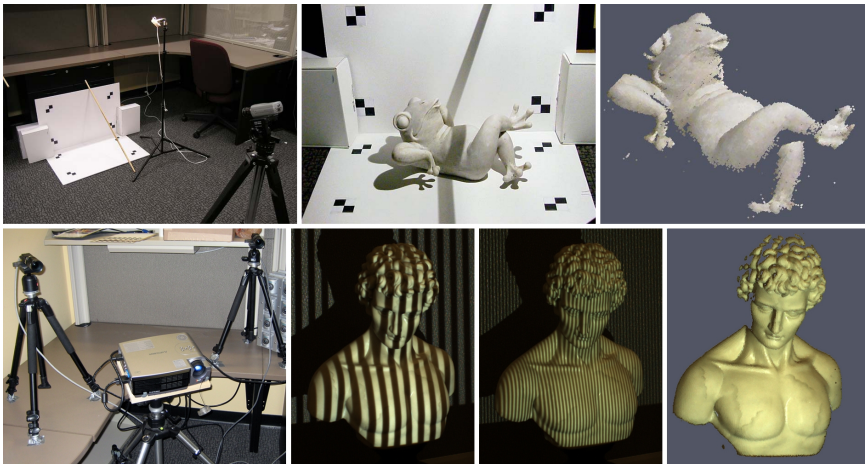
Build Your Own 3D Scanner: 3D Photography for Beginners



SIGGRAPH 2009 Course Notes

Wednesday, August 5, 2009

Douglas Lanman
Brown University
dlanman@brown.edu

Gabriel Taubin
Brown University
taubin@brown.edu

Abstract

Over the last decade digital photography has entered the mainstream with inexpensive, miniaturized cameras routinely included in consumer electronics. Digital projection is poised to make a similar impact, with a variety of vendors offering small form factor, low-cost projectors. As a result, active imaging is a topic of renewed interest in the computer graphics community. In particular, low-cost homemade 3D scanners are now within reach of students and hobbyists with a modest budget.

This course provides a beginner with the necessary mathematics, software, and practical details to leverage projector-camera systems in their own 3D scanning projects. An example-driven approach is used throughout, with each new concept illustrated using a practical scanner implemented with off-the-shelf parts. First, the mathematics of triangulation is explained using the intersection of parametric and implicit representations of lines and planes in 3D. The particular case of ray-plane triangulation is illustrated using a scanner built with a single camera and a modified laser pointer. Camera calibration is explained at this stage to convert image measurements to geometric quantities. A second example uses a single digital camera, a halogen lamp, and a stick. The mathematics of rigid-body transformations are covered through this example. Next, the details of projector calibration are explained through the development of a classic structured light scanning system using a single camera and projector pair.

A minimal post-processing pipeline is described to convert the point clouds produced by the example scanners to watertight meshes. Key topics covered in this section include: surface representations, file formats, data structures, polygonal meshes, and basic smoothing and gap-filling operations. The course concludes by detailing the use of such models in rapid prototyping, entertainment, cultural heritage, and web-based applications. An updated set of course notes and software are maintained at `http://mesh.brown.edu/dlanman/scan3d`.

Prerequisites

Attendees should have a basic undergraduate-level knowledge of linear algebra. While executables are provided for beginners, attendees with prior knowledge of Matlab, C/C++, and Java programming will be able to directly examine and modify the provided source code.

Speaker Biographies

Douglas Lanman
Brown University
dlanman@brown.edu
http://mesh.brown.edu/dlanman

Douglas Lanman is a fourth-year Ph.D. student at Brown University. As a graduate student his research has focused on computational photography, particularly in the use of active illumination for 3D reconstruction. He received a B.S. in Applied Physics with Honors from Caltech in 2002 and a M.S. in Electrical Engineering from Brown University in 2006. Prior to joining Brown, he was an Assistant Research Staff Member at MIT Lincoln Laboratory from 2002-2005. Douglas has worked as an intern at Intel, Los Alamos National Laboratory, INRIA Rhône-Alpes, Mitsubishi Electric Research Laboratories (MERL), and the MIT Media Lab.

Gabriel Taubin
Brown University
taubin@brown.edu
http://mesh.brown.edu/taubin

Gabriel Taubin is an Associate Professor of Engineering and Computer Science at Brown University. He earned a Licenciado en Ciencias Matemáticas from the University of Buenos Aires, Argentina in 1981 and a Ph.D. in Electrical Engineering from Brown University in 1991. He was named an IEEE Fellow for his contributions to three-dimensional geometry compression technology and multimedia standards, won the Eurographics 2002 Günter Enderle Best Paper Award, and was named an IBM Master Inventor. He has authored 58 reviewed book chapters, journal or conference papers, and is a co-inventor of 43 international patents. Before joining Brown in the Fall of 2003, he was a Research Staff Member and Manager at the IBM T. J. Watson Research Center since 1990. During the 2000-2001 academic year he was Visiting Professor of Electrical Engineering at Caltech. His main line of research has been related to the development of efficient, simple, and mathematically sound algorithms to operate on 3D objects represented as polygonal meshes, with an emphasis on technologies to enable the use of 3D models for web-based applications.

Course Outline

First Session: 8:30 am – 10:15 am

8:30	*All*	Introduction
8:45	*Taubin*	The Mathematics of 3D Triangulation
9:05	*Lanman*	3D Scanning with Swept-Planes
9:30	*Lanman*	Camera and Swept-Plane Light Source Calibration
10:00	*Taubin*	Reconstruction and Visualization using Point Clouds

Break: 10:15 am – 10:30 am

Second Session: 10:30 am – 12:15 pm

10:30	*Lanman*	Structured Lighting
10:45	*Lanman*	Projector Calibration and Reconstruction
11:00	*Taubin*	Combining Point Clouds from Multiple Views
11:25	*Taubin*	Surface Reconstruction from Point Clouds
11:50	*Taubin*	Elementary Mesh Processing
12:05	*All*	Conclusion / Q & A

Contents

Chapter 1

Introduction to 3D Photography

Over the last decade digital photography has entered the mainstream with inexpensive, miniaturized cameras routinely included in consumer electronics. Digital projection is poised to make a similar impact, with a variety of vendors offering small, low-cost projectors. As a result, active imaging is a topic of renewed interest in the computer graphics community. In particular, homemade 3D scanners are now within reach of students and hobbyists with a modest budget.

This course provides a beginner with the necessary mathematics, software, and practical details to leverage projector-camera systems in their own 3D scanning projects. An example-driven approach is used throughout; each new concept is illustrated using a practical scanner implemented with off-the-shelf parts. A minimal post-processing pipeline is presented for merging multiple scans to produce watertight meshes. The course concludes by detailing how these approaches are used in rapid prototyping, entertainment, cultural heritage, and web-based applications.

These course notes are organized into three primary sections, spanning theoretical concepts, practical construction details, and algorithms for constructing high-quality 3D models. Chapters 1 and 2 survey the field and present the unifying concept of triangulation. Chapters 3–5 document the construction of projector-camera systems, swept-plane scanners, and structured lighting, respectively. The post-processing pipeline and recent advances are covered in Chapters 6–7. We encourage attendees to email the authors with questions or links to their own 3D scanning projects that draw on the course material. Revised course notes, updated software, recent publications, and similar do-it-yourself projects are maintained on the course website at `http://mesh.brown.edu/dlanman/scan3d`.

1.1 3D Scanning Technology

Metrology is an ancient and diverse field, bridging the gap between mathematics and engineering. Efforts at measurement standardization were first undertaken by the Indus Valley Civilization as early as 2600–1900 BCE. Even with only crude units, such as the length of human appendages, the development of geometry revolutionized the ability to measure distance accurately. Around 240 BCE, Eratosthenes estimated the circumference of the Earth from knowledge of the elevation angle of the Sun during the summer solstice in Alexandria and Syene. Mathematics and standardization efforts continued to mature through the Renaissance (1300–1600 CE) and into the Scientific Revolution (1550–1700 CE). However, it was the Industrial Revolution (1750–1850 CE) which drove metrology to the forefront. As automatized methods of mass production became commonplace, advanced measurement technologies ensured interchangeable parts were just that–accurate copies of the original.

Through these historical developments, measurement tools varied with mathematical knowledge and practical needs. Early methods required direct contact with a surface (e.g., callipers and rulers). The pantograph, invented in 1603 by Christoph Scheiner, uses a special mechanical linkage so movement of a stylus (in contact with the surface) can be precisely duplicated by a drawing pen. The modern coordinate measuring machine (CMM) functions in much the same manner, recording the displacement of a probe tip as it slides across a solid surface (see Figure 1.1). While effective, such contact-based methods can harm fragile objects and require long periods of time to build an accurate 3D model. Non-contact scanners address these limitations by observing, and possibly controlling, the interaction of light with the object.

