values, but we can somehow deduce the minimum and maximum values f attains within a given interval. Rather than plotting some approximation of f, we can plot a block covering all the possible values within the bounding box. Or, if we can obtain better bounds for f within a shorter interval, we can subdivide the original interval into shorter segments. Thus, interval analysis allows us to narrow down the interval within which the function value lies by eliminating the impossible intervals, ideally converging arbitrarily close to the actual value.

In our case we use interval analysis to determine binary function values that indicate whether a point in a volume is inside or outside an object. We partition space into regions (cubes) and conservatively determine for each region whether it lies completely inside, completely outside, or neither. Using recursive subdivision, we efficiently prune large regions of space away from the object boundary, and focus both computation and storage on the spatial region of interest, that is, the region next to the surface.

Thin objects

Some methods that employ signed surface distance functions are unable to correctly reconstruct thin objects. Curless and Levoy [1996], for example, build a distance function by storing positive distances to voxels in front of the surface and negative distances to voxels behind the surface. In addition to distances, weights are stored to facilitate combining data from different views. With this method, views from opposite sides of a thin object interfere and may cancel each other, preventing reliable extraction of the zero set of the signed distance function.

In the case of a thin sheet of paper, our algorithm would construct a thin layer of octree cubes (voxels) straddling the paper by carving away the space between the sensor and the object but leaving the voxels that intersect the object, creating a shell around the surface. This is illustrated in Fig. 4-9(a). Note, however, that our method can fail in the presence of measurement noise and registration error if the minimum cube size is set too small, as illustrated in Fig. 4-9(b). Due to a registration error the indentation on the light gray view penetrates in front of the darker view, and vice versa. This causes too many cubes to be carved away, creating a hole. This example shows that in order to correctly reconstruct thin objects, the minimum size for the cubes should be larger than the sum of the registration error, the range sampling error, and the sampling density.

Implicit removal of outliers.

One of the benefits of our approach is that it automatically removes most of the outliers in the input data. Suppose that our first view contains some outlier data points that fall outside the object. Because we have only a single view, our algorithm first labels the cubes containing the outliers as lying on the object surface. However, if from the perspective of another view one of these cubes lies in front the object (or even background), the cube is removed, thereby eliminating the corresponding outlying data. This may also happen if the outlying point is a valid data point from the background instead of an erroneous range measurement. Suppose that the views were obtained by reorienting the object in front of a stationary scanner and that the views have been registered into a common coordinate system using a subset of the surface points. Now the background moves relative to the object from view to view, and in general the background data in one view can be identified by another view and labeled as such or removed altogether.

Outliers that lie behind the surface as seen from the scanner can be damaging, since they could cause our algorithm to incorrectly carve away a piece of the object. For several reasons, this has not been a problem for us. Most outliers we have observed either belong to other objects in the background, or appear around the silhouette edges of the object. We use background data to carve the space around the object more efficiently, and outliers at the object boundaries do not cause deep holes to be carved into the object.

Contributions

The main contribution of this section is the development of a robust method for generating an initial mesh that recovers the object topology. In comparison with Curless and Levoy's [1996] space carving method, our method has the following desirable properties:

- It is easier to implement.
- It is faster to execute.
- Its hierarchical processing saves resources and makes it easy to interactively decide the desired level of resolution.
- It can better deal with thin surfaces.

4.3 Mesh optimization

The hierarchical space carving algorithm produces a mesh that is not a very good approximation for the object surface. The mesh is dense, and the orientations of the mesh faces

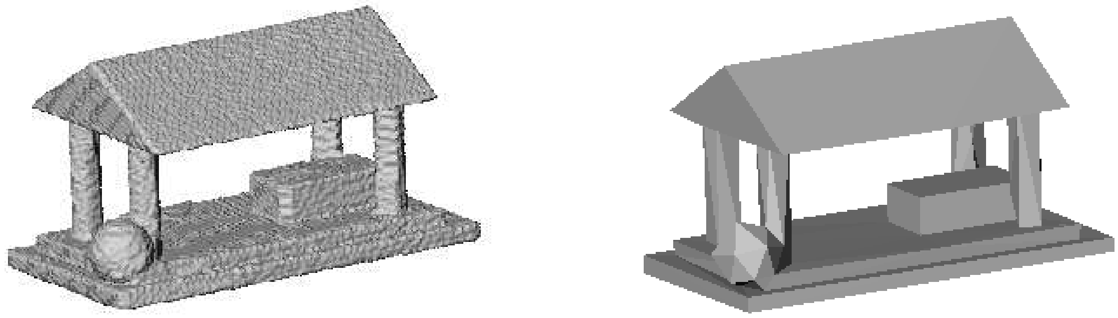


Figure 4-10 An initial mesh for a temple (smoothed for display, 203442 faces) and a simplified version obtained using the mesh optimization algorithm of [Hoppe et al. 93] (188 faces).

suffer from severe quantization as they are strictly aligned with the coordinate axes. However, the mesh is a proper 2D manifold and its vertices are relatively close to the object surface, so it serves as an excellent starting point for nonrobust optimization methods.

The mesh optimization method we adopted is described by Hoppe *et al.* [1993,1994]. The approach tries to fit the initial mesh to a set of unorganized points, while at the same time simplifying the structure of the mesh. Clearly these two goals are conflicting: the more faces there are in the mesh, the more accurately the data can be approximated. However, a concise surface description is faster to operate on, to transfer, and to display. A practical goal is therefore a compact but still fairly accurate mesh. The left side of Fig. 4-10 shows the initial mesh created from a synthetic data set describing a temple. The mesh is dense, containing over 200,000 triangles, and it has been smoothed for display using a simple surface fairing method introduced by Taubin [1995]. On the right is the simplified and fitted result containing less than a tenth of a per cent of the original faces. The planar sections of the temple have an excellent fit, while some accuracy was sacrificed for conciseness in the curved regions.

For completeness, we briefly describe the mesh optimization algorithm developed by Hoppe *et al.*, including the energy function and the iterative process by which it is minimized. Finally we show another mesh optimization example.

4.3.1 Energy function

A central concept in mesh optimization is the definition of the energy function that is to be minimized. Hoppe *et al.* [1993] proposed an energy function that is the sum of three terms: E_{dist}, E_{rep}, and E_{spring}.

The first term, E_{dist}, attempts to pull the mesh close to the data points. It is defined by the sum of the squared distances of the data points to the mesh. The second term, E_{rep}, strives to make the mesh simple. Its definition is simply the number of vertices in the mesh scaled by a representation cost c_{rep}. Adding vertices increases E_{rep} while deleting vertices decreases it. The last term, E_{spring}, is defined as the sum of squared lengths of the mesh edges, scaled by a spring coefficient κ. This term is not intended to smooth the mesh. Instead, it is meant to guide the optimization into a stable energy well, and its influence can be gradually decreased by decreasing κ as the optimization converges to a solution. A nonzero spring coefficient is especially useful at surface locations that are due to hole filling properties of space carving, as those areas have no corresponding range data.

4.3.2 Iterative minimization method

Directly minimizing the energy function is a difficult discontinuous nonlinear optimization problem. Hoppe *et al.* break the solution process into two simpler optimization steps that are applied iteratively to the current mesh. In the first step, the mesh structure/connectivity is held fixed, while the vertices are moved to bring the mesh close to the data points. The second step attempts to simplify the mesh structure while keeping the mesh close to the data. As the iteration proceeds, parameters such as the representation cost c_{rep} and the spring coefficient κ are modified. Typically, c_{rep} is increased as the mesh gets more concise, while κ can be reduced if the mesh seems to behave well and the input data is not very noisy.

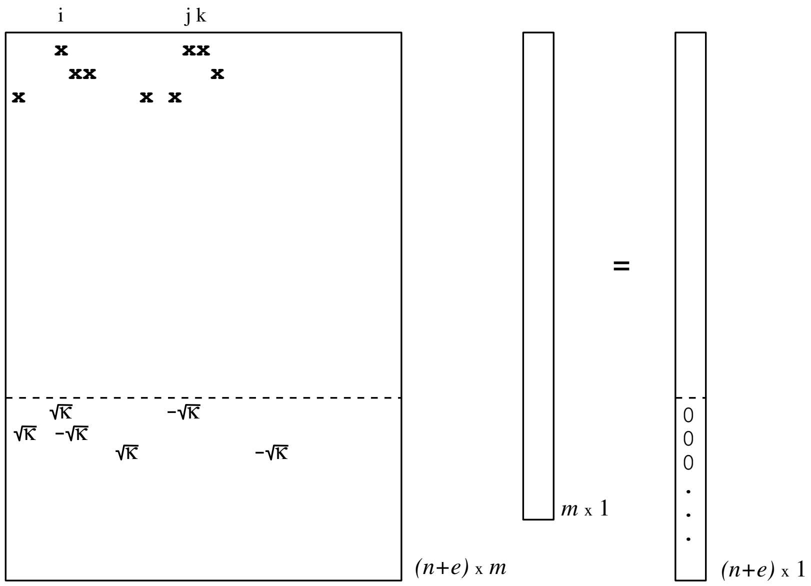


Figure 4-11 The structure of the linear system $A\mathbf{x} = \mathbf{b}$.

Optimize over vertex positions

The first subproblem is to optimize the vertex positions over a fixed mesh. First, the data points are projected to the closest point on the mesh, and the barycentric coordinates of the projection point are calculated with respect to the mesh triangle containing it. Using this information we can separately find the x, y, and z coordinates by minimizing a function of the form $||A\mathbf{x} - \mathbf{b}||_2^2$, once for each vertex coordinate. Minimizing that function is equivalent to solving in the least-squares sense the linear system $A\mathbf{x} = \mathbf{b}$, whose structure is illustrated in Fig. 4-11. Let us denote the number of data points by n, the number of vertices in the mesh by m, and the number of edges in the mesh by e. Then A is an $(n + e) \times m$ matrix and $\mathbf{b}$ is an $(n + e)$ vector, where the first n rows correspond to E_{dist} and the last e rows correspond to E_{spring}, while $\mathbf{x}$ contains the unknown vertex coordinates. In Fig. 4-11 the first data point projects onto the mesh triangle spanned by the vertices i, j, and k. All columns of the first row of A are zeros except the columns i, j, and k, which contain the barycentric coordinates of the projection of the first point. The first row in $\mathbf{b}$ contains one of the coordinates of the first point. All rows up to row n are filled in the same manner. Assuming the first edge in the

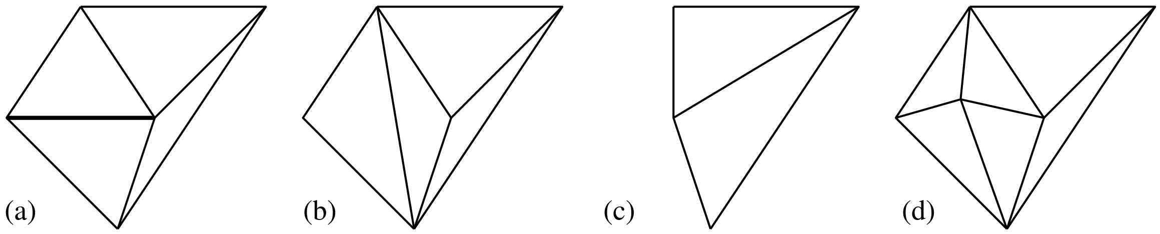


Figure 4-12 Elementary mesh transformations. (a) The initial configuration, with the current edge drawn in bold. (b) Edge swap. (c) Edge collapse. (d) Edge split.

mesh connects vertices i and j, row $n+1$ of A contains $\sqrt{\kappa}$ and $-\sqrt{\kappa}$ in the corresponding columns and zeros elsewhere, while the corresponding row in $\mathbf{b}$ contains zero. This system is best solved using the method of conjugate gradients (*cf.* [Golub & Van Loan 96]), which can take advantage of the system's sparse structure. Strictly speaking, this system does not minimize the distances of the data points to the mesh. Instead, the relative locations of the data points' projections onto the mesh are fixed and the vertex positions that minimize the data points' distances to their fixed projections are found. This tends to pull the mesh triangles close to the data, but allows very little sliding of the mesh along the data, even with a large spring coefficient.

Optimize over mesh transformations

The second subproblem is to lower the energy by altering the mesh structure. This is done by repeated applications of the three elementary mesh transformations: *edge swap, edge collapse*, and *edge split*. These transformations are illustrated in Fig. 4-12. All the edges are placed into a candidate set and they are randomly removed one at a time. The algorithm first tries to collapse the selected edge. An edge collapse removes the edge and the two incident triangles; it also unifies the vertices at the edge's ends. If the edge collapse does not change the topological type of the mesh, a local optimization process finds a new position for the unified vertex, and if the total energy is reduced, the edge collapse is accepted. If the edge cannot be collapsed, the algorithm considers edge swap and edge split, in that order, with similar local optimizations and consistency checks. If one of the transformations is

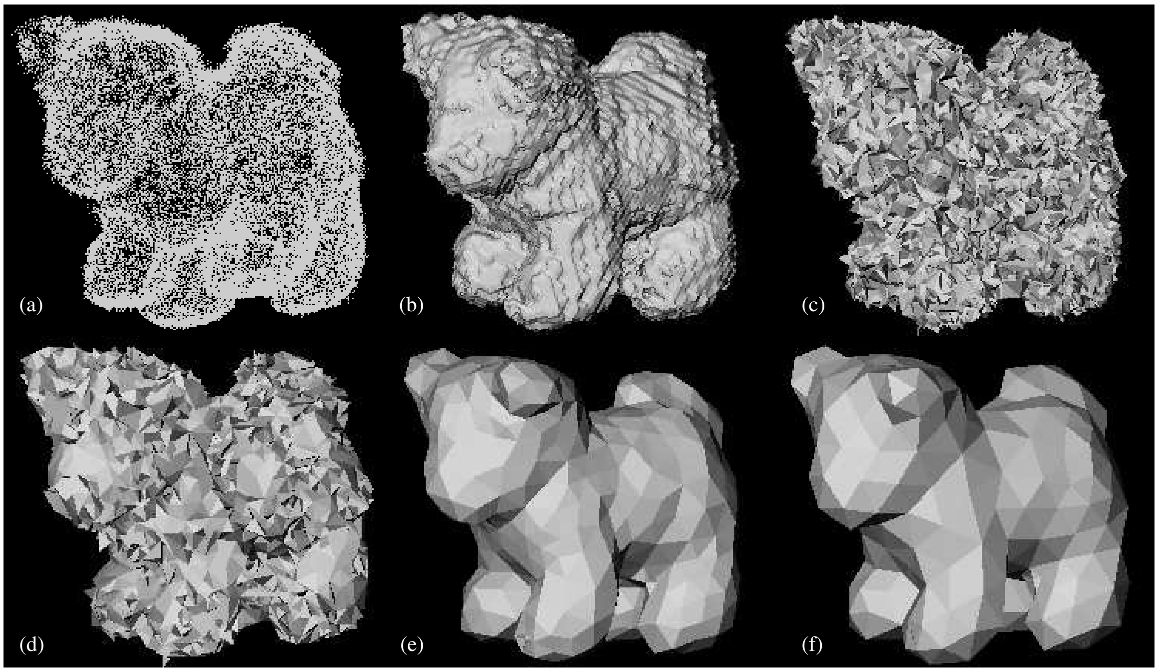


Figure 4-13 (a) Point cloud of toy dog. (b) Initial mesh created by hierarchical space carving, 49,316 faces. (c) First simplification of the mesh, 13,262 faces. (d) Second simplification, 4,610 faces. (e) Third simplification, 1,008 faces. (f) Last simplification, 576 faces.

accepted, all the neighboring edges are added to the candidate set. The process is repeated until the candidate set is empty.

4.3.3 Discussion

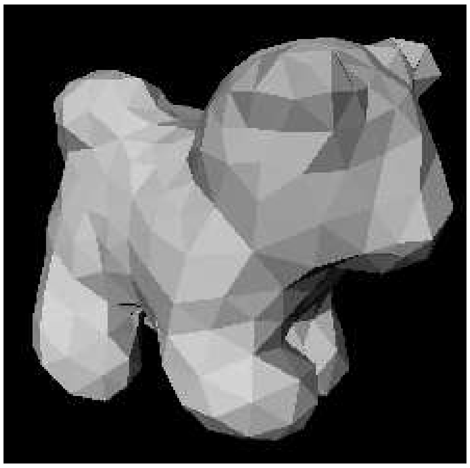
Figure 4-13 shows a typical mesh optimization sequence using the method presented above. We start with a point cloud (a) and a dense initial mesh (b) created by our hierarchical space carving algorithm. Figures 4-13(c)-(f) show four snapshots from the optimization process. The representation penalty c_{rep} is increased monotonically during the iteration. The spring constant κ seemed to be too low at first, resulting in a roughly chiseled mesh, and was kept at a higher constant level during later iterations. We end up with a concise but accurate model of a toy dog.

Chapter 5
View-dependent texturing

5.1 Introduction

If the purpose for scanning objects and reconstructing their surfaces is to display the objects on a computer monitor, obtaining the surface geometry is not enough. Although the coarse polygon mesh in Fig. 5-1(a) is recognizable as a toy husky, we can get a much more realistic picture by texture mapping one or more color images onto the mesh surface. The textured model in Fig. 5-1(b) almost completely hides the polygonal nature of the underlying surface representation, and the intricate fur texture hints at a very detailed geometry, which in fact only exists in the mind of the beholder.

Although texture mapping is currently the most practical approach to adding realistic colors to a geometric model, we could take an approach based more closely on physics.



(a)

(b)

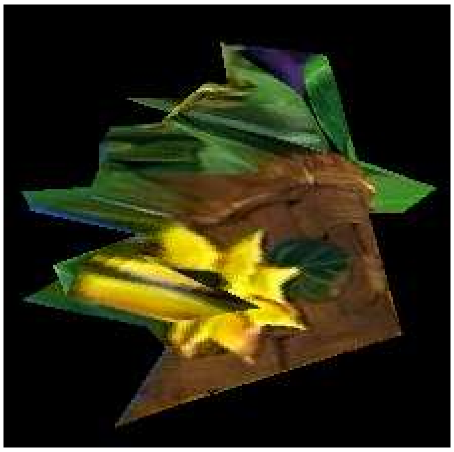
Figure 5-1 (a) The geometry of a toy husky. (b) Realism is much enhanced by texture mapping.

Instead of mapping an image onto the surface, we could map a reflectance function that relates the incoming light to the reflected light. There are many practical difficulties in estimating this function, as it depends on the angles of the incoming and reflected light (with respect to the surface normal) as well as the wavelength composition of the incoming light. Also, interreflections are problematic both for acquisition and rendering. Additionally, in order to create realistic renderings, the geometric description of the surface would have to be very detailed. Even if it is, the resulting images can usually be immediately recognized as computer-generated, because many lighting effects created by detail like the fur in Fig. 5-1(b) are extremely difficult to recreate.

Conventional texture mapping assigns to each surface its own texture map that is independent of the viewing direction. While this approach is straightforward to implement and works well with models that have been directly produced by a computer, it has several shortcomings when used to texture map an approximate geometric model reconstructed from scanned range data. Figure 5-2 demonstrates that the quality of scanned color depends on the view direction. In Fig. 5-2(a) an individual scan of a flower basket is seen from approximately the same direction as it was scanned, and it looks very realistic. Figure 5-2(b) shows the same textured surface from another viewpoint. This second view makes obvious the low quality of both the color and geometry data on surfaces that are nearly parallel to the scanning direction.

The problem shown in Fig. 5-2 can be at least partially overcome by using only the color information obtained from directions approximately normal to each surface patch and then warping the color data slightly so the different images blend together smoothly. However, there is a more fundamental problem with static texturing, as illustrated in Fig. 5-3. There we have only a coarse representation of the flower basket geometry, yet the color variations in Figs. 5-3(a) and (b) suggest fine geometric detail in the weave pattern of the basket. However, when the model in (b) is seen from the viewpoint in (a) or vice versa (as in Figs. 5-3(c) and (d)), the result is an unnatural warping of the weave pattern. This suggests that if texturing is used to convey the illusion of fine geometric features on coarse approximate surfaces,

(a)



(b)

Figure 5-2 View-dependent scan data. (a) A realistic looking textured surface. (b) The same surface looks much less realistic when viewed from a different direction.

there may not exist any single texture that would look natural from every direction.

To avoid these problems we use *view-dependent* texturing of surface models. View-dependent texturing combines a number of images in varying proportions based on the viewing direction. View-dependent texturing was initially used in *image-based rendering* systems [Chen & Williams 93, McMillan & Bishop 95, Gortler *et al.* 96, Levoy & Hanrahan 96], some of which do not use any geometric information, but it is also increasingly being used to texture map geometric models [Debevec *et al.* 96, Niem & Wingbermühle 97, Matsumoto *et al.* 97, Pulli *et al.* 97a].

In the next section we examine how image-based rendering interpolates between suitable pixels from the input images to create pixels in the output image. That analysis is used as justification in the two subsequent sections for our two methods of view-dependent texturing. First, we describe a method for how the complete models created using the methods presented in Chapter 4 can be texture mapped. This method projects a ray through each pixel of a virtual camera to the object, projects the ray-surface intersection back to input images, and combines the colors at those locations using various weights. Then we present another method that we call *view-based rendering*. In view-based rendering the range maps are not

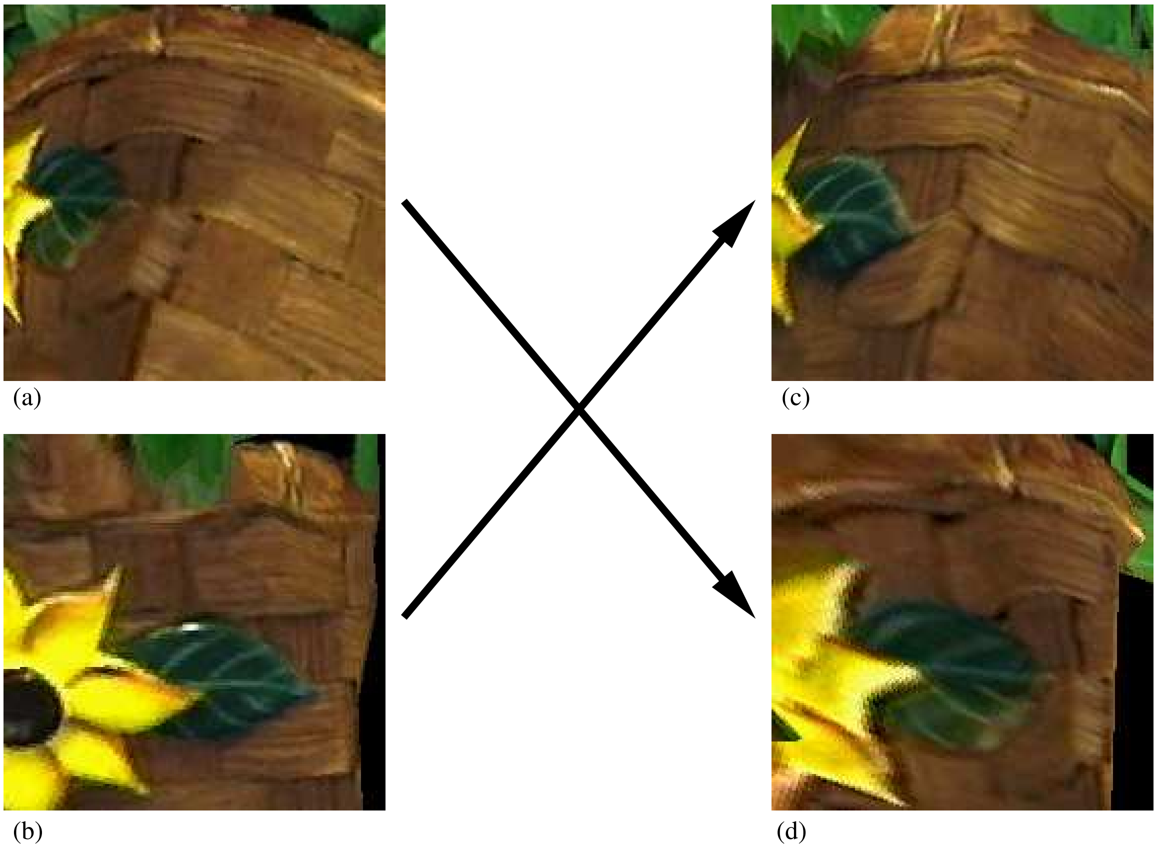


Figure 5-3 No single texture looks good from every direction. (a) and (b) show the same detail of a basket from two directions. (c) and (d) show the data rotated as seen from the other viewpoint.

integrated into a global model. Instead, we approximate the range map of each view by a triangle mesh that is texture mapped using the associated color image. A few such textured meshes are rendered from the viewpoint of a virtual camera, and the views are integrated in image space rather than in 3D. Finally, a discussion section concludes the chapter.