1.1.1 Passive Methods

Non-contact optical scanners can be categorized by the degree to which controlled illumination is required. Passive scanners do not require direct control of any illumination source, instead relying entirely on ambient light. Stereoscopic imaging is one of the most widely used passive 3D imaging systems, both in biology and engineering. Mirroring the human visual system, stereoscopy estimates the position of a 3D scene point by triangulation [LN04]; first, the 2D projection of a given point is identified in each camera. Using known calibration objects, the imaging properties of each camera are estimated, ultimately allowing a single 3D line to be

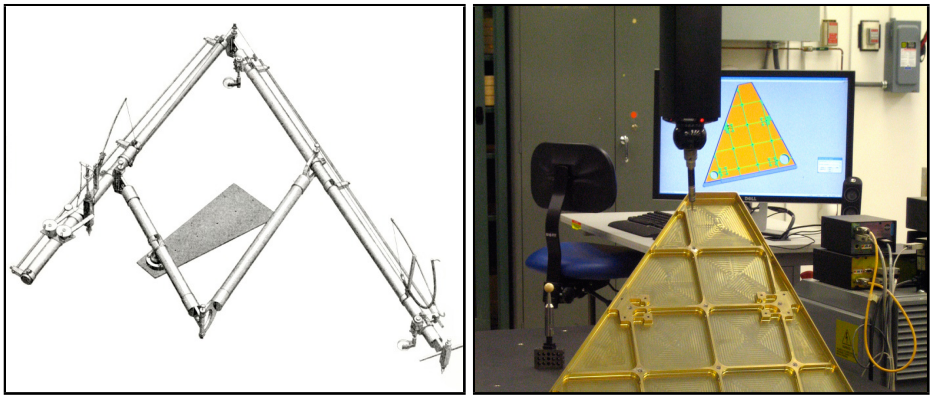


Figure 1.1: Contact-based shape measurement. (Left) A sketch of Sorenson's engraving pantograph patented in 1867. (Right) A modern coordinate measuring machining (from Flickr user hyperbolation). In both devices, deflection of a probe tip is used to estimate object shape, either for transferring engravings or for recovering 3D models, respectively.

drawn from each camera's center of projection through the 3D point. The intersection of these two lines is then used to recover the depth of the point.

Trinocular [VF92] and multi-view stereo [HZ04] systems have been introduced to improve the accuracy and reliability of conventional stereoscopic systems. However, all such passive triangulation methods require *correspondences* to be found among the various viewpoints. Even for stereo vision, the development of matching algorithms remains an open and challenging problem in the field [SCD*06]. Today, real-time stereoscopic and multi-view systems are emerging, however certain challenges continue to limit their widespread adoption [MPL04]. Foremost, flat or periodic textures prevent robust matching. While machine learning methods and prior knowledge are being advanced to solve such problems, multi-view 3D scanning remains somewhat outside the domain of hobbyists primarily concerned with accurate, reliable 3D measurement.

Many alternative passive methods have been proposed to sidestep the correspondence problem, often times relying on more robust computer vision algorithms. Under controlled conditions, such as a known or constant background, the external boundaries of foreground objects can be reliably identified. As a result, numerous shape-from-silhouette algorithms have emerged. Laurentini [Lau94] considers the case of a finite number of cameras observing a scene. The visual hull is defined as the union of the gener-

alized viewing cones defined by each camera's center of projection and the detected silhouette boundaries. Recently, free-viewpoint video [CTMS03] systems have applied this algorithm to allow dynamic adjustment of viewpoint [MBR*00, SH03]. Cipolla and Giblin [CG00] consider a differential formulation of the problem, reconstructing depth by observing the visual motion of occluding contours (such as silhouettes) as a camera is perturbed.

Optical imaging systems require a sufficiently large aperture so that enough light is gathered during the available exposure time [Hec01]. Correspondingly, the captured imagery will demonstrate a limited depth of field; only objects close to the plane of focus will appear in sharp contrast, with distant objects blurred together. This effect can be exploited to recover depth, by increasing the aperture diameter to further reduce the depth of field. Nayar and Nakagawa [NN94] estimate shape-from-focus, collecting a focal stack by translating a single element (either the lens, sensor, or object). A focus measure operator [WN98] is then used to identify the plane of best focus, and its corresponding distance from the camera.

Other passive imaging systems further exploit the depth of field by modifying the shape of the aperture. Such modifications are performed so that the point spread function (PSF) becomes invertible and strongly depth-dependent. Levin et al. [LFDF07] and Farid [Far97] use such coded apertures to estimate intensity and depth from defocused images. Greengard et al. [GSP06] modify the aperture to produce a PSF whose rotation is a function of scene depth. In a similar vein, shadow moiré is produced by placing a high-frequency grating between the scene and the camera. The resulting interference patterns exhibit a series of depth-dependent fringes.

While the preceding discussion focused on optical modifications for 3D reconstruction from 2D images, numerous model-based approaches have also emerged. When shape is known *a priori*, then coarse image measurements can be used to infer object translation, rotation, and deformation. Such methods have been applied to human motion tracking [KM00, OSS*00, dAST*08], vehicle recognition [Sul95, FWM98], and human-computer interaction [RWLB01]. Additionally, user-assisted model construction has been demonstrated using manual labeling of geometric primitives [Deb97].

1.1.2 Active Methods

Active optical scanners overcome the correspondence problem using controlled illumination. In comparison to non-contact and passive methods, active illumination is often more sensitive to surface material properties. Strongly reflective or translucent objects often violate assumptions made

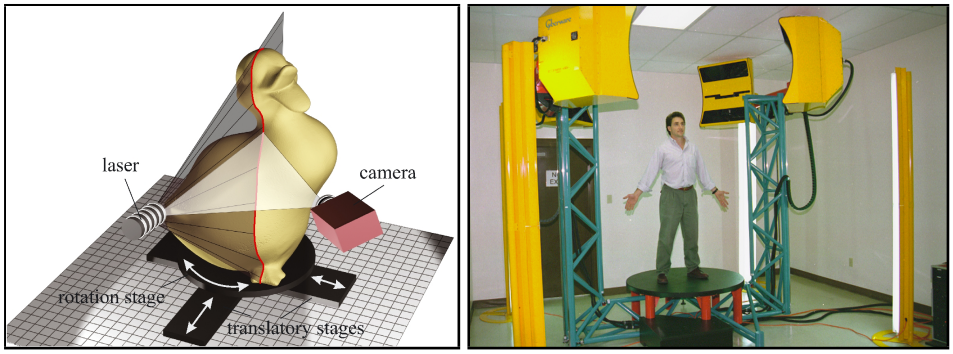


Figure 1.2: Active methods for 3D scanning. (Left) Conceptual diagram of a 3D slit scanner, consisting of a mechanically translated laser stripe. (Right) A Cyberware scanner, applying laser striping for whole body scanning (from Flickr user NIOSH).

by active optical scanners, requiring additional measures to acquire such problematic subjects. For a detailed history of active methods, we refer the reader to the survey article by Blais [Bla04]. In this section we discuss some key milestones along the way to the scanners we consider in this course.

Many active systems attempt to solve the correspondence problem by replacing one of the cameras, in a passive stereoscopic system, with a controllable illumination source. During the 1970s, single-point laser scanning emerged. In this scheme, a series of fixed and rotating mirrors are used to raster scan a single laser spot across a surface. A digital camera records the motion of this "flying spot". The 2D projection of the spot defines, with appropriate calibration knowledge, a line connecting the spot and the camera's center of projection. The depth is recovered by intersecting this line with the line passing from the laser source to the spot, given by the known deflection of the mirrors. As a result, such single-point scanners can be seen as the optical equivalent of coordinate measuring machines.

As with CMMs, single-point scanning is a painstakingly slow process. With the development of low-cost, high-quality CCD arrays in the 1980s, slit scanners emerged as a powerful alternative. In this design, a laser projector creates a single planar sheet of light. This "slit" is then mechanically-swept across the surface. As before, the known deflection of the laser source defines a 3D plane. The depth is recovered by the intersection of this plane with the set of lines passing through the 3D stripe on the surface and the camera's center of projection.

Effectively removing one dimension of the raster scan, slit scanners remain a popular solution for rapid shape acquisition. A variety of commercial products use swept-plane laser scanning, including the Polhemus FastSCAN [Pol], the NextEngine [Nex], the SLP 3D laser scanning probes from Laser Design [Las], and the HandyScan line of products [Cre]. While effective, slit scanners remain difficult to use if moving objects are present in the scene. In addition, because of the necessary separation between the light source and camera, certain occluded regions cannot be reconstructed. This limitation, while shared by many 3D scanners, requires multiple scans to be merged—further increasing the data acquisition time.

A digital "structured light" projector can be used to eliminate the mechanical motion required to translate the laser stripe across the surface. Naïvely, the projector could be used to display a single column (or row) of white pixels translating against a black background to replicate the performance of a slit scanner. However, a simple swept-plane sequence does not fully exploit the projector, which is typically capable of displaying arbitrary 24-bit color images. Structured lighting sequences have been developed which allow the projector-camera correspondences to be assigned in relatively few frames. In general, the identity of each plane can be encoded spatially (i.e., within a single frame) or temporally (i.e., across multiple frames), or with a combination of both spatial and temporal encodings. There are benefits and drawbacks to each strategy. For instance, purely spatial encodings allow a single static pattern to be used for reconstruction, enabling dynamic scenes to be captured. Alternatively, purely temporal encodings are more likely to benefit from redundancy, reducing reconstruction artifacts. We refer the reader to a comprehensive assessment of such codes by Salvi et al. [SPB04].