5.2 Image-based rendering

The basic problem in image-based rendering is to compute an image of a scene from a new camera pose, given a set of input images and their corresponding camera poses. A useful abstraction in this context is the *light field function* (also known as the *plenoptic function*).

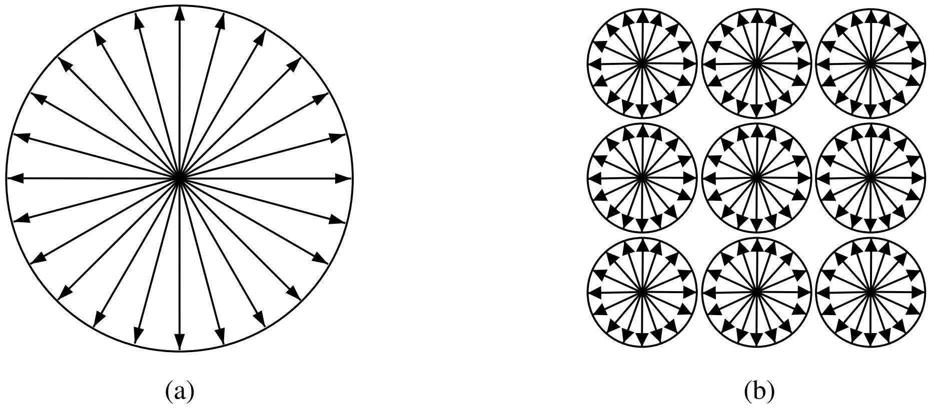


Figure 5-4 (a) A pencil of rays describes the colors of points visible from a given point. (b) The light field function describes the colors of all rays starting from any point.

Levoy and Hanrahan [1996] define the light field as the radiance leaving a point in a given direction. For our purposes, it is more convenient to define the light field as the radiance arriving at a point from a given direction (see Fig. 5-4).

More precisely, we define a *ray* to be a directed half-line originating from a 3D *basepoint*. We may therefore represent a ray as an ordered pair $(\mathbf{x}, \hat{\mathbf{n}}) \in \mathbb{R}^3 \times S^2$, where $\mathbf{x}$ is the ray's basepoint, $\hat{\mathbf{n}}$ is its direction, and S^2 denotes the unit sphere. The light field is then a function

$$f : \mathbb{R}^3 \times S^2 \to \mathbb{R}^3,$$

which assigns to each ray $(\mathbf{x}, \hat{\mathbf{n}})$ an RGB color $f(\mathbf{x}, \hat{\mathbf{n}})$, as shown in Fig. 5-4(b). Note that our rays start from the observer and are directed toward the object, while light propagates in the opposite direction. Thus, $f(\mathbf{x}, \hat{\mathbf{n}})$ measures the radiance at $\mathbf{x}$ in the direction $-\hat{\mathbf{n}}$. The collection of rays starting from a point is called a *pencil*. If we had complete knowledge of the light field function, we could render any view from any location $\mathbf{x}$ by associating each pixel of a virtual camera with a ray (or an average of rays) from the pencil based at $\mathbf{x}$.

The complete light field function is only needed to render entire environments. If we are content to render an individual object, it suffices to know the light field function for the subset of $\mathbb{R}^3 \times S^2$ of "inward" rays originating from points on a convex surface M that encloses the object. Following Gortler *et al.* [1996], we call this simpler function a *lumigraph*. We call the surface M that encloses the object the *lumigraph surface*. Figure 5-5 shows a schematic of the lumigraph domain for the case of a spherical lumigraph surface.

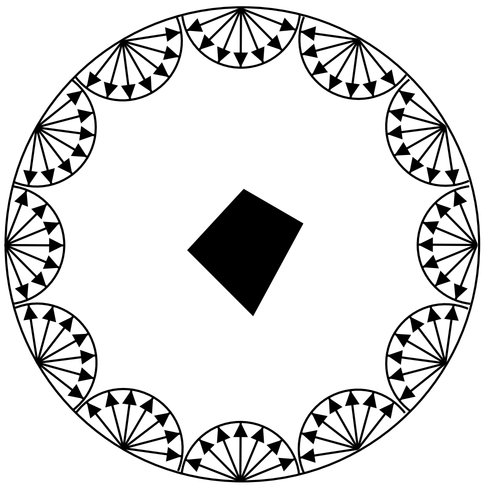


Figure 5-5: A spherical lumigraph surface.

The lumigraph contains all the information needed to synthesize an image from any viewpoint outside the convex hull of the object being modeled. Each pixel in the image corresponds to a ray that intersects the lumigraph surface M at a point, say $\mathbf{x}$. If $\hat{\mathbf{n}}$ is the direction of that ray, then the RGB color value assigned to the pixel is $f(\mathbf{x}, \hat{\mathbf{n}})$.

5.2.1 Distance measures for rays

In practice we will never be able to acquire the full 5D light field function or even a complete 4D lumigraph. Instead we will have a discrete set of images of the scene, taken at some finite resolution. In other words, we have the values of the function for a sample of rays (or more precisely, for local averages of the light field function). To render the scene from a new viewpoint, we need to estimate the values of the function for a set of query rays from its values for the sample rays. Thus, *image-based rendering is an interpolation problem.*

In a generic interpolation problem, one is given the values of a function at a discrete set of sample points. The function value at a new query point is estimated by a weighted average of function values at the sample points, with weights concentrating on samples that are close to the query point. The performance of any interpolation method is critically dependent on the definition of "closeness."

In image-based rendering, the aim is to paint pixels on the image plane of a virtual camera, and therefore we look for rays close to the one associated with some particular pixel. In the next three subsections we examine factors that should be taken into account when interpolating the query ray value from existing rays. It is clear that we should concentrate on sample rays that intersect the object surface close to where the query ray does. However, as we will show, merely considering distance between intersection points of rays with the object surface is only justified for a flat Lambertian surface; in general ray direction should also be considered. The third factor is the sampling density of the surface color.

Ray-surface intersection

Figure 5-6 shows a piece of a lumigraph with several pencils of rays. If we have no information about the object's surface geometry, as in Fig. 5-6, we have to concentrate on pencils that are close to the query ray, and interpolate between rays that are parallel to the query ray. The denser the pencils are on the lumigraph surface M, and the more rays in each pencil, the more similar rays to the query ray we can expect to find. In other words, the expected error is directly proportional to the spacing between sample rays (or inversely proportional to density).

The lumigraph representation has a lot of redundancy, at least if the surface is approximately a Lambertian reflector: there are many rays that see the same point on the object surface. Suppose we don't know the precise object geometry, but we have a rough estimate of the object surface, or at least an average distance from the lumigraph surface to the object surface. In this case, illustrated in Fig. 5-6(b), we let $\mathbf{x}_0$ denote the intersection of the query ray $(\mathbf{x}, \hat{\mathbf{n}})$ with the surface approximation and choose rays in the lumigraph samples that point at $\mathbf{x}_0$. The expected error in our estimate of $f(\mathbf{x}, \hat{\mathbf{n}})$ should now be less than in case (a). Or, we can achieve the same error using far fewer sample rays (i.e., input images).

Figure 5-6(c) illustrates the case where we have accurate information about the object geometry. To estimate $f(\mathbf{x}, \hat{\mathbf{n}})$, we can locate the sample rays that intersect the object surface

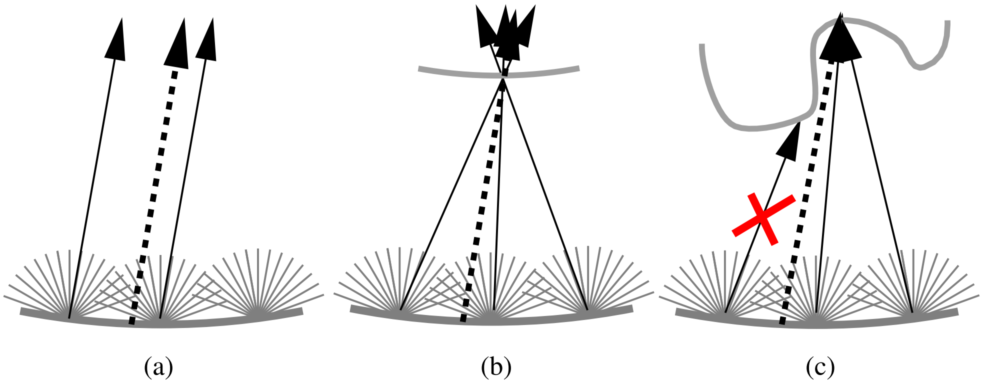


Figure 5-6 The query ray is dotted; sample rays are solid. (a) Choose similar rays. (b) Choose rays pointing to where the query ray seems to meet surface. (c) Choose rays intersecting the surface where the query ray does.

at the same location as the query ray. With an accurate surface description, it is possible to find all the stored rays that intersect the surface where the query ray does, and even detect rays that are directed toward that point but really intersect some other part of the surface first. Naturally, the expected error with a given collection of rays is now the least.

Ray direction

The estimate of the light field function can be improved by weighting more heavily sample rays whose directions are near that of the query ray. There are three justifications for this claim. First, few surfaces reflect incoming light uniformly in every direction. A typical example of this is a specular reflection from a shiny surface, but the appearance of many materials (velvet and hair, for example) varies significantly with viewing direction. In model-based rendering, this angular dependence is captured by the bidirectional reflectance distribution function (BRDF); in image-based rendering it suggests that we favor rays with similar directions.

Second, undetected self-occlusions may cause us to incorrectly conclude that two sample rays intersect the object surface at the same point, giving us a poor estimate of the light field function. If the occlusion is due to a large-scale object feature and we have enough

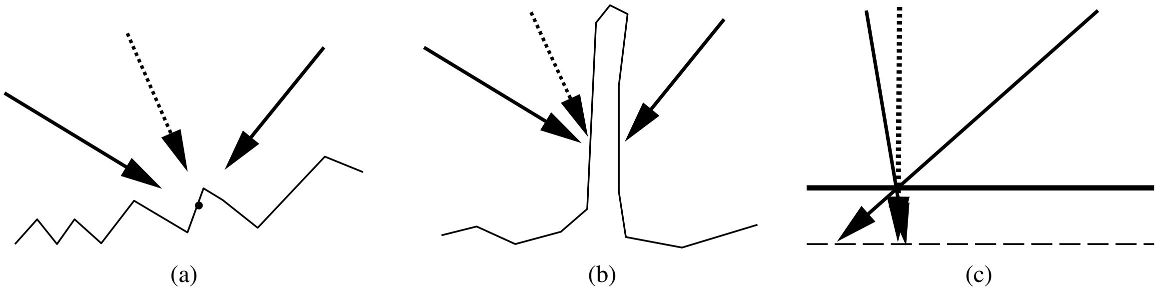


Figure 5-7 (a) Detailed surface geometry can cause occlusions that make the surface appear different from different directions. (b) Thin features can cause a discrepancy between surface distance and spatial distance of intersection points. (c) The more parallel the rays the less damaging an error in an estimate of surface distance.

information about the surface geometry, we can discover the self-occlusion and cull away occluded rays, as in Fig. 5-6(c). However, if the occlusion is due to small-scale surface geometry and we only have an approximation of the surface geometry, the occlusion is much harder to detect, as shown in Fig. 5-7(a). Moreover, if the object has thin features, as illustrated in Figure 5-7(b), then rays may approach the object surface from opposite directions and intersect it at points that are nearby spatially, yet far apart along the surface. We can decrease the likelihood of such errors by more heavily weighting sample rays whose directions are near the direction of the query ray.

Third, as shown in Fig. 5-7(c), when the angle between the query ray and the sample ray is large, small errors in the surface geometry can lead to large errors in the estimate of distance between the intersection points with the object surface. We get more robust results by favoring rays almost parallel to the query ray.