Both slit scanners and structured lighting are ill-suited for scanning dynamic scenes. In addition, due to separation of the light source and camera, certain occluded regions will not be recovered. In contrast, time-of-flight rangefinders estimate the distance to a surface from a single center of projection. These devices exploit the finite speed of light. A single pulse of light is emitted. The elapsed time, between emitting and receiving a pulse, is used to recover the object distance (since the speed of light is known). Several economical time-of-flight depth cameras are now commercially available, including Canesta's CANESTAVISION [HARN06] and 3DV's Z-Cam [IY01]. However, the depth resolution and accuracy of such systems (for static scenes) remain below that of slit scanners and structured lighting.

Active imaging is a broad field; a wide variety of additional schemes

have been proposed, typically trading system complexity for shape accuracy. As with model-based approaches in passive imaging, several active systems achieve robust reconstruction by making certain simplifying assumptions about the topological and optical properties of the surface. Woodham [Woo89] introduces photometric stereo, allowing smooth surfaces to be recovered by observing their shading under at least three (spatially disparate) point light sources. Hernández et al. [HVB*07] further demonstrate a real-time photometric stereo system using three colored light sources. Similarly, the complex digital projector required for structured lighting can be replaced by one or more printed gratings placed next to the projector and camera. Like shadow moiré, such projection moiré systems create depth-dependent fringes. However, certain ambiguities remain in the reconstruction unless the surface is assumed to be smooth.

Active and passive 3D scanning methods continue to evolve, with recent progress reported annually at various computer graphics and vision conferences, including 3-D Digital Imaging and Modeling (3DIM), SIGGRAPH, Eurographics, CVPR, ECCV, and ICCV. Similar advances are also published in the applied optics communities, typically through various SPIE and OSA journals. We will survey several promising recent works in Chapter 7.

1.2 Concepts and Scanners in this Course

This course is grounded in the unifying concept of triangulation. At their core, stereoscopic imaging, slit scanning, and structured lighting all attempt to recover the shape of 3D objects in the same manner. First, the correspondence problem is solved, either by a passive matching algorithm or by an active "space-labeling" approach (e.g., projecting known lines, planes, or other patterns). After establishing correspondences across two or more views (e.g., between a pair of cameras or a single projector-camera pair), triangulation recovers the scene depth. In stereoscopic and multi-view systems, a point is reconstructed by intersecting two or more corresponding lines. In slit scanning and structured lighting systems, a point is recovered by intersecting corresponding lines and planes.

To elucidate the principles of such triangulation-based scanners, this course describes how to construct classic slit scanners, as well as a structured lighting system. As shown in Figure 1.3, our slit scanner is inspired by the work of Bouguet and Perona [BP]. In this design, a wooden stick and halogen lamp replicate the function of a manually-translated laser stripe

Figure 1.3: 3D photography using planar shadows. From left to right: the capture setup, a single image from the scanning sequence, and a reconstructed object (rendered as a colored point cloud).

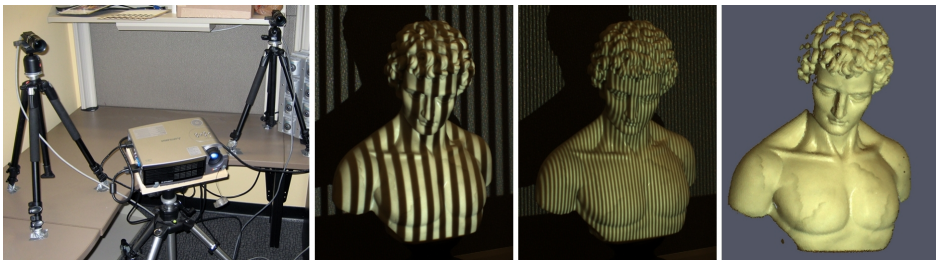


Figure 1.4: Structured light for 3D scanning. From left to right: a structured light scanning system containing a pair of digital cameras and a single projector, two images of an object illuminated by different bit planes of a Gray code structured light sequence, and a reconstructed 3D point cloud.

projector, allowing shadow planes to be swept through the scene. The details of its construction are presented in Chapter 4. As shown in Figure 1.4, our structured lighting system contains a single projector and one or more digital cameras. In Chapter 5, we describe its construction and examine several temporally-encoded illumination sequences.

By providing example data sets, open source software, and detailed implementation notes, we hope to enable beginners and hobbyists to replicate our results. We believe the process of building your own 3D scanner is enjoyable and instructive. Along the way, you'll likely learn a great deal about the practical use of projector-camera systems, hopefully in a manner that supports your own research. To that end, we conclude in Chapter 7 by discussing some of the projects that emerged when this course was previously taught at Brown University in 2007 and 2009. We will continue to update these notes and the website with links to any do-it-yourself scanners or research projects undertaken by course attendees.

Chapter 2

The Mathematics of Triangulation

This course is primarily concerned with the estimation of 3D shape by illuminating the world with certain known patterns, and observing the illuminated objects with cameras. In this chapter we derive models describing this image formation process, leading to the development of reconstruction equations allowing the recovery of 3D shape by geometric triangulation.

We start by introducing the basic concepts in a coordinate-free fashion, using elementary algebra and the language of analytic geometry (e.g., points, vectors, lines, rays, and planes). Coordinates are introduced later, along with relative coordinate systems, to quantify the process of image formation in cameras and projectors.

2.1 Perspective Projection and the Pinhole Model

A simple and popular geometric model for a camera or a projector is the *pinhole model*, composed of a plane and a point external to the plane (see Figure 2.1). We refer to the plane as the *image plane*, and to the point as the *center of projection*. In a camera, every 3D point (other than the center of projection) determines a unique line passing through the center of projection. If this line is not parallel to the image plane, then it must intersect the image plane in a single *image point*. In mathematics, this mapping from 3D points to 2D image points is referred to as a *perspective projection*. Except for the fact that light traverses this line in the opposite direction, the geometry of a projector can be described with the same model. That is, given a 2D image point in the projector's image plane, there must exist a unique line containing this point and the center of projection (since the center of projection cannot belong to the image plane). In summary, light travels away

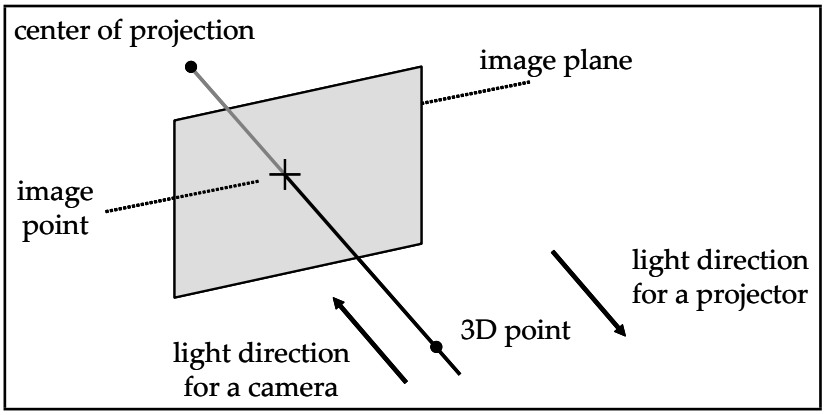


Figure 2.1: Perspective projection under the pinhole model.

from a projector (or towards a camera) along the line connecting the 3D scene point with its 2D perspective projection onto the image plane.

2.2 Geometric Representations

Since light moves along straight lines (in a homogeneous medium such as air), we derive 3D reconstruction equations from geometric constructions involving the intersection of lines and planes, or the approximate intersection of pairs of lines (two lines in 3D may not intersect). Our derivations only draw upon elementary algebra and analytic geometry in 3D (e.g., we operate on points, vectors, lines, rays, and planes). We use lower case letters to denote points p and vectors v. All the vectors will be taken as column vectors with real-valued coordinates $v \in \mathbb{R}^3$, which we can also regard as matrices with three rows and one column $v \in \mathbb{R}^{3\times 1}$. The length of a vector v is a scalar $\|v\| \in \mathbb{R}$. We use matrix multiplication notation for the inner product $v_1^t v_2 \in \mathbb{R}$ of two vectors v_1 and v_2, which is also a scalar. Here $v_1^t \in \mathbb{R}^{1\times 3}$ is a row vector, or a 1×3 matrix, resulting from transposing the column vector v_1. The value of the inner product of the two vectors v_1 and v_2 is equal to $\|v_1\|\|v_2\| \cos(\alpha)$, where α is the angle formed by the two vectors ($0 \leq \alpha \leq 180°$). The $3 \times N$ matrix resulting from concatenating N vectors $v_1, \ldots, v_N$ as columns is denoted $[v_1|\cdots|v_N] \in \mathbb{R}^{3\times N}$. The vector product $v_1 \times v_2 \in \mathbb{R}^3$ of the two vectors v_1 and v_2 is a vector perpendicular to both v_1 and v_2, of length $\|v_1 \times v_2\| = \|v_1\|\,\|v_2\|\,\sin(\alpha)$, and direction determined by the right hand rule (i.e., such that the determinant of the

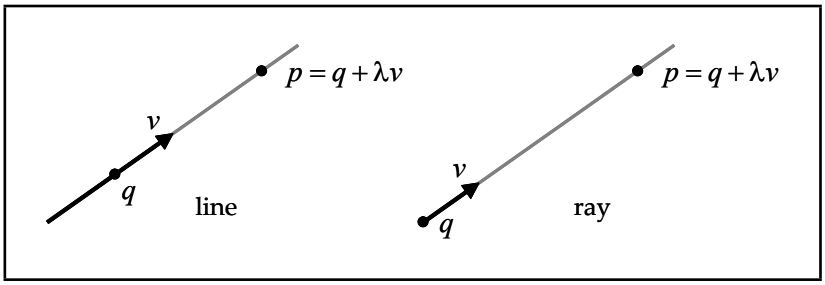


Figure 2.2: Parametric representation of lines and rays.

matrix $[v_1|v_2|v_1 \times v_2]$ is non-negative). In particular, two vectors v_1 and v_2 are linearly dependent (i.e., one is a scalar multiple of the other), if and only if the vector product $v_1 \times v_2$ is equal to zero.

2.2.1 Points and Vectors

Since vectors form a vector space, they can be multiplied by scalars and added to each other. Points, on the other hand, do not form a vector space. But vectors and points are related: a point plus a vector $p + v$ is another point, and the difference between two points $q - p$ is a vector. If p is a point, λ is a scalar, and v is a vector, then $q = p + \lambda v$ is another point. In this expression, λv is a vector of length $|\lambda| \, \|v\|$. Multiplying a point by a scalar λp is not defined, but an *affine* combination of N points $\lambda_1 p_1 + \cdots + \lambda_N p_N$, with $\lambda_1 + \cdots + \lambda_N = 1$, is well defined:

$$\lambda_1 p_1 + \cdots + \lambda_N p_N = p_1 + \lambda_2 (p_2 - p_1) + \cdots + \lambda_N (p_N - p_1) \; .$$

2.2.2 Parametric Representation of Lines and Rays

A line L can be described by specifying one of its points q and a direction vector v (see Figure 2.2). Any other point p on the line L can be described as the result of adding a scalar multiple λv, of the direction vector v, to the point q (λ can be positive, negative, or zero):

$$L = \{p = q + \lambda v \; : \; \lambda \in \mathbb{R}\} \; . \tag{2.1}$$

This is the *parametric* representation of a line, where the scalar λ is the parameter. Note that this representation is not unique, since q can be replaced by any other point on the line L, and v can be replaced by any non-zero

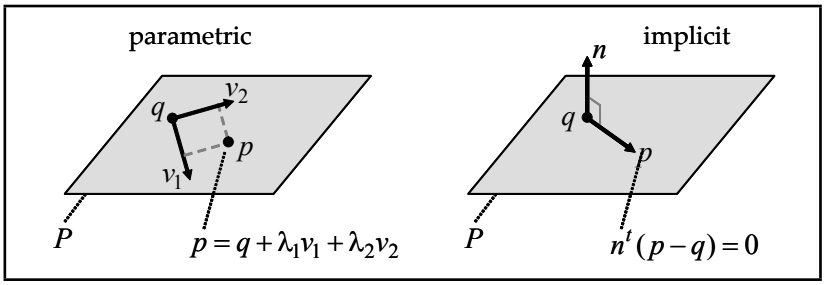


Figure 2.3: Parametric and implicit representations of planes.

scalar multiple of v. However, for each choice of q and v, the correspondence between parameters $\lambda \in \mathbb{R}$ and points p on the line L is one-to-one.

A ray is *half* of a line. While in a line the parameter λ can take any value, in a ray it is only allowed to take non-negative values.

$$R = \{p = q + \lambda v \ : \ \lambda \geq 0\}$$

In this case, if the point q is changed, a different ray results. Since it is unique, the point q is called the *origin* of the ray. The direction vector v can be replaced by any *positive* scalar multiple, but not by a negative scalar multiple. Replacing the direction vector v by a negative scalar multiple results in the *opposite* ray. By convention in projectors, light traverses rays along the direction determined by the direction vector. Conversely in cameras, light traverses rays in the direction opposite to the direction vector (i.e., in the direction of decreasing λ).

2.2.3 Parametric Representation of Planes

Similar to how lines are represented in parametric form, a plane P can be described in *parametric* form by specifying one of its points q and two linearly independent direction vectors v_1 and v_2 (see Figure 2.3). Any other point p on the plane P can be described as the result of adding scalar multiples $\lambda_1 v_1$ and $\lambda_2 v_2$ of the two vectors to the point q, as follows.

$$P = \{p = q + \lambda_1 v_1 + \lambda_2 v_2 \ : \ \lambda_1, \lambda_2 \in \mathbb{R}\}$$

As in the case of lines, this representation is not unique. The point q can be replaced by any other point in the plane, and the vectors v_1 and v_2 can be replaced by any other two linearly independent linear combinations of v_1 and v_2.

2.2.4 Implicit Representation of Planes

A plane P can also be described in *implicit* form as the set of zeros of a linear equation in three variables. Geometrically, the plane can be described by one of its points q and a normal vector n. A point p belongs to the plane P if and only if the vectors $p - q$ and n are orthogonal, such that

$$P = \{p \ : \ n^t(p - q) = 0\} \ . \tag{2.2}$$

Again, this representation is not unique. The point q can be replaced by any other point in the plane, and the normal vector n by any non-zero scalar multiple λn.

To convert from the parametric to the implicit representation, we can take the normal vector $n = v_1 \times v_2$ as the vector product of the two basis vectors v_1 and v_2. To convert from implicit to parametric, we need to find two linearly independent vectors v_1 and v_2 orthogonal to the normal vector n. In fact, it is sufficient to find one vector v_1 orthogonal to n. The second vector can be defined as $v_2 = n \times v_1$. In both cases, the same point q from one representation can be used in the other.

2.2.5 Implicit Representation of Lines

A line L can also be described in *implicit* form as the intersection of two planes, both represented in implicit form, such that

$$L = \{p \ : \ n_1^t(p - q) = n_2^t(p - q) = 0\}, \tag{2.3}$$

where the two normal vectors n_1 and n_2 are linearly independent (if n_1 an n_2 are linearly dependent, rather than a line, the two equations describe the same plane). Note that when n_1 and n_2 are linearly independent, the two implicit representations for the planes can be defined with respect to a common point belonging to both planes, rather than to two different points. Since a line can be described as the intersection of many different pairs of planes, this representation is not unique. The point q can be replaced by any other point belonging to the intersection of the two planes, and the two normal vectors can be replaced by any other pair of linearly independent linear combinations of the two vectors.

To convert from the parametric representation of Equation 2.1 to the implicit representation of Equation 2.3, one needs to find two linearly independent vectors n_1 and n_2 orthogonal to the direction vector v. One way to do so is to first find one non-zero vector n_1 orthogonal to v, and then

take n_2 as the vector product $n_2 = v \times n_1$ of v and n_1. To convert from implicit to parametric, one needs to find a non-zero vector v orthogonal to both normal vectors n_1 and n_2. The vector product $v = n_1 \times n_2$ is one such vector, and any other is a scalar multiple of it.

2.3 Reconstruction by Triangulation

As will be discussed in Chapters 4 and 5, it is common for projected illumination patterns to contain identifiable lines or points. Under the pinhole projector model, a projected line creates a plane of light (the unique plane containing the line on the image plane and the center of projection), and a projected point creates a ray of light (the unique line containing the image point and the center of projection).

While the intersection of a ray of light with the object being scanned can be considered as a single illuminated point, the intersection of a plane of light with the object generally contains many illuminated curved segments (see Figure 1.2). Each of these segments is composed of many illuminated points. A single illuminated point, visible to the camera, defines a camera ray. For now, we assume that the locations and orientations of projector and camera are known with respect to the global coordinate system (with procedures for estimating these quantities covered in Chapter 3). Under this assumption, the equations of projected planes and rays, as well as the equations of camera rays corresponding to illuminated points, are defined by parameters which can be measured. From these measurements, the location of illuminated points can be recovered by intersecting the planes or rays of light with the camera rays corresponding to the illuminated points. Through such procedures the depth ambiguity introduced by pinhole projection can be eliminated, allowing recovery of a 3D surface model.