Surface color sampling

The pixels in a camera image do not sample the surface uniformly. As illustrated in Fig. 5-8, the surface is sampled much more densely in areas that are close to perpendicular to the viewing direction, while tilted surfaces are sampled sparsely. The surface area visible to a pixel increases by a factor of $1/\cos\phi$, where ϕ is the angle between the local surface normal

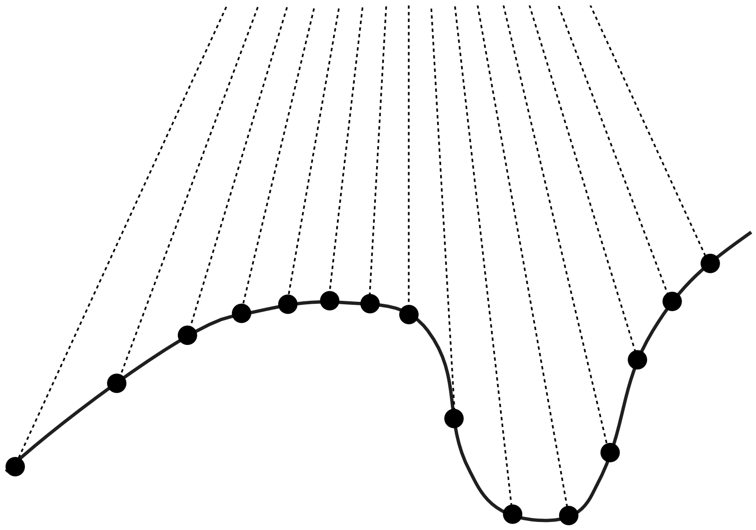


Figure 5-8 The quality of surface color sampling depends on the relative surface orientation with respect to the scanner.

and the direction to the camera. Thus not only is the sampling sparser on tilted surfaces, but also each sample is an average of the color over a larger surface area. A similar situation arises if the input images are taken at different distances from the object surface. Images taken from far away sample the surface more sparsely, and each pixel integrates the colors over a larger area. Therefore surface sampling density should be taken into account when interpolating between input rays. We should assign lower weights to input rays that sample the surface at coarser resolutions than the query ray, and we should apply a low-pass filter to those that sample at finer resolutions.

5.3 View-dependent texturing of complete surfaces

We have implemented the ideas described in the previous section within a viewer that can display objects from arbitrary directions. Although our models are represented as triangle meshes that were created from range scans using the methods presented in the previous chapters, our rendering method can be applied to any surface representation (NURBS patches or subdivision surfaces, for instance).

In the following sections we first describe how we choose which input images are used to texture the surface. Then we explain how we find pixels in the input images that correspond to the object surface location visible to a particular pixel in the viewer. We describe our averaging scheme for combining those rays, and we finally discuss the preprocessing steps that allow the implementation of our texturing method to execute at interactive speeds.

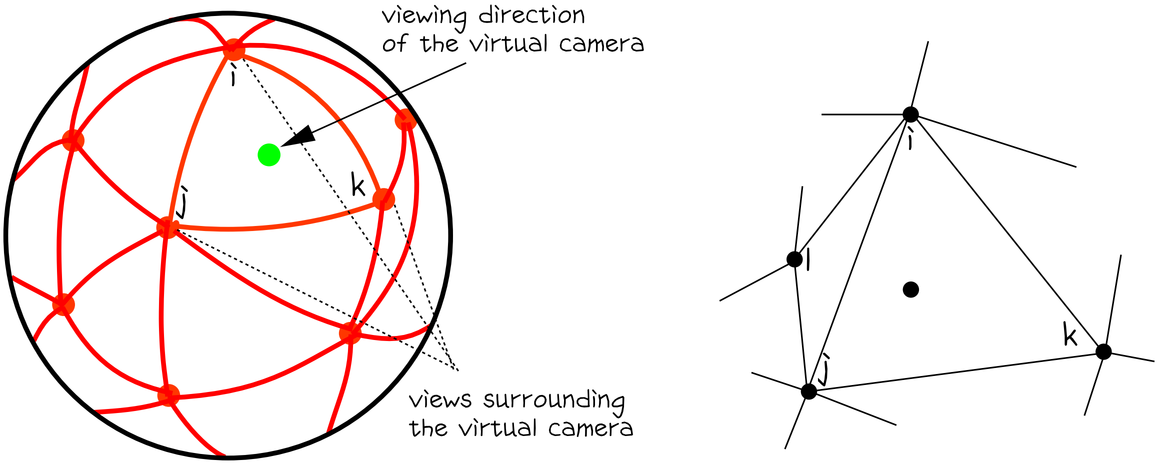
5.3.1 Choosing views

In principle, any camera view that sees the same surface point as the viewer could contribute to the color of the corresponding pixel. However, as we pointed out in Section 5.2.1, views with viewing directions[1] far away from that of the virtual camera should not be used if closer views are available. Otherwise, self-occlusions become much more frequent, so only a small portion, if any, of the surface is likely to be visible both to the viewer and to the distant view. Additionally, small errors in registration, camera calibration, and surface reconstruction lead to larger errors in backprojecting surface points to the color images. In our implementation we only search for compatible rays from three input images that have been taken from nearby viewing directions.

To facilitate the selection of suitable views, the views are organized as illustrated in Fig. 5-9(a). We place vertices corresponding to each of the viewing directions of the input images on a unit sphere, and then compute a Delaunay triangulation of the sphere using those vertices. We use an incremental Delaunay triangulation algorithm and the data structures proposed by Guibas and Stolfi [1985]. That algorithm was designed for triangulating vertices that lie in a plane; some modifications were required to perform the triangulation on a sphere.

When we start rendering a frame we determine the viewing direction of the viewer and place a corresponding vertex on the unit sphere as shown in Fig. 5-9(a). The triangle containing that vertex determines the three views within which we will search for candidate

[1] With a viewing direction of a camera we mean the orientation of the camera's optical axis.



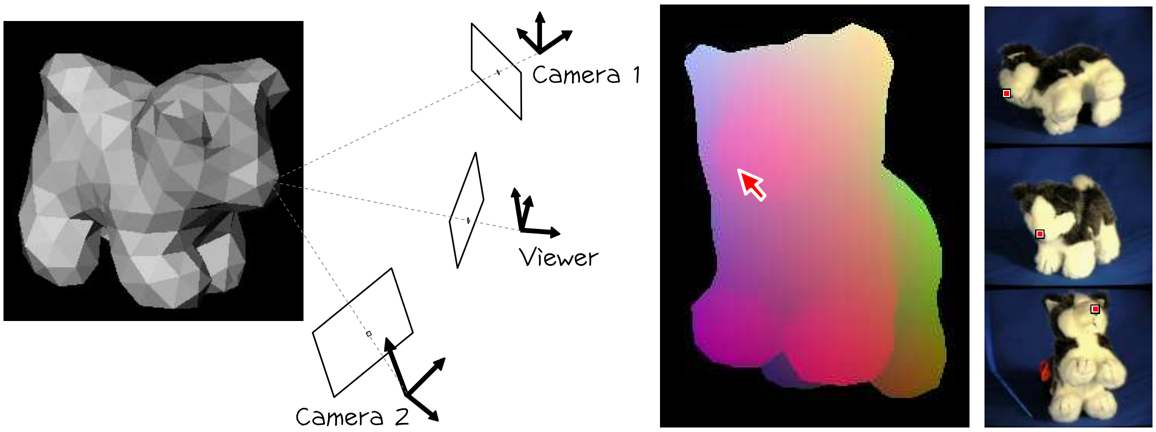
(a) (b)

Figure 5-9 (a) A Delaunay triangulation of the unit sphere is computed for the vertices corresponding to the viewing directions of the input images. The triangle (i, j, k) containing the viewing direction of the virtual camera determines the three views used to search for candidate rays. (b) Though view l is closer to the current viewing direction, view k is a better choice.

rays for the surface points visible to the viewer. Note that these views are not always the three closest views, though the closest one is guaranteed to be among the three. For example, in Fig. 5-9(b) view l is closer to the current view direction than view k. However, we prefer to use view k because i, j, and l all lie to one side of the current view. If there is some part of the surface visible to the viewer but occluded in views i and j, that location is more likely to be visible in view k than in view l.

5.3.2 Finding compatible rays

When we aim our viewer at an object, the first task in determining the color of a particular pixel is to locate the point on the object surface that is visible through that pixel. Figure 5-10(a) illustrates this problem by showing a ray through one of the viewer's pixels ending at its first intersection with the object surface. We obtain candidate rays that might see the same surface point by projecting the intersection point back to the input images. For example, the viewer pixel marked by the arrow in Fig. 5-10(b) corresponds to a point on



(a) (b) (c)

Figure 5-10 **(a)** A ray from the viewers center is cast through a pixel, intersecting the object surface. The intersection point is projected back to color images, producing candidate pixels for coloring the pixel of the original ray in the viewer. **(b)** A false color rendering of the surface geometry uses hardware to find the surface point visible through each pixel. Red, blue, and green channels encode x, y, and z coordinates, respectively. **(c)** The 3D point corresponding to the pixel pointed to in (b) is projected into three color images (the red dot on the nose).

the dog's snout, which projects back to the red dots in the three images in Fig. 5-10(c).

There are two straightforward ways we can use graphics hardware to determine the surface point visible through a given pixel. The method we have chosen to implement (and the one taken by Gortler *et al.* [1996]) is illustrated in Fig. 5-10(b). First, we calculate the axis-aligned bounding box for the triangle mesh representing the object. We then scale and translate the coordinates of each vertex so that the bounding box is transformed to a cube with unit-length sides. Now we can encode the x, y, and z coordinates of each vertex in the red, green, and blue components of its color, so that when the mesh is rendered in the viewer, we get an image like the one in Fig. 5-10(b). Within each triangle, the graphics hardware uses Gouraud shading to interpolate the color, end therefore also the encoded surface location. The surface location visible through a pixel is then given by the pixel's color, after scaling by the dimensions of the original bounding box and translating to the position of the minimum corner of the bounding box.

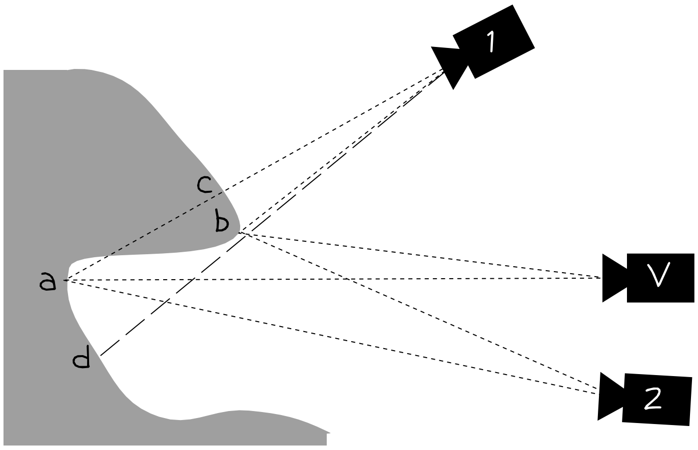


Figure 5-11 The virtual camera sees two points *a* and *b*, and they can be reliably projected back to camera 2. However, *a* and *b* project back to pixels of camera 1 that actually see points *c* and *d*, due to self-occlusion (*c*) and errors in registration, calibration, or surface reconstruction (*d*).

The method described above can only be applied directly to polygonal surface representations. For a more general technique, we could read in the depth buffer after rendering the untextured surface, and convert each depth value to the corresponding 3D point using the known projection and viewing transformations. However, that approach requires over four times as many basic operations (multiplications, divisions, and additions) as the false-color rendering approach.

Once we have determined the surface point corresponding to a viewer pixel, we obtain candidate rays by projecting that point back to the input images as shown in Fig. 5-10(c). To perform these projections, we need to know each camera's internal and external parameters. In our case, the camera calibration procedure used for obtaining depth from stereo (Chapter 2) determines the internal parameters of the camera; the external parameters for each view are obtained by registering the range maps into a common coordinate system (Chapter 3).

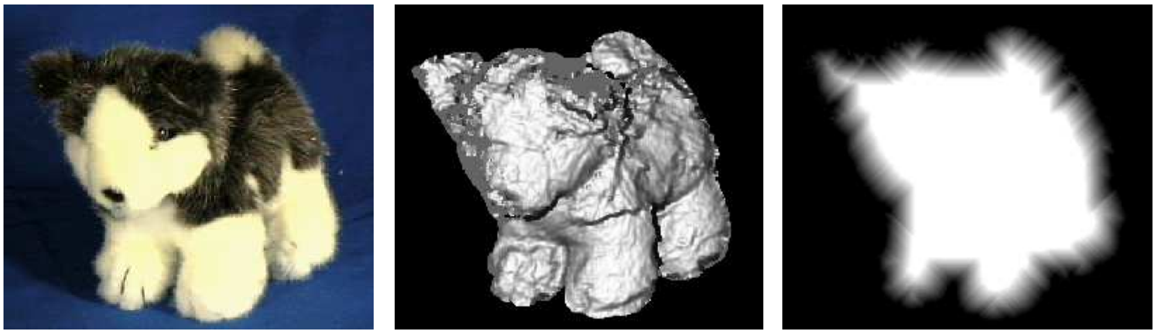
As Fig. 5-11 illustrates, not all the candidate rays obtained through backprojection should be accepted. The virtual camera of the viewer sees two surface points, *a* and *b*, and those points are also clearly visible to camera 2, which can produce reliable candidate rays for coloring the corresponding pixels. However, point *a* is not visible to camera 1 due to self-occlusion; the ray from camera 1 pointing at *a* sees point *c* instead. Point *b*, on the other hand, *is* visible to camera 1, though just barely, but in this case minute errors in camera calibration or registration lead point *b* to project to a pixel that really sees *d* instead. We can detect these problems easily if we retain the original range maps for each camera. For example, we can calculate the distance from point *a* to camera 1 and compare it to the range map value for the pixel *a* projects to. If these distances differ significantly (as they do in this case), then we reject the ray.