2.3.1 Line-Plane Intersection

Computing the intersection of a line and a plane is straightforward when the line is represented in parametric form

$$L = \{p = q_L + \lambda v \ : \ \lambda \in \mathbb{R}\},$$

and the plane is represented in implicit form

$$P = \{p \ : \ n^t(p - q_P) = 0\} \ .$$

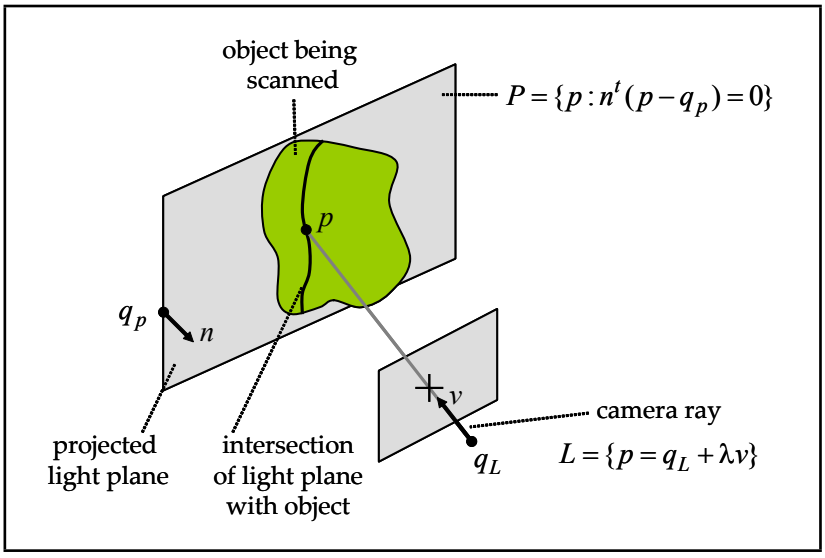


Figure 2.4: Triangulation by line-plane intersection.

Note that the line and the plane may not intersect, in which case we say that the line and the plane are *parallel*. This is the case if the vectors v and n are orthogonal $n^t v = 0$. The vectors v and n are also orthogonal when the line L is contained in the plane P. Whether or not the point q_L belongs to the plane P differentiates one case from the other. If the vectors v and n are not orthogonal $n^t v \neq 0$, then the intersection of the line and the plane contains exactly one point p. Since this point belongs to the line, it can be written as $p = q_L + \lambda v$, for a value λ which we need to determine. Since the point also belongs to the plane, the value λ must satisfy the linear equation

$$n^t(p - q_p) = n^t(\lambda v + q_L - q_p) = 0 \ ,$$

or equivalently

$$\lambda = \frac{n^t(q_P - q_L)}{n^t v} \ . \tag{2.4}$$

Since we have assumed that the line and the plane are not parallel (i.e., by checking that $n^t v \neq 0$ beforehand), this expression is well defined. A geometric interpretation of line-plane intersection is provided in Figure 2.4.

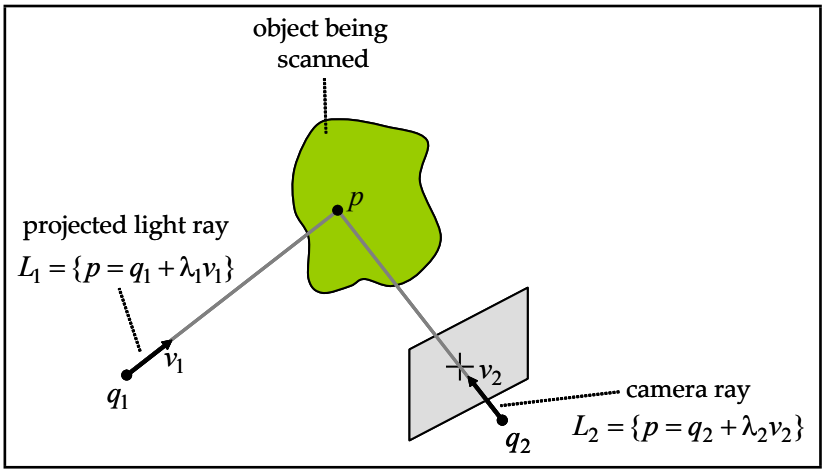


Figure 2.5: Triangulation by line-line intersection.

2.3.2 Line-Line Intersection

We consider here the intersection of two arbitrary lines L_1 and L_2, as shown in Figure 2.5.

$$L_1 = \{p = q_1 + \lambda_1 v_1 \ : \ \lambda_1 \in \mathbb{R}\} \quad \text{and} \quad L_2 = \{p = q_2 + \lambda_2 v_2 \ : \ \lambda_2 \in \mathbb{R}\}$$

Let us first identify the special cases. The vectors v_1 and v_2 can be linearly dependent (i.e., if one is a scalar multiple of the other) or linearly independent.

The two lines are parallel if the vectors v_1 and v_2 are linearly dependent. If, in addition, the vector $q_2 - q_1$ can also be written as a scalar multiple of v_1 or v_2, then the lines are identical. Of course, if the lines are parallel but not identical, they do not intersect.

If v_1 and v_2 are linearly independent, the two lines may or may not intersect. If the two lines intersect, the intersection contains a single point. The necessary and sufficient condition for two lines to intersect, when v_1 and v_2 are linearly independent, is that scalar values λ_1 and λ_2 exist so that

$$q_1 + \lambda_1 v_1 = q_2 + \lambda_2 v_2,$$

or equivalently so that the vector $q_2 - q_1$ is linearly dependent on v_1 and v_2.

Since two lines may not intersect, we define the *approximate intersection* as the point which is *closest* to the two lines. More precisely, whether two

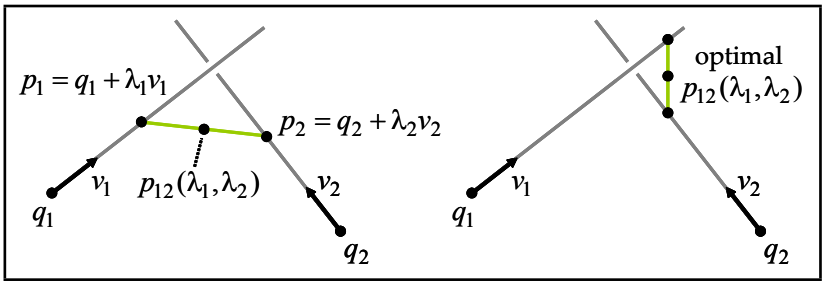


Figure 2.6: The midpoint $p_{12}(\lambda_1, \lambda_2)$ for arbitrary values (left) of λ_1, λ_2 and for the optimal values (right).

lines intersect or not, we define the approximate intersection as the point p which minimizes the sum of the square distances to both lines

$$\phi(p, \lambda_1, \lambda_2) = \|q_1 + \lambda_1 v_1 - p\|^2 + \|q_2 + \lambda_2 v_2 - p\|^2 \ .$$

As before, we assume v_1 and v_2 are linearly independent, such the approximate intersection is a unique point.

To prove that the previous statement is true, and to determine the value of p, we follow an algebraic approach. The function $\phi(p, \lambda_1, \lambda_2)$ is a quadratic non-negative definite function of five variables, the three coordinates of the point p and the two scalars λ_1 and λ_2.

We first reduce the problem to the minimization of a different quadratic non-negative definite function of only two variables λ_1 and λ_2. Let $p_1 = q_1 + \lambda_1 v_1$ be a point on the line L_1, and let $p_2 = q_2 + \lambda_2 v_2$ be a point on the line L_2. Define the midpoint p_{12}, of the line segment joining $p1$ and $p2$, as

$$p_{12} = p_1 + \frac{1}{2}(p_2 - p_1) = p_2 + \frac{1}{2}(p_1 - p_2) \ .$$

A necessary condition for the minimizer $(p, \lambda_1, \lambda_2)$ of ϕ is that the partial derivatives of ϕ, with respect to the five variables, all vanish at the minimizer. In particular, the three derivatives with respect to the coordinates of the point p must vanish

$$\frac{\partial \phi}{\partial p} = (p - p_1) + (p - p_2) = 0 \ ,$$

or equivalently, it is necessary for the minimizer point p to be the midpoint p_{12} of the segment joining p_1 and p_2 (see Figure 2.6).

As a result, the problem reduces to the minimization of the square distance from a point p_1 on line L_1 to a point p_2 on line L_2. Practically, we

must now minimize the quadratic non-negative definite function of two variables

$$\psi(\lambda_1, \lambda_2) = 2\phi(p_{12}, \lambda_1, \lambda_2) = \|(q_2 + \lambda_2 v_2) - (q_1 + \lambda_1 v_1)\|^2 .$$

Note that it is still necessary for the two partial derivatives of ψ, with respect to λ_1 and λ_2, to be equal to zero at the minimum, as follows.

$$\frac{\partial \psi}{\partial \lambda_1} = v_1^t(\lambda_1 v_1 - \lambda_2 v_2 + q_1 - q_2) = \lambda_1 \|v_1\| - \lambda_2 v_1^t v_2 + v_1^t(q_1 - q_2) = 0$$

$$\frac{\partial \psi}{\partial \lambda_2} = v_2^t(\lambda_2 v_2 - \lambda_1 v_1 + q_2 - q_1) = \lambda_2 \|v_2\|^2 - \lambda_2 v_2^t v_1 + v_2^t(q_2 - q_1) = 0$$

These provide two linear equations in λ_1 and λ_2, which can be concisely expressed in matrix form as

$$\begin{pmatrix} \|v_1\|^2 & -v_1^t v_2 \\ -v_2^t v_1 & \|v_2\|^2 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} v_1^t(q_2 - q_1) \\ v_2^t(q_1 - q_2) \end{pmatrix} .$$

It follows from the linear independence of v_1 and v_2 that the 2×2 matrix on the left hand side is non-singular. As a result, the unique solution to the linear system is given by

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} \|v_1\|^2 & -v_1^t v_2 \\ -v_2^t v_1 & \|v_2\|^2 \end{pmatrix}^{-1} \begin{pmatrix} v_1^t(q_2 - q_1) \\ v_2^t(q_1 - q_2) \end{pmatrix}$$

or equivalently

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \frac{1}{\|v_1\|^2\|v_2\|^2 - (v_1^t v_2)^2} \begin{pmatrix} \|v_2\|^2 & v_1^t v_2 \\ v_2^t v_1 & \|v_1\|^2 \end{pmatrix} \begin{pmatrix} v_1^t(q_2 - q_1) \\ v_2^t(q_1 - q_2) \end{pmatrix} . \quad (2.5)$$

In conclusion, the approximate intersection p can be obtained from the value of either λ_1 or λ_2 provided by these expressions.

2.4 Coordinate Systems

So far we have presented a coordinate-free description of triangulation. In practice, however, image measurements are recorded in discrete pixel units. In this section we incorporate such coordinates into our prior equations, as well as document the various coordinate systems involved.

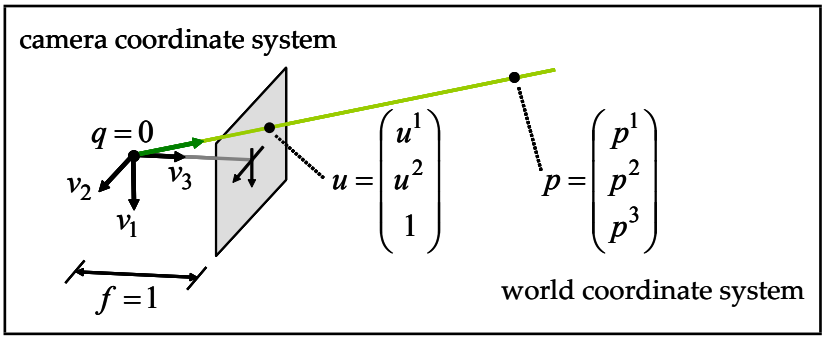


Figure 2.7: The ideal pinhole camera.

2.4.1 Image Coordinates and the Pinhole Camera

Consider a pinhole model with center of projection o and image plane $P = \{p = q + u^1 v_1 + u^2 v_2 : u^1, u^2 \in \mathbb{R}\}$. Any 3D point p, not necessarily on the image plane, has coordinates $(p^1, p^2, p^3)^t$ relative to the origin of the world coordinate system. On the image plane, the point q and vectors v_1 and v_2 define a local coordinate system. The image coordinates of a point $p = q + u^1 v_1 + u^2 v_2$ are the parameters u^1 and u^2, which can be written as a 3D vector $u = (u^1, u^2, 1)$. Using this notation point p is expressed as

$$\begin{pmatrix} p^1 \\ p^2 \\ p^3 \end{pmatrix} = [v_1|v_2|q] \begin{pmatrix} u^1 \\ u^2 \\ 1 \end{pmatrix} .$$

2.4.2 The Ideal Pinhole Camera

In the *ideal pinhole camera* shown in Figure 2.7, the center of projection o is at the origin of the world coordinate system, with coordinates $(0, 0, 0)^t$, and the point q and the vectors v_1 and v_2 are defined as

$$[v_1|v_2|q] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} .$$

Note that not every 3D point has a projection on the image plane. Points without a projection are contained in a plane parallel to the image passing through the center of projection. An arbitrary 3D point p with coordinates $(p^1, p^2, p^3)^t$ belongs to this plane if $p^3 = 0$, otherwise it projects onto an

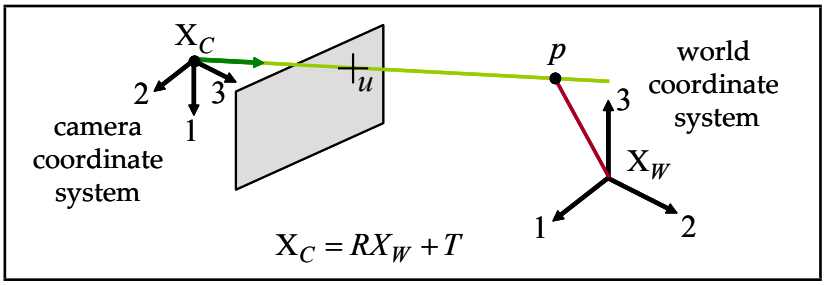


Figure 2.8: The general pinhole model.

image point with the following coordinates.

$$\begin{aligned} u^1 &= p^1/p^3 \\ u^2 &= p^2/p^3 \end{aligned}$$

There are other descriptions for the relation between the coordinates of a point and the image coordinates of its projection; for example, the projection of a 3D point p with coordinates $(p^1, p^2, p^3)^t$ has image coordinates $u = (u^1, u^2, 1)$ if, for some scalar $\lambda \neq 0$, we can write

$$\lambda \begin{pmatrix} u^1 \\ u^2 \\ 1 \end{pmatrix} = \begin{pmatrix} p^1 \\ p^2 \\ p^3 \end{pmatrix} . \tag{2.6}$$

2.4.3 The General Pinhole Camera

The center of a general pinhole camera is not necessarily placed at the origin of the world coordinate system and may be arbitrarily oriented. However, it does have a *camera coordinate system* attached to the camera, in addition to the *world coordinate system* (see Figure 2.8). A 3D point p has world coordinates described by the vector $p_W = (p^1_W, p^2_W, p^3_W)^t$ and camera coordinates described by the vector $p_C = (p^1_C, p^2_C, p^3_C)^t$. These two vectors are related by a rigid body transformation specified by a translation vector $T \in \mathbb{R}^3$ and a rotation matrix $R \in \mathbb{R}^{3\times3}$, such that

$$p_C = R\,p_W + T\ .$$

In camera coordinates, the relation between the 3D point coordinates and the 2D image coordinates of the projection is described by the ideal pinhole

camera projection (i.e., Equation 2.6), with $\lambda u = p_C$. In world coordinates this relation becomes

$$\lambda\, u = R\, p_W + T\ . \tag{2.7}$$

The parameters (R, T), which are referred to as the *extrinsic parameters* of the camera, describe the location and orientation of the camera with respect to the world coordinate system.

Equation 2.7 assumes that the unit of measurement of lengths on the image plane is the same as for world coordinates, that the distance from the center of projection to the image plane is equal to one unit of length, and that the origin of the image coordinate system has image coordinates $u^1 = 0$ and $u^2 = 0$. None of these assumptions hold in practice. For example, lengths on the image plane are measured in pixel units, and in meters or inches for world coordinates, the distance from the center of projection to the image plane can be arbitrary, and the origin of the image coordinates is usually on the upper left corner of the image. In addition, the image plane may be tilted with respect to the ideal image plane. To compensate for these limitations of the current model, a matrix $K \in \mathbb{R}^{3\times 3}$ is introduced in the projection equations to describe *intrinsic parameters* as follows.

$$\lambda\, u = K(R\, p_W + T) \tag{2.8}$$

The matrix K has the following form

$$K = \begin{pmatrix} f\, s_1 & f\, s_\theta & o^1 \\ 0 & f\, s_2 & o^2 \\ 0 & 0 & 1 \end{pmatrix}\ ,$$

where f is the *focal length* (i.e., the distance between the center of projection and the image plane). The parameters s_1 and s_2 are the first and second coordinate scale parameters, respectively. Note that such scale parameters are required since some cameras have non-square pixels. The parameter s_θ is used to compensate for a tilted image plane. Finally, $(o^1, o^2)^t$ are the image coordinates of the intersection of the vertical line in camera coordinates with the image plane. This point is called the *image center* or *principal point*. Note that all intrinsic parameters embodied in K are independent of the camera pose. They describe physical properties related to the mechanical and optical design of the camera. Since in general they do not change, the matrix K can be estimated once through a calibration procedure and stored (as will be described in the following chapter). Afterwards, image plane measurements in pixel units can immediately be "normalized", by

multiplying the measured image coordinate vector by K^{-1}, so that the relation between a 3D point in world coordinates and 2D image coordinates is described by Equation 2.7.

Real cameras also display non-linear lens distortion, which is also considered intrinsic. Lens distortion compensation must be performed prior to the normalization described above. We will discuss appropriate lens distortion models in Chapter 3.

2.4.4 Lines from Image Points

As shown in Figure 2.9, an image point with coordinates $u = (u^1, u^2, 1)^t$ defines a unique line containing this point and the center of projection. The challenge is to find the parametric equation of this line, as $L = \{p = q + \lambda\, v : \lambda \in \mathbb{R}\}$. Since this line must contain the center of projection, the projection of all the points it spans must have the same image coordinates. If p_W is the vector of world coordinates for a point contained in this line, then world coordinates and image coordinates are related by Equation 2.7 such that $\lambda\, u = R\, p_W + T$. Since R is a rotation matrix, we have $R^{-1} = R^t$ and we can rewrite the projection equation as

$$p_W = (-R^t T) + \lambda\, (R^t u)\ .$$

In conclusion, the line we are looking for is described by the point q with world coordinates $q_W = -R^t T$, which is the center of projection, and the vector v with world coordinates $v_W = R^t u$.