5.3.3 Combining rays

The colors of the compatible rays are averaged together via a weighting scheme that uses three different weights: *directional* weight, *sampling quality* weight, and *feathering* weight.

The task of the directional weight is to favor rays originating from views whose viewing direction is close to that of the virtual camera. Not only should a view's weight increase as the current viewing direction moves closer, but the other views' weights should decrease, leaving only the closest view when the viewpoints coincide. Because we use three input views forming a triangle containing the current view, it is natural to choose as weights the *barycentric coordinates* of the current view with respect to the others. For four points lying in a plane, the fourth point can be expressed as a unique weighted sum of the other three points. The barycentric coordinates are those weights; they sum to one and if the fourth point is within the triangle formed by the other three, they are all nonnegative; and they linearly interpolate the three points to produce the fourth. In our case the four points lie on a sphere rather than a plane but we can still compute barycentric coordinates. We radially project the vertex of the current view direction onto the planar triangle formed by the surrounding three views (i.e., the triangle is intersected by a line passing through the point and the center of the sphere). The directional weight of each of the three input views is given by the corresponding barycentric coordinate of the projected point.

The sampling quality weight directly reflects how well a ray samples the surface. We assign to each ray/pixel of each input image a weight that is defined as the cosine of the angle



(a) (b) (c)

Figure 5-12 (a) A view of a toy dog. (b) The sampling quality weight. (c) The feathering weight.

between the local surface normal and the direction from the surface point to the sensor. This weight is illustrated in Fig. 5-12(b) for a view of the toy dog. Notice the uniform dark gray close to the ears and one cheek. The scanner didn't provide any range data for those pixels, so we arbitrarily assign a relatively low weight corresponding to about 65 degrees surface orientation with respect to the sensor.

The feathering weight is used mostly to hide artifacts due to differences in lighting among the input images. Without the feathering weight, the boundaries of the input views become noticeable because a view contributes to pixel colors on one side of a silhouette edge but not on the other. The feathering weight is calculated for each original 2D color image, and it remains constant. As illustrated in Fig. 5-12(c), the feathering weight is zero outside of the object, and it grows linearly to a maximum value of one within the object.

5.3.4 Precomputation for run-time efficiency

The directional weight changes every time the viewer moves with respect to the object, so we have to recompute it again for each frame. However, the sampling quality and feathering weights remain constant, so we can determine them in advance and store them in a lookup table. We preprocess each pixel of each image by calculating these two weights and storing their product in the alpha channel of a pixel, where it is readily accessible.

A large part of the viewer's processing time is spent projecting object surface points onto input images. To correct for radial distortion of the images we have to solve for the roots of a cubic polynomial. We can avoid this calculation by preprocessing the input images (along with associated information such as the weights and range data) to remove the distortions beforehand.

Once we have undistorted input images we can further optimize our viewer by collapsing each view's registration and projection transformations into a single 3×4 matrix that transforms a homogeneous 3D point into a homogeneous 2D image point.

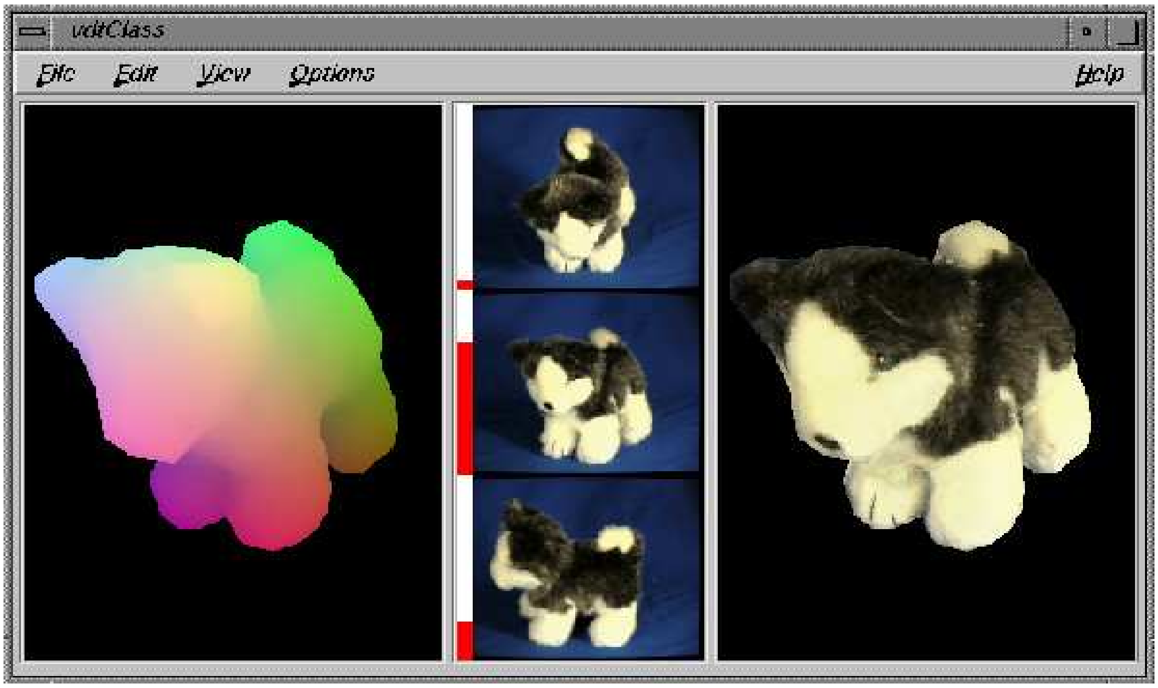


Figure 5-13 An interactive viewer. On the left is a rendering of the geometry in which color encodes the 3D coordinates of the surface point visible through each pixel. In the middle are the three views that have been chosen as inputs. The bar next to each view shows that view's directional weight. The final image is on the right.

5.3.5 Results

We have implemented an interactive viewer for displaying view-dependently textured objects (see Fig. 5-13). Our approach does not require hardware texture mapping, yet we can display complex textured models at interactive rates (5 to 6 frames per second on an SGI O2). The only part of the algorithm that uses hardware graphics acceleration is the rendering of z-buffered Gouraud-shaded polygons to determine which points on the object surface are visible in each frame. The algorithm can be easily modified to work with arbitrary surface descriptions by finding the visible surface points using the z-buffer instead of rendering triangles colored by location.

Figure 5-14 A set of views close to the current viewing direction are chosen (top left), their textured geometries are rendered from the viewpoint of the virtual camera (lower left), and the images are composited into the final result (right).

5.4 View-based rendering

The method described in the previous section follows a traditional approach to displaying scanned objects with colors: first integrate the range scans into a single model, and then paste the color images onto the surface. In this section we propose an alternative to that approach: instead of integrating the range maps into a single model, we model each range map separately. Each of these view-based models is textured with the color image from its associated view, and the different models are integrated in image space during rendering. Figure 5-14 gives an overview of this method, which we refer to as view-based rendering.

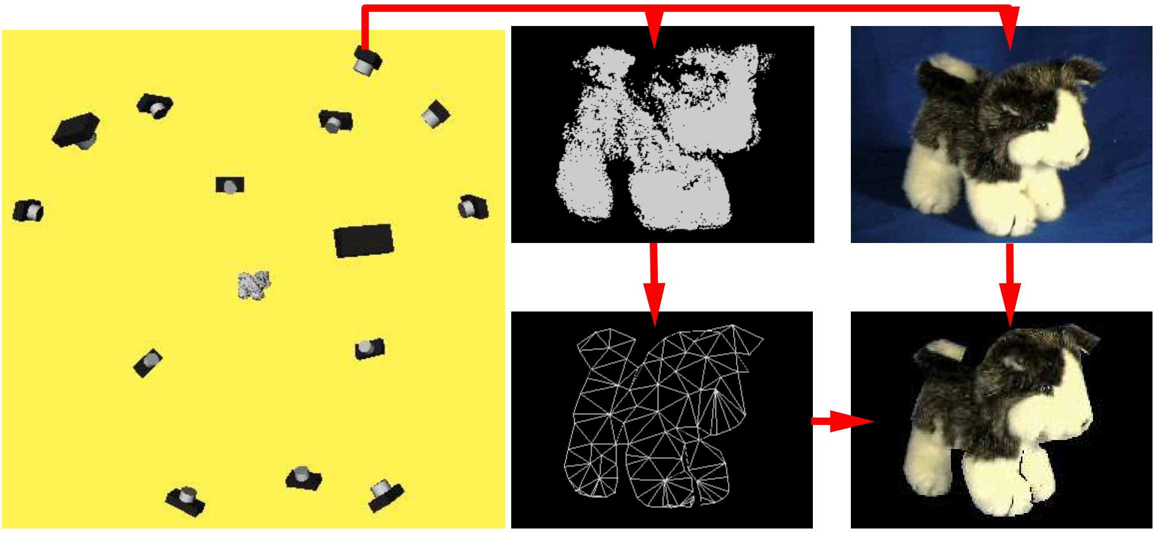


Figure 5-15 A sequence of scans obtains a color image and a range map from each viewpoint around an object. The viewpoints are registered with respect to the object by aligning the range maps. Each range map is approximated by a triangular mesh, which is then textured using the associated color image.

The rest of this section describes in more detail the steps involved in view-based rendering and concludes with the results of our implementation.

5.4.1 View-based modeling

Figure 5-15 summarizes the creation of view-based models. The first steps are the same as in creating a single surface model: object color and geometry are scanned from a set of viewpoints around the object, and the scans are registered with respect to an object-centered coordinate system. However, instead of integrating the range data into a single model, the views are modeled individually, and each model is statically textured using the color image of the corresponding scan.

The attraction of view-based modeling is clear: it is much easier than modeling complete surfaces. We can completely avoid the difficult task of determining the topology and connectivity of the object we are trying to display. In view-based modeling the surface model is implicit: it is a single-valued height field, and modeling becomes merely function approximation.

We have divided the creation of view-based models into the following subtasks:

- Classify each pixel in the scans as belonging to either object or background.

- Approximate the object silhouette by a polygon.

- Create a constrained triangulation that includes the four corners of the range map as well as the silhouette polygon's vertices and edges.

- Modify the 2D triangulation by adding, deleting, and moving vertices, and retriangulating until the mesh projects to pixels marked as object but not to those marked as background. At the same the depth coordinates of the vertices are optimized so that the distance from the range data to the triangulation is minimized.

As a postprocessing step, we remove triangles that are close to perpendicular to the viewing direction, as they are likely to be step edges due to self-occlusion.

5.4.2 Integration in image space

In the method described in Section 5.2, we found the rays pointing at the surface point visible through a pixel by projecting that point back to input images. In view-based rendering, we can take advantage of texture-mapping hardware and view-based models to project the input images to the image plane of the virtual camera. First, we choose three views around the current viewing direction using the method described in Section 5.3.1. We then render each of the textured triangle meshes of the selected views atop one another from the viewpoint of the virtual camera.

Figure 5-16(a) shows the result of this simple approach. In terms of ray interpolation, the graphics hardware interpolates the rays within each view, finding for each pixel a ray that intersects the surface approximately where the pixel's query ray would. However, there is no interpolation between the views: z-buffering essentially chooses the ray from the mesh

(a) (b)

Figure 5-16 (a) The result of combining three views by rendering each of the view-based meshes from the viewpoint of the virtual camera. (b) Using the three weights and soft z-buffering produces a much better result.

that happens to be closest to the camera at each pixel, discarding the others. Because the geometry and registration of the meshes are only approximate, we get a lot of visible artifacts.

We can improve this technique by interpolating rays between different views using the weights introduced in Section 5.3.3 and by dealing correctly with self-occlusions. These improvements, which are demonstrated in in Fig. 5-16(b), are discussed below.

Weights

The three weights used in view-based rendering are exactly the same as those described in Section 5.3.3: directional weight, sampling quality weight, and feathering weight. The directional weight is calculated once for each frame using the barycentric coordinates as discussed previously. We compute a sampling quality weight for each triangle in a mesh by taking the cosine of the angle between the triangle normal and the vector from the triangle centroid to the sensor. This weight is used as the alpha value of the triangle at rendering time. The feathering weight is based on the distance of a pixel from the object silhouette in the input image, and it is applied directly in the alpha channel of the image used to texture map the mesh. The rendering hardware combines the weights by multiplying the alpha value of each triangle with the alpha channel of the texture map.

Self-occlusions

Most self-occlusions are handled during rendering of individual views using back-face culling and z-buffering. When combining renderings of view-based models, part of one view's model may occlude part of another view's model. Unless the surfaces are relatively close to each other, we need to prevent the occluded pixel from contributing to the pixel color. This is done by performing "soft" z-buffering, in software. For each pixel, we first consult the z-buffer information of each separately rendered view and search for the smallest value. Views with z-values within a fixed threshold from the closest are included in the composition; others are excluded. The threshold is chosen to slightly exceed an upper estimate of the combination of the sampling, registration, and polygonal approximation errors.

Figure 5-17 illustrates a potential problem. The view-based surface approximation of the rightmost camera has failed to notice a step edge due to self-occlusion in the data, and has incorrectly connected two surface regions. When we perform soft z-buffering for the pixel corresponding to the dashed line, the incorrectly connected step edge is so much closer than the other view's mesh that we throw away the correct sample. Fortunately, we can avoid this problem by treating the weights as confidence measures while we perform soft z-buffering. If a pixel with a very low confidence value covers a pixel with a high confidence value, we can ignore the low-confidence pixel altogether.

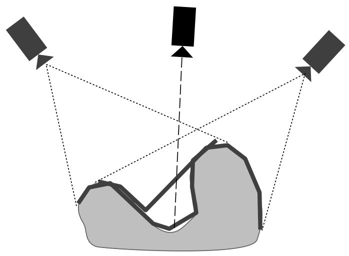


Figure 5-17: Problems with undetected step edges.

Pixel composition algorithm

Figure 5-18 gives the pseudocode for the view composition algorithm. The function `min_reliable_z()` returns the minimum z for a given pixel, unless the closest pixel is a low-confidence point that would occlude a high-confidence point, in which case the z value of the closest high-confidence point is returned.

It is also possible to apply the directional weight using graphics hardware. After we

```
FOR EACH pixel
    z_min        := min_reliable_z( pixel )
    pixel_color  := (0,0,0)
    pixel_weight := 0
    FOR EACH view
        IF z_min <= z[view,pixel] <= z_min+thr_soft_z THEN
            weight       := alpha[view,pixel] * dir_weight[view]
            pixel_color  += weight * color[view,pixel]
            pixel_weight += weight
        ENDIF
    END
    color[pixel] := pixel_color / pixel_weight
END
```

Figure 5-18 Pseudocode for the pixel composition algorithm.

render each view, we have to read the information from the frame buffer. OpenGL allows us to scale each pixel while reading the frame buffer into memory. If we scale the alpha channel by the directional weight, the resulting alpha values contain the final weights.

5.4.3 Results

We have implemented view-based rendering on an SGI Maximum Impact with a 250 MHz MIPS 4400 processor. We first obtain a polygonal approximation consisting of 100–250 triangles for each view. The user is free to rotate, zoom, and pan the object in front of the virtual camera. For each frame, three nearby views are chosen. The texture-mapped polygonal approximations of the views are rendered from the current viewpoint into separate 256×256 windows. The images are combined pixel-by-pixel into a composite image. For the two models that we built, a toy husky and a flower basket, we obtained frame rates of 6–8 frames per second.

Figure 5-19 Our viewer shows the three view-based models rendered from the viewpoint of the virtual camera. The final image is on the bottom right.

5.5 Discussion

5.5.1 Related work

Most of the work related to this chapter deals with image-based rendering. The following four works inspired us to think about view-based texturing in terms of finding and blending rays. Chen [1995] and McMillan and Bishop [1995] modeled environments by storing the light field function around a point. The rays visible from a point are mapped to a cylinder around that point, and new horizontal views are created by warping a portion of the cylinder to the image plane. Both systems allow rotations about the vertical axis, but they do not support continuous translation of the viewpoint. Levoy and Hanrahan [1996] and Gortler *et al.* [1996] developed image synthesis systems that use a lumigraph and that support continuous translation and rotation of the view point. In fact, the term "lumigraph" that we use to describe the 4D slice of the light field is borrowed from Gortler *et al.* Both systems use a cube surrounding the object as the lumigraph surface. To create a lumigraph from digitized images of a real object, Levoy and Hanrahan moved the camera in a regular pattern into a known set of positions, and projected the camera images back to the lumigraph cube. Gortler *et al.* moved a hand-held video camera around an object placed on a capture stage. The capture stage is patterned with a set of concentric circles, allowing the camera pose to be estimated for each image. The rays from the images are projected to the lumigraph surface, and the lumigraph is interpolated from these samples and stored as a grid of 2D images. In both systems, new images are synthesized from a stored grid of 2D images by an interpolation procedure, but Gortler *et al.* use additional geometric information to improve on ray interpolation. One advantage of the lumigraph methods over surface reconstruction techniques is that they can capture the appearance of any object regardless of the complexity of its surface. The main disadvantage of the lumigraph is the difficulty of storing and accessing its enormous representation.

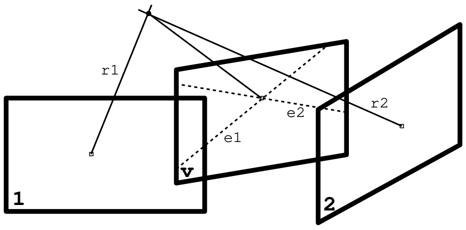


Figure 5-20: Two matching rays correspond to the pixel of the virtual camera where the projections of the rays intersect.

The "algebraic" approach to image-based rendering using pairs of images and pixel correspondences was introduced by Laveau and Faugeras [1994]. It has since been used in several other systems [McMillan & Bishop 95, Werner *et al.* 95, Evgeniou 96]. Given correct dense pixel correspondences, one can calculate the 3D coordinates of surface points visible in both images, and then project these to the image plane of the virtual camera. However, the projection can also be calculated directly without 3D reconstruction. This is illustrated in Fig. 5-20, which shows the stored images 1 and 2, and the image plane of the virtual camera v. Since the pixel marked in image 1 corresponds to the one marked in image 2, their associated rays r_1 and r_2 are assumed to intersect at the same location on the object surface. That point projects to the image v at the intersection of the epipolar lines e_1 and e_2, which are the projections of r_1 and r_2 onto image v. The color of the destination pixel would be a combination of the colors of the input pixels. The pixel correspondence mapping between the input images is not easy to do reliably, especially within regions of homogeneous color. Fortunately, though, the regions where such pixels project have almost constant color, so a projection error of a few pixels typically does not cause visible artifacts.

There is not much published work for view-dependent texturing of complete models. Debevec *et al.* [Debevec *et al.* 96, Debevec 96] developed a system that fits user-generated geometric models of buildings to digitized images by interactively associating image features with model features and adjusting model parameters to fit the images. The basic geometric models are quite simple, but they can be improved by model-based stereo from several input images. View-dependent textures are applied to the buildings by projecting the color images onto the geometric model. If more than one image projects to the same location, images taken from view directions close to that of the virtual camera are weighted more heavily, but the exact weighting function is not described. A blending weight is used close to the boundaries in the input data; our feathering weight performs the same function. Although Debevec *et al.* mention the importance of taking image resolution or sampling quality into account, they leave that for future work.

There are a number of articles related to our work that do not focus on the display of scanned objects, but rather on rapid rendering of complex environments. The idea is to trade unbounded scene complexity for bounded image complexity. Chen and Williams [1993] render a large number of views of a complicated scene and obtain accurate pixel correspondences from the depth values stored along with the pixel colors. The missing views needed for a walk-through of the virtual environment are interpolated from the stored ones. Max and Ohsaki [1995] used similar techniques for rendering trees from precomputed z-buffer views. However, instead of morphing between the precomputed images, their method reprojects the images pixel-by-pixel. Shade *et al.* [1996] partition the geometric primitives in the scene, render images of each group of primitives, and texture map the images onto quadrilaterals, instead of displaying the actual geometry. Mark *et al.* [1997] investigate the use of image-based rendering to increase the frame rate for remotely viewing virtual worlds. Their proposed system would remotely render images from geometric models at 5 frames per second and send them to a local computer that warps and interpolates between two consecutive frames at about 60 frames per second. The 3D warp is the same as the one described by Chen and Williams: a dense triangle mesh is constructed from the z-values of each of the two views being interpolated is performed. Normal vectors and z-values at each pixel are used to locate false connections across step edges between occluding and occluded surfaces. Their logic for discarding unreliable pixels on triangles spanning step edges is similar to ours, but independently derived.

Darsa *et al.* [1997] describe another approach for rapidly displaying complicated environments. The virtual environment is divided into cubes. From the center of each cube, six views (one for each face of the cube) are rendered. The geometry of each view is tessellated (using the z-buffer) into a sparse triangle mesh, which is texture mapped using the rendered color image. A viewer at the center of a cube can simply view the textured polygon meshes stored at the cube walls. If the viewer moves, parts of the scene previously hidden become visible. The textured meshes from other cubes can be used to fill the holes. The authors discuss different weighting schemes for merging meshes from several cubes. Their quality weight is based on the relative orientation of a triangle in the mesh with respect to the center of the camera that created the view, and is essentially the same as our sampling quality weight. They also use another weight related to the distance from the current position to the cube centers, which is different from but analogous to our directional weight.

5.5.2 Contributions

The contributions of this chapter are:

- We have provided an analysis of image-based rendering methods in terms of apparent colors visible along rays, demonstrating that knowledge of surface geometry allows us to use fewer input images. We have also shown the importance of the direction of rays in reducing expected errors when the geometry is known only approximately.

- The preprocessing of the input view directions into a Delaunay triangulation over a sphere provides a practical way of quickly selecting a small set of views that are likely to contain the data required to texture map the object from the current viewpoint. We also get natural interpolating directional weights from the barycentric coordinates of the current viewpoint with respect to the selected views.

- Our first method for view-dependent texturing (using complete models) does not require hardware-supported texture mapping and generalizes to any surface representation, without any problems in texture parametrization over the surface.

- Our second method demonstrates that it is possible to integrate the scan data at rendering time in image space, allowing display of difficult objects that do not allow reliable surface reconstruction.

- Both approaches for view-dependent texturing work for surfaces of arbitrary topology.

Chapter 6

Summary and future work

In this dissertation, we have presented a complete system that begins with scanning both the geometry and color of real objects and finally displays realistic images of those objects from arbitrary viewpoints at interactive speeds. In the following sections we summarize our data acquisition system, registration methods, surface reconstruction methods, and view-dependent texturing methods. After each summary, we discuss possible related future work.

6.1 Data acquisition

The surface reconstruction process begins with scanning both the geometry and the color of an object. For this purpose we have built a stereo camera system. A great advantage of using the same sensor to scan both geometry and color is that these complementary data are automatically registered together. That is, we automatically know which pixel in the color data corresponds to which sampled 3D point on the object surface.

A stereo camera system works by finding pixels in at least two cameras images such that the pixels correspond to the same point on the object surface. Once the correspondence is established, the 3D coordinates of the surface point can be easily determined using optical triangulation. However, determining the pixel correspondences is not a trivial task, and it can be close to impossible if the visible surfaces exhibit uniform observed color. To make the correspondence problem tractable, we use active lighting, i.e., we project a light pattern onto the surface that is used to aid in establishing pixel correspondences. In order to make the task as unambiguous as possible, we use only a single vertical stripe of light, enabling us to reliably match illuminated pixels in the images. All pixels can be matched by repeatedly moving the light stripe slightly and matching the illuminated pixels.

The accuracy of optical triangulation depends on the accuracy of locating the center of the light stripe in the camera images. This can be done reliably using traditional methods only when the scanned object consists of a single planar surface; more complicated geometries may bias the results. For scanners that sweep the light stripe at a constant rate across an object, it is possible to accurately track the motion of the stripe and infer location more reliably from the estimated motion. The incorporation of spacetime analysis [Curless & Levoy 95] significantly improved the quality of our range data.