2.4.5 Planes from Image Lines

A straight line on the image plane can be described in either parametric or implicit form, both expressed in image coordinates. Let us first consider the implicit case. A line on the image plane is described by one implicit equation of the image coordinates

$$L = \{u : l^t u = l^1 u^1 + l^2 u^2 + l^3 = 0\}\ ,$$

where $l = (l^1, l^2, l^3)^t$ is a vector with $l^1 \neq 0$ or $l^2 \neq 0$. Using active illumination, projector patterns containing vertical and horizontal lines are common. Thus, the implicit equation of an horizontal line is

$$L_H = \{u : l^t u = u^2 - \nu = 0\}\ ,$$

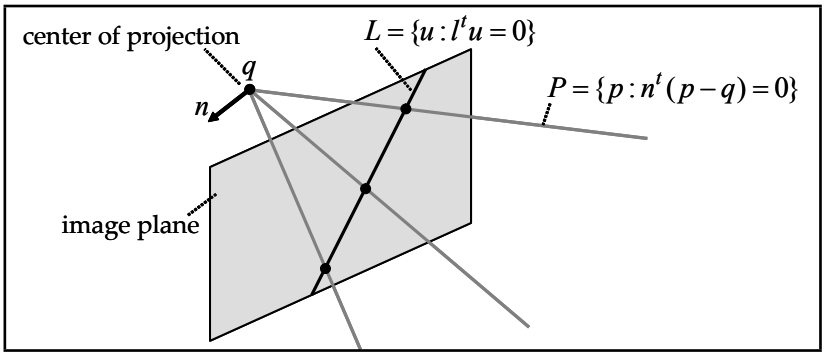


Figure 2.9: The plane defined by an image line and the center of projection.

where ν is the second coordinate of a point on the line. In this case we can take $l = (0, 1, -\nu)^t$. Similarly, the implicit equation of a vertical line is

$$L_V = \{u : l^t u = u^1 - \nu = 0\} ,$$

where ν is now the first coordinate of a point on the line. In this case we can take $l = (1, 0, -\nu)^t$. There is a unique plane P containing this line L and the center of projection. For each image point with image coordinates u on the line L, the line containing this point and the center of projection is contained in P. Let p be a point on the plane P with world coordinates p_W projecting onto an image point with image coordinates u. Since these two vectors of coordinates satisfy Equation 2.7, for which $\lambda\, u = R\, p_W + T$, and the vector u satisfies the implicit equation defining the line L, we have

$$0 = \lambda l^t u = l^t (R\, p_W + T) = (R^t l)^t \, (p_W - (-R^t T)) .$$

In conclusion, the implicit representation of plane P, corresponding to Equation 2.2 for which $P = \{p : n^t(p - q) = 0\}$, can be obtained with n being the vector with world coordinates $n_W = R^t l$ and q the point with world coordinates $q_W = -R^t T$, which is the center of projection.

Chapter 3

Camera and Projector Calibration

Triangulation is a deceptively simple concept, simply involving the pairwise intersection of 3D lines and planes. Practically, however, one must carefully calibrate the various cameras and projectors so the equations of these geometric primitives can be recovered from image measurements. In this chapter we lead the reader through the construction and calibration of a basic projector-camera system. Through this example, we examine how freely-available calibration packages, emerging from the computer vision community, can be leveraged in your own projects. While touching on the basic concepts of the underlying algorithms, our primarily goal is to help beginners overcome the "calibration hurdle".

In Section 3.1 we describe how to select, control, and calibrate a digital camera suitable for 3D scanning. The general pinhole camera model presented in Chapter 2 is extended to address lens distortion. A simple calibration procedure using printed checkerboard patterns is presented, following the established method of Zhang [Zha00]. Typical calibration results, obtained for the cameras used in Chapters 4 and 5, are provided as a reference.

While well-documented, freely-available camera calibration tools have emerged in recent years, community tools for projector calibration have received significantly less attention. In Section 3.2, we describe custom projector calibration software developed for this course. A simple procedure is used, wherein a calibrated camera observes a planar object with both printed and projected checkerboards on its surface. Considering the projector as an inverse camera, we describe how to estimate the various parameters of the projection model from such imagery. We conclude by reviewing calibration results for the structured light projector used in Chapter 5.

3.1 Camera Calibration

In this section we describe both the theory and practice of camera calibration. We begin by briefly considering what cameras are best suiting for building your own 3D scanner. We then present the widely-used calibration method originally proposed by Zhang [Zha00]. Finally, we provide step-by-step directions on how to use a freely-available MATLAB-based implementation of Zhang's method.

3.1.1 Camera Selection and Interfaces

Selection of the "best" camera depends on your budget, project goals, and preferred development environment. For instance, the two scanners described in this course place different restrictions on the imaging system. The swept-plane scanner in Chapter 4 requires a video camera, although a simple camcorder or webcam would be sufficient. In contrast, the structured lighting system in Chapter 5 can be implemented using a still camera. However, the camera must allow computer control of the shutter so image capture can be synchronized with image projection. In both cases, the range of cameras are further restricted to those that are supported by your development environment.

At the time of writing, the accompanying software for this course was primarily written in MATLAB. If readers wish to collect their own data sets using our software, we recommend obtaining a camera supported by the Image Acquisition Toolbox for MATLAB [Mat]. Note that this toolbox supports products from a variety of vendors, as well as any DCAM-compatible FireWire camera or webcam with a Windows Driver Model (WDM) or Video for Windows (VFW) driver. For FireWire cameras the toolbox uses the CMU DCAM driver [CMU]. Alternatively, if you select a WDM or VFW camera, Microsoft DirectX 9.0 (or higher) must be installed.

If you do not have access to any camera meeting these constraints, we recommend either purchasing an inexpensive FireWire camera or a high-quality USB webcam. While most webcams provide compressed imagery, FireWire cameras typically allow access to raw images free of compression artifacts. For those on a tight budget, we recommend the Unibrain Fire-i (available for around $100 USD). Although more expensive, we also recommend cameras from Point Grey Research. The camera interface provided by this vendor is particularly useful if you plan on developing more advanced scanners than those presented here. As a point of reference, our scanners were built using a pair of Point Grey GRAS-20S4M/C Grasshop-

Figure 3.1: Recommended cameras for course projects. (Left) Unibrain Fire-i IEEE-1394a digital camera, capable of 640×480 YUV 4:2:2 capture at 15 fps. (Middle) Logitech QuickCam Orbit AF USB 2.0 webcam, capable of 1600×1200 image capture at 30 fps. (Right) Point Grey Grasshopper IEEE-1394b digital camera; frame rate and resolution vary by model.

per video cameras. Each camera can capture a 1600×1200 24-bit RGB image at up to 30 Hz [Poia].

Outside of MATLAB, a wide variety of camera interfaces are available. However, relatively few come with camera calibration software, and even fewer with support for projector calibration. One exception, however, is the OpenCV (Open Source Computer Vision) library [Opea]. OpenCV is written in C, with wrappers for C# and Python, and consists of optimized implementations of many core computer vision algorithms. Video capture and display functions support a wide variety of cameras under multiple operating systems, including Windows, Mac OS, and Linux. Note, however, that projector calibration is not currently supported in OpenCV.

3.1.2 Calibration Methods and Software

Camera Calibration Methods

Camera calibration requires estimating the parameters of the general pinhole model presented in Section 2.4.3. This includes the intrinsic parameters, being focal length, principal point, and the scale factors, as well as the extrinsic parameters, defined by the rotation matrix and translation vector mapping between the world and camera coordinate systems. In total, 11 parameters (5 intrinsic and 6 extrinsic) must be estimated from a calibration sequence. In practice, a lens distortion model must be estimated as well. We recommend the reader review [HZ04, MSKS05] for an in-depth description of camera models and calibration methods.

At a basic level, camera calibration required recording a sequence of images of a calibration object, composed of a unique set of distinguishable features with known 3D displacements. Thus, each image of the calibration object provides a set of 2D-to-3D correspondences, mapping image coordinates to scene points. Naïvely, one would simply need to optimize over the set of 11 camera model parameters so that the set of 2D-to-3D correspondences are correctly predicted (i.e., the projection of each known 3D model feature is close to its measured image coordinates).

Many methods have been proposed over the years to solve for the camera parameters given such correspondences. In particular, the factorized approach originally proposed Zhang [Zha00] is widely-adopted in most community-developed tools. In this method, a planar checkerboard pattern is observed in two or more orientations (see Figure 3.2). From this sequence, the intrinsic parameters can be separately solved. Afterwards, a single view of a checkerboard can be used to solve for the extrinsic parameters. Given the relative ease of printing 2D patterns, this method is commonly used in computer graphics and vision publications.

Recommended Software

A comprehensive list of calibration software is maintained by Bouguet on the toolbox website at `http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/links.html`. We recommend course attendees use the MATLAB toolbox. Otherwise, OpenCV replicates many of its functionalities, while supporting multiple platforms. Although calibrating a small number of cameras using these tools is straightforward, calibrating a large network of cameras is a relatively recent and challenging problem in the field. If your projects lead you in this direction, we suggest the Multi-Camera Self-Calibration toolbox [SMP05]. This software takes a unique approach to calibration; rather than using multiple views of a planar calibration object, a standard laser point is simply translated through the working volume. Correspondences between the cameras are automatically determined from the tracked projection of the laser pointer in each image. We encourage attendees to email us with their own preferred tools. We will maintain an up-to-date list on the course website. For the remainder of the course notes, we will use the MATLAB toolbox for camera calibration.