6.1.1 Future work

Continuous surface digitization

Rather than discretely finding for each pixel the corresponding pixel on the other images, even if that could be done in subpixel accuracy, one could try to fit a piecewise continuous surface to the timespace data of each camera. This surface is a height field over the image, where the "height" is the time. The center of the beam in the i^{th} image would then be the piecewise continuous intersection curve of that surface and a plane that is the constant i over the whole image. One could then imagine to take such curves from two cameras, extend the curves in the camera images into surfaces in 3D using the camera calibration data, and intersect the 3D surfaces. The intersection curve lies on the object surface. This way, instead of digitizing discrete surface points, we could digitize piecewise continuous curves on the surface. However, we could further extend this approach to digitize piecewise continuous surfaces. We could project the isocontour curves into 3D *continuously* in time, intersect the projection surfaces, and create a piecewise continuous surface in 3D parameterized by the image rows of one of the cameras and the time.

6.2 Registration

With a scanner based on optical triangulation we can only scan one side of an object at a time. In order to scan more of the surface, we have to reorient the object (or the scanner), perform the scan, and repeat until all of the surface has been scanned. Unfortunately, each scan represents the range data in the scanner coordinate system, and the data has to be aligned before surfaces can be reconstructed. Provided there is enough overlap with the scans, we can register, or express the range data in a single object-centered coordinate system by aligning both the geometric and color data. Alternatively, we can think of registration as recovering the scanner pose with respect to the object for each scan by aligning the data.

Most registration methods assume an approximate initial registration which is then refined. The registration is an iterative process that improves the registration until the process converges. Typically scanned points in one view are paired with points in another view, using various heuristics so that the paired points would correspond to the same surface points. Using those pairs the surfaces can be moved closer together, and the hope is that the point pairing becomes easier and more reliable. We have developed a better point pairing method that simultaneously establishes all the correspondences between the two scans by projecting the colored geometry onto the scanner image plane, aligning the resulting color images (2D image registration is a much simpler problem), and pairing surface points projecting to the same point on the image plane. The range data is then moved closer using a robust statistical regression method. This approach pairs most points with their true corresponding points in the other view, and typically converges in only a few iterations, while traditional methods take much longer and may even produce inferior results.

When simultaneously registering a large collection of views, the correspondence problem becomes even more severe. Our solution begins with registering each view with several neighboring views in a pairwise fashion. After a view has been registered with respect to another view, it is easy to find corresponding points simply using spatial proximity. The established pairings are stored, and finally the registration transformations for all the views are found simultaneously using the stored point pairs.

6.2.1 Future work

Better 2D image registration

The weak link in our registration method is the 2D registration of an image of a mesh with another color image. The method works fairly well if the images were taken in similar lighting conditions and there is a large overlap. However, when the scans are taken in differing lighting conditions, the image registration becomes unstable and unreliable as the corresponding surface locations have different colors, so a more robust 2D registration method would be useful. Registration methods based on Fourier and wavelet analysis aim to align features in different frequency bands and are less susceptible to slow or constant color differences.

6.3 Surface reconstruction

6.3.1 Initial mesh construction

The most difficult part on surface reconstruction is to correctly determine the connectivity of the surface. If we are first given a rough estimate of the surface with fixed topology, the problem becomes a much easier function estimation problem. We initially used the Hoppe *et al.* [1992] algorithm to create initial meshes, but we noticed that the algorithm is not very robust if the assumptions of uniform and low-noise surface samplings are not met. We developed a hierarchical space carving algorithm that is much more robust in the case of missing and noisy data. While Hoppe *et al.*'s algorithm works with unstructured data, our method requires the input to be in the form of range maps. The algorithm partitions space into an octree and processes it hierarchically. If, based on sensor data, it can detect that an octree node (a cube) is outside the object, it marks that space as such. On the other hand, if it cannot determine that the whole cube is fully inside or outside the object, it subdivides the node and performs similar investigations on the eight descendent cubes. Finally, it extracts the boundary between the outside cubes and all the other cubes. The result is an approximate surface close to the actual surface. The surface is a proper 2D manifold; possible holes due to missing data are filled with a plausible surface. Our inspiration for this approach was the system developed by Curless and Levoy [1996]. Our method is much simpler to implement, processes the space in a hierarchical fashion which leads to savings in execution time and space, and our approach can better deal with thin surfaces.

Future work

Our method produces a blocky surface as each face can only be parallel or perpendicular to the neighboring faces. We could post-process the resulting mesh to better conform to the data by projecting the mesh vertices into the data. We could in fact use the same weighting scheme that Curless and Levoy used. The difference is that they first estimate the surface distance function and then extract the surface. In our case the mesh is extracted first, and now we can better deal with thin surfaces: all we have to do is to make sure that the vertices are not positioned so that the surface self-intersects. In Curless and Levoy's method each range measurement influences the distance function estimate of several cubes; in our case each mesh vertex is influenced by several range measurements.

6.3.2 Mesh optimization

The initial mesh that we construct using our hierarchical space carving algorithm is precisely that: a starting point to obtain a mesh that more closely fits the data and is as sparse as possible while still being able to closely approximate the object. We use the Hoppe *et al.* [1993] algorithm for the mesh optimization. The method iterates between modifying mesh vertex locations to better fit the mesh to the data and stochastically simplifying the mesh.

Future work

There are many possible improvements to the mesh optimization method that we used [Hoppe *et al.* 93], some of which appear in Hoppe's later work [Hoppe 96, Popović & Hoppe 97] dealing with geometry simplification and progressive transmission of geometric models. Hoppe's 1996 algorithm places each of the possible edge collapses in a priority queue based on how much the energy would change after the collapse. Edges are then collapsed in succession until the desired size of the mesh is achieved. With this approach there is no need for the representation penalty c_{rep}. However, it is possible for a good mesh configuration to remain undetected because the edge collapse is the only mesh transformation that is used.

Other possible improvements include using our volumetric information to penalize operations that bring the mesh into free space. It is also possible to try other optimization schedules. For example, Hansen *et al.* [1996] showed that by alternating between periods of vertex insertions and periods of vertex removals, they were able to obtain much better fits to the data than would be possible by just simplifying a dense mesh or refining a sparse mesh.

6.4 View-dependent texturing

The visual appearance of an object is determined not only by its geometry, but also by the surface color and texture. The less uniform the color, the more important role it typically plays. We have two reasons to use view-dependent texturing, where the texture used to determine the color changes depending on the viewing direction, rather than static texturing where each surface element has exactly the same colors independent of the viewing direction. The first reason is that the scan data is view-dependent: the accuracy and sampling density of both geometric and color data depend on the relative orientations of the surface and scanning direction. The second and more important reason is that we can use view-dependent texturing to compress geometry. The original surface may contain fine detailed geometry and thus appears different when viewed from different directions. Even if our approximation does not directly capture that fine geometry, we can capture its appearance using view-dependent texturing.

We have developed two approaches for view-dependent texturing of reconstructed surfaces using color images. The first method in essence projects a few color images onto an approximate surface model. In practice, the projection starts from the virtual viewer where a ray is projected from the viewpoint through a pixel. The ray is intersected with the surface model, and the intersection point is projected back to the color images, producing a candidate color from each image to the original pixel in the viewer. The candidate colors are blended using weights that depend on how closely the viewing direction of a color image matches that of the virtual viewer, how well the candidate pixel samples the surface, and how close the candidate pixel is to the object silhouette.

In the second method the range scans are modeled individually; there is no single complete surface model. The view-based models are textured with the associated color images, and the virtual viewer separately renders several of these models from the current viewpoint. Colors from the same pixel of the separate partial images are combined using the same weights as in the first method.

While the first method integrates the geometric data in 3D and then projects color data onto the image using the geometric model, the second method combines color and geometry for each scan and integrates the data of several scans in run-time and in screen space.

6.4.1 Future work

Better automatic mesh construction

Currently we build the view-based models interactively. The user first interactively separates the object from the background. Then the user adds vertices until he or she is satisfied with the polygonal approximation. While the user decides the pixel locations of new vertices, the program optimizes the depths of the vertices (using the conjugate gradients method [Press *et al.* 92]) to keep the mesh close to the data. In the end, the user selects the triangles that should be kept, and the program discards the triangles that project to background or cross step edges.

Building view-based models is not a central part of our work; instead, our main goal in view-based rendering is to demonstrate the feasibility of the idea. However, for a working modeling system it is clearly desirable to automate this task. Garland and Heckbert have recently developed a fast mesh simplification method that seems like a good candidate [Garland & Heckbert 97].

An alternative approach

As previously mentioned, our first approach can be used with any kind of surface representation, not just with triangle meshes. With triangle meshes, we are able to use texture mapping hardware to speed up rendering. Rather than painting a pixel at a time, this approach would texture map the object a triangle at a time.

The preprocessing consists of projecting the vertices of the mesh into each input image and determining the visibility of the triangle in that view using the triangle's orientation and the view's range data. With each triangle we store pointers to the images in which it is visible, along with texture coordinates for the vertices and an average sampling quality weight. At run-time the program chooses for each triangle that is visible from the viewer a few (e.g., 1–3) views whose viewing directions do not differ much from the current direction, calculates a directional weight for each selected view, multiplies it by the sampling quality weight, and scales the resulting weights of the selected texture patches so that they add to one. Then the selected textures can each be rendered to the same triangle using the scaled weight as a blending factor.

Relighting

Our view-based texturing approach can only be used to view objects in the lighting conditions that prevailed when the color images were obtained. It is possible to extend our method to allow relighting of the object in different illumination conditions. The idea is to store a reflectance distribution function in place of a color for each pixel of each input image.

A practical setup would require arranging several light sources at various known positions around the working volume of the scanner. First, the object geometry is scanned once. Then, without moving the object or the cameras, several color images are taken, each in different known lighting conditions. From this information, we can estimate a unidirectional reflectance distribution function for each pixel. The distribution gives the observed radiance as a function of wavelength and the direction of the incoming light. Some results using synthetic models and lighting are reported by Wong *et al.* [1997].

In our current method, the color value of a contributing image ray is determined by looking up a stored RGB value. The only difference in the proposed method would be to change that lookup to an evaluation of the reflectance function associated with the ray. Notice that this approach would automatically encode many lighting effects, such as self-shadowing, diffuse inter-object reflections, and to a certain degree, even specular highlights, without the need to model them explicitly.

Bibliography

[Barnard & Fischler 82] S. Barnard and M. Fischler. Computational stereo. *Computing Surveys*, 14(4):554–572, 1982.

[Bergevin *et al.* 95] R. Bergevin, D. Laurendeau, and D. Poussart. Registering range views of multipart objects. *Computer Vision and Image Understanding*, 61(1):1–16, January 1995.

[Besl & McKay 92] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Patt. Anal. Machine Intell.*, 14(2):239–256, February 1992.

[Besl 86] P. J. Besl. *Surfaces in early range image understanding*. PhD dissertation, University of Michigan, Ann Arbor, 1986.

[Besl 88] P. J. Besl. *Surfaces in range image understanding*. Springer-Verlag, 1988.

[Besl 90] P. J. Besl. The free-form surface matching problem. In H. Freeman, editor, *Machine Vision for Three-Dimensional Scenes*. Academic, New York, 1990.

[Blais & Levine 93] G. Blais and M. D. Levine. Registering multiview range data to create 3d computer objects. Technical Report TR-CIM-93-16, Centre for Intelligent Machines, McGill University, 1993.

[Blais & Levine 95] G. Blais and M. D. Levine. Registering multiview range data to create 3d computer objects. *IEEE Trans. Patt. Anal. Machine Intell.*, 17(8):820–824, August 1995.

[Champleboux *et al.* 92] G. Champleboux, S. Lavallée, R. Szeliski, and L. Brunie. From accurate range imaging sensor calibration to accurate model-based 3-d object localization. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 83–89, June 1992.

[Chen & Medioni 92] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, April 1992.

[Chen & Williams 93] S. E. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, August 1993.

[Chen 94] Y. Chen. *Description of Complex Objects Using Multiple Range Images*. PhD dissertation, Institute for Robotics and Intelligent Systems, University of Southern California, 1994. Also technical report IRIS-94-328.

[Chen 95] S. E. Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH 95 Conference Proceedings*, pages 29–38. ACM SIGGRAPH, Addison Wesley, August 1995.

[Chien *et al.* 88] C. H. Chien, Y. B. Sim, and J. K. Aggarwal. Generation of volume/surface octree from range data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '88)*, pages 254–260, June 1988.

[Chua & Jarvis 96] C. Chua and R. Jarvis. 3-D free-form surface registration and object recognition. *International Journal of Computer Vision*, 17(1):77–99, January 1996.

[Curless & Levoy 95] B. Curless and M. Levoy. Better optical triangulation through spacetime analysis. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, pages 987–994, June 1995.

[Curless & Levoy 96] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH 96 Conference Proceedings*, pages 303–312. ACM SIGGRAPH, Addison Wesley, August 1996.

[Curless 97] B. L. Curless. *New methods for surface reconstruction from range images*. PhD dissertation, Department of Electrical Engineering, Stanford University, 1997.

[Darsa *et al.* 97] L. Darsa, B. C. Silva, and A. Varshney. Navigating static environments using image-space simplification and morphing. In *Proc. 1997 Symposium on Interactive 3D graphics*, April 1997.

[Davies 90] E. R. Davies. *Machine vision : theory, algorithms, practicalities*. Academic, 1990.

[Debevec 96] P. Debevec. *Modeling and rendering architecture from photographs*. PhD dissertation, Department of Computer Science, University of California at Berkeley, 1996.

[Debevec *et al.* 96] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996.

[Dorai *et al.* 96] C. Dorai, G. Wang, A. K. Jain, and C. Mercer. From images to models: Automatic 3D object model construction from multiple views. In *Proceedings of the 13th IAPR International Conference on Pattern Recognition*, pages 770–774, 1996.

[Eggert *et al.* 96a] D. W. Eggert, A. W. Fitzgibbon, and R. B. Fisher. Simultaneous registration of multiple range views for use in reverse engineering. In *Proceedings of the 13th IAPR International Conference on Pattern Recognition*, pages 243–247, 1996.

[Eggert *et al.* 96b] D. W. Eggert, A. W. Fitzgibbon, and R. B. Fisher. Simultaneous registration of multiple range views for use in reverse engineering. Technical Report 804, Dept. of Artificial Intelligence, University of Edinburgh, 1996.

[Evgeniou 96] T. Evgeniou. Image based rendering using algebraic techniques. Technical Report A.I. Memo No. 1592, Massachusetts Institute of Technology, 1996.

[Faugeras & Hebert 86] O. D. Faugeras and M. Hebert. The representation, recognition, and locating of 3-d objects. *International Journal of Robotic Research*, 5(3):27–52, Fall 1986.

[Foley *et al.* 90] J. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics principles and practice*. Addison-Wesley, 2nd edition, 1990.

[Friedman *et al.* 77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[Gagnon *et al.* 94] H. Gagnon, M. Soucy, R. Bergevin, and D. Laurendeau. Registration of multiple range views for automatic 3-d model building. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 581–586, 1994.

[Garland & Heckbert 97] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997.

[Godin *et al.* 94] G. Godin, M. Rioux, and R. Baribeau. Three-dimensional registration using range and intensity information. In *Proc. SPIE vol. 2350: Videometrics III*, pages 279–290, 1994.

[Golub & Van Loan 96] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[Gortler *et al.* 96] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996.

[Guibas & Stolfi 85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[Hansen *et al.* 96] M. Hansen, C. Kooperberg, and S. Sardy. Triogram models. Technical Report No. 304, Dept. of Statistics, University of Washington, Seattle, WA, 1996.

[Hebert *et al.* 95] M. Hebert, K. Ikeuchi, and H. Delingette. A spherical representation for recognition of free-form surfaces. *IEEE Trans. Patt. Anal. Machine Intell.*, 17(7):681–690, July 1995.

[Higuchi *et al.* 94] K. Higuchi, H. Delingette, M. Hebert, and K. Ikeuchi. Merging multiple views using a spherical representation. In *Proceedings of the 2nd CAD-based Vision Workshop*, pages 124–131, February 1994.

[Hilton *et al.* 96] A. Hilton, A. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Computer Vision – ECCV '96*. Springer, April 1996.

[Hoppe 94] H. Hoppe. *Surface reconstruction from unorganized points*. PhD dissertation, Dept. of Computer Science and Engineering, Univ. of Washington, June 1994.

[Hoppe 96] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96*, pages 99–108, August 1996.

[Hoppe *et al.* 92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of SIGGRAPH '92*, pages 71–78, July 1992.

[Hoppe *et al.* 93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

[Hoppe *et al.* 94] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of SIGGRAPH '94*, July 1994.

[Horn 87] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, April 1987.

[Jin *et al.* 95] H. Jin, T. Duchamp, H. Hoppe, J. A. McDonald, K. Pulli, and W. Stuetzle. Surface reconstruction from misregistered data. In *Proc. SPIE vol. 2573: Vision Geometry IV*, pages 324–328, 1995.

[Johnson & Hebert 97] A. Johnson and M. Hebert. Surface registration by matching oriented points. In *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling*, pages 121–128, May 1997.

[Kang *et al.* 95] S. B. Kang, J. A. Webb, C. L. Zitnick, and T. Kanade. A multibaseline stereo system with active illumination and real-time image acquisition. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, pages 88–93, June 1995.

[Klette & Zamperoni 96] R. Klette and P. Zamperoni. *Handbook of image processing operators*. John Wiley & Sons, 1996.

[Koivunen & Vezien 94] V. Koivunen and J.-M. Vezien. Multiple representation approach to geometric model construction from range data. In *Proceedings of the 2nd CAD-based Vision Workshop*, pages 132–139, February 1994.

[Laurendeau *et al.* 96] D. Laurendeau, G. Roth, and L. Borgeat. Optimization algorithms for range image registration. In *Proc. Vision Interface*, pages 141–151, 1996.

[Laveau & Faugeras 94] S. Laveau and O. D. Faugeras. 3-d scene representation as a collection of images and fundamental matrices. Technical Report RR 2205, INRIA, France, 1994. Available from ftp://ftp.inria.fr/INRIA/tech-reports/RR/RR-2205.ps.gz.

[Levoy & Hanrahan 96] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996.

[Li & Crebbin 94] A. Li and G. Crebbin. Octree encoding of objects from range images. *Pattern Recognition*, 27(5):727–739, May 1994.

[Loop 87] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, Univ. of Utah, 1987.

[Lorensen & Cline 87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[Mark *et al.* 97] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proc. 1997 Symposium on Interactive 3D graphics*, April 1997.

[Masuda & Yokoya 95] T. Masuda and N. Yokoya. A robust method for registration and segmentation of multiple range images. *Computer Vision and Image Understanding*, 61(3):295–307, May 1995.

[Masuda *et al.* 96] T. Masuda, K. Sakaue, and N. Yokoya. Registration and integration of multiple range images for 3-D model constuction. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 879–883, June 1996.

[Matsumoto *et al.* 97] Y. Matsumoto, H. Terasaki, K. Sugimoto, and T. Arakawa. A portable three-dimensional digitizer. In *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling*, pages 197–204, May 1997.

[Max & Ohsaki 95] N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In *Eurographics Rendering Workshop 1995*, pages 74–81;359–360. Eurographics, June 1995.

[McMillan & Bishop 95] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH 95 Conference Proceedings*, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995.

[Niem & Wingbermühle 97] W. Niem and J. Wingbermühle. Automatic reconstruction of 3d objects using a mobile monoscopic camera. In *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling*, pages 173–180, May 1997.

[Okutomi & Kanade 93] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Trans. Patt. Anal. Machine Intell.*, 15(4):353–363, April 1993.

[Pito 96] R. Pito. Mesh integration based on co-measurements. In *Proc. IEEE Int. Conf. on Image Processing, Special Session on Range Image Analysis*, 1996.

[Popović & Hoppe 97] J. Popović and H. Hoppe. Progressive simplicial complexes. In *Proceedings of SIGGRAPH '97*, August 1997.

[Potmesil 83] M. Potmesil. Generating models for solid objects by matching 3d surface segments. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pages 1089–1093, August 1983.

[Press *et al.* 92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 2nd edition, 1992.

[Pulli *et al.* 97a] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle. Surface modeling and display from range and color data. In *Keynote address at International Conference on Image Analysis and Processing '97*, Lecture Notes in Computer Science 1310, pages 385–397. Springer-Verlag, 1997.

[Pulli *et al.* 97b] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Proc. 8th Eurographics Workshop on Rendering*, June 1997.

[Pulli *et al.* 97c] K. Pulli, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle. Robust meshes from multiple range maps. In *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling*, pages 205–211, May 1997.

[Rousseeuw & Leroy 87] P. Rousseeuw and A. Leroy. *Robust regression & outlier detection*. John Wiley & Sons, 1987.

[Rousseeuw & van Zomeren 90] P. Rousseeuw and B. van Zomeren. Unmasking multivariate outliers and leverage points. *Journal of American Statistical Association*, 85(411):633–651, 1990.

[Rutishauser *et al.* 94] M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitrarily shaped objects. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 573–580, 1994.

[Shade *et al.* 96] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996.

[Shum *et al.* 95] H. Shum, K. Ikeuchi, and R. Reddy. Principal component analysis with missing data and its application to polyhedral object modeling. *IEEE Trans. Patt. Anal. Machine Intell.*, 17(9):854–867, September 1995.

[Snyder 92] J. Snyder. Interval analysis for computer graphics. In *Proceedings of SIGGRAPH '92*, pages 121–130, July 1992.

[Soucy & Laurendeau 95] M. Soucy and D. Laurendeau. A dynamic integration algorithm to model surfaces from multiple range views. *Machine Vision and Applications*, 8(1):53–62, 1995.

[Stein & Medioni 90] F. Stein and G. Medioni. Toss-a system for efficient three dimensional object recognition. In *Proc. DARPA Image Understanding Workshop*, 1990.

[Stoddart & Hilton 96] A. J. Stoddart and A. Hilton. Registration of multiple point sets. In *Proceedings of the 13th IAPR International Conference on Pattern Recognition*, pages 40–44, 1996.

[Szeliski & Lavallée 94] R. Szeliski and S. Lavallée. Matching 3-d anatomical surfaces with non-rigid deformations using octree-splines. In *Proc. IEEE Workshop on Biomedical Image Analysis*, pages 144–153, 1994.

[Szeliski & Shum 97] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of SIGGRAPH '97*, pages 251–258, August 1997.

[Szeliski 93] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

[Tarbox & Gottschlich 94] G. Tarbox and S. Gottschlich. IVIS: An integrated volumetric inspection system. In *Proceedings of the 1994 Second CAD-Based Vision Workshop*, pages 220–227, February 1994.

[Taubin 95] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of SIGGRAPH '95*, pages 351–358, August 1995.

[Tsai 87] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.

[Turk & Levoy 94] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH '94*, pages 311–318, July 1994.

[Wang & Jepson 94] Z. Wang and A. Jepson. A new closed-form solution for absolute orientation. In *Proc. Conf. on Computer Vision and Pattern Recognition*, pages 129–134, June 1994.

[Weik 97] S. Weik. Registration of 3-d partial surface models using luminance and depth information. In *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling*, pages 93–100, May 1997.

[Werner *et al.* 95] T. Werner, R. D. Hersch, and V. Hlaváč. Rendering real-world objects using view interpolation. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, pages 957–962, June 1995.

[Wong *et al.* 97] T.-T. Wong, P.-A. Heng, S.-H. Or, and W.-Y. Ng. Image-based rendering with controllable illumination. In *Proc. 8th Eurographics Workshop on Rendering*, June 1997.

[Zhang 94] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152, 1994.

Vita

Kari Pulli began his studies in Electrical Engineering at the University of Oulu in Finland in 1986. Starting in 1987 he also began studying Economics at the University of Vaasa. He studied 1989–90 at the University of Minnesota (Minneapolis, MN) through the ISEP program, and received a Bachelor in Computer Science degree with high distinction. In 1990–91 Kari studied at the University of Paderborn in Germany as a Rotary International scholar, where he wrote his master's thesis. He received his M.Sc. in Electrical Engineering from the University of Oulu in 1991 (with high distinction), and also received the Finnish Technical Society's national prize for the "Diploma thesis of 1991". During 1991–93 he worked as a researcher at the University of Oulu and obtained the degree of Licentiate in Technology in Computer Engineering. In 1993 he began graduate studies at University of Washington as a Fulbright scholar, and received his M.Sc. degree in 1995 and his Ph.D. degree in 1997. In 1998, Kari will join the Department of Computer Science and Electrical Engineering at Stanford University as a Research Associate.