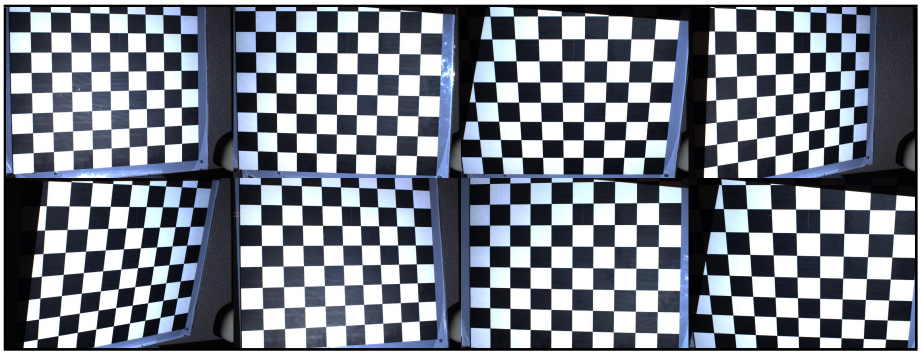


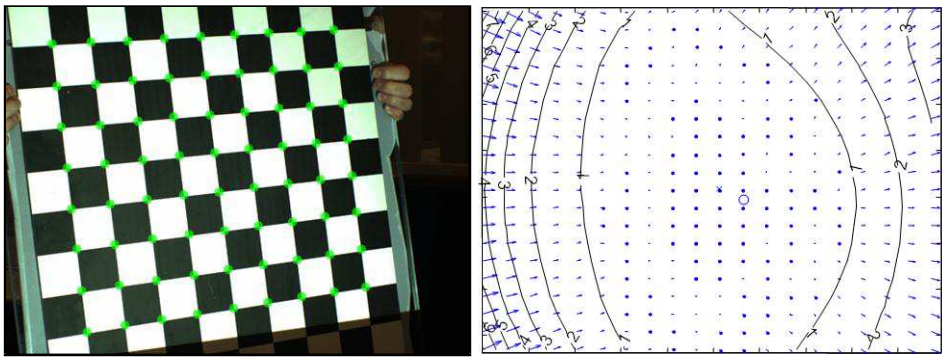
Figure 3.2: Camera calibration sequence containing multiple views of a checkerboard at various positions and orientations throughout the scene.

3.1.3 Calibration Procedure

In this section we describe, step-by-step, how to calibrate your camera using the Camera Calibration Toolbox for MATLAB. We also recommend reviewing the detailed documentation and examples provided on the toolbox website [Bou]. Specifically, new users should work through the first calibration example and familiarize themselves with the description of model parameters (which differ slightly from the notation used in these notes).

Begin by installing the toolbox, available for download at `http://www.vision.caltech.edu/bouguetj/calib_doc/`. Next, construct a checkerboard target. Note that the toolbox comes with a sample checkerboard image; print this image and affix it to a rigid object, such as piece of cardboard or textbook cover. Record a series of 10–20 images of the checkerboard, varying its position and pose between exposures. Try to collect images where the checkerboard is visible throughout the image.

Using the toolbox is relatively straightforward. Begin by adding the toolbox to your MATLAB path by selecting "File → Set Path...". Next, change the current working directory to one containing your calibration images (or one of our test sequences). Type `calib` at the MATLAB prompt to start. Since we're only using a few images, select "Standard (all the images are stored in memory)" when prompted. To load the images, select "Image names" and press return, then "`j`". Now select "Extract grid corners", pass through the prompts without entering any options, and then follow the on-screen directions. (Note that the default checkerboard has 30mm×30mm squares). Always skip any prompts that appear, unless you are more familiar with the toolbox options. Once you've finished selecting



(a) camera calibration (b) camera lens distortion

Figure 3.3: Estimating the intrinsic parameters of the camera. (a) Calibration image collected using a printed checkerboard. A least-squares procedure is used to simultaneously optimize the intrinsic and extrinsic camera parameters in order to minimize the difference between the predicted and known positions of the checkerboard corners (denoted as green circles). (b) The resulting fourth-order lens distortion model for the camera, where isocontours denote the displacement (in pixels) between an ideal pinhole camera image and that collected with the actual lens.

corners, choose "Calibration", which will run one pass though the calibration algorithm. Next, choose "Analyze error". Left-click on any outliers you observe, then right-click to continue. Repeat the corner selection and calibration steps for any remaining outliers (this is a manually-assisted form of bundle adjustment). Once you have an evenly-distributed set of reprojection errors, select "Recomp. corners" and finally "Calibration". To save your intrinsic calibration, select "Save".

From the previous step you now have an estimate of how pixels can be converted into normalized coordinates (and subsequently optical rays in world coordinates, originating at the camera center). Note that this procedure estimates both the intrinsic and extrinsic parameters, as well as the parameters of a lens distortion model. In following chapters, we will describe the use of various functions within the calibration toolbox in more detail. Typical calibration results, illustrating the lens distortion and detected checkerboard corners, are shown in Figure 3.3. Extrinsic calibration results are shown in Figure 3.7, demonstrating that the estimated centers of projection and fields of view correspond with the physical prototype.

Figure 3.4: Recommended projectors for course projects. (Left) Optoma PK-101 Pico Pocket Projector. (Middle) 3M MPro110 Micro Professional Projector. (Right) Mitsubishi XD300U DLP projector used in Chapter 5.

3.2 Projector Calibration

We now turn our attention to projector calibration. Following the conclusions of Chapter 2, we model the projector as an inverse camera (i.e., one in which light travels in the opposite direction as usual). Under this model, calibration proceeds in a similar manner as with cameras. Rather than photographing fixed checkerboards, we *project* known checkerboard patterns and photograph their distorted appearance when reflected from a diffuse rigid object. This approach has the advantage of being a direct extension of Zhang's calibration algorithm for cameras. As a result, much of the software can be shared between camera calibration and projector calibration.

3.2.1 Projector Selection and Interfaces

Almost any digital projector can be used in your 3D scanning projects, since the operating system will simply treat it as an additional display. However, we recommend at least a VGA projector, capable of displaying a 640×480 image. For building a structured lighting system, you'll want to purchase a camera with equal (or higher) resolution as the projector. Otherwise, the recovered model will be limited to the camera resolution. Additionally, those with DVI or HDMI interfaces are preferred for their relative lack of analogue to digital conversion artifacts.

The technologies used in consumer projectors have matured rapidly over the last decade. Early projectors used an LCD-based spatial light modulator and a metal halide lamp, whereas recent models incorporate a digital micromirror device (DMD) and LED lighting. Commercial offerings vary greatly, spanning large units for conference venues to embedded projectors for mobile phones. A variety of technical specifications must be considered

when choosing the "best" projector for your 3D scanning projects. Variations in throw distance (i.e., where focused images can be formed), projector artifacts (i.e., pixelization and distortion), and cost are key factors.

Digital projectors have a tiered pricing model, with brighter projectors costing significantly more than dimmer ones. At the time of writing, a 1024×768 projector can be purchased for around $400–$600 USD. Most models in this price bracket have a 1000:1 contrast ratio with an output around 2000 ANSI lumens. Note that this is about as bright as a typical 100 W incandescent light bulb. Practically, such projectors are sufficient for projecting a 100 inch (diagonal) image in a well-lit room.

For those on a tighter budget, we recommend purchasing a hand-held projector. Also known as "pocket" projectors, these miniaturized devices typically use DMD or LCoS technology together with LED lighting. Current offerings include the 3M MPro, Aiptek V10, Aaxatech P1, and Optoma PK101, with prices around $300 USD. While projectors in this class typically output only 10 lumens, this is sufficient to project up to a 50 inch (diagonal) image (in a darkened room). However, we recommend a higher-lumen projector if you plan on scanning large objects in well-lit environments.

While your system will consider the projector as a second display, your development environment may or may not easily support fullscreen display. For instance, MATLAB does not natively support fullscreen display (i.e., without window borders or menus). One solution is to use Java display functions, with which the MATLAB GUI is built. Code for this approach is available at `http://www.mathworks.com/matlabcentral/fileexchange/11112`. Unfortunately, we found that this approach only works for the primary display. As an alternative, we recommend using the Psychophysics Toolbox [Psy]. While developed for a different application, this toolbox contains OpenGL wrappers allowing simple and direct fullscreen control of the system displays from MATLAB. For details, please see our structured light source code. Finally, for users working outside of MATLAB, we recommend controlling projectors through OpenGL.

3.2.2 Calibration Methods and Software

Projector calibration has received increasing attention, in part driven by the emergence of lower-cost digital projectors. As mentioned at several points, a projector is simply the "inverse" of a camera, wherein points on an image plane are mapped to outgoing light rays passing through the center of projection. As in Section 3.1.2, a lens distortion model can augment the basic general pinhole model presented in Chapter 2.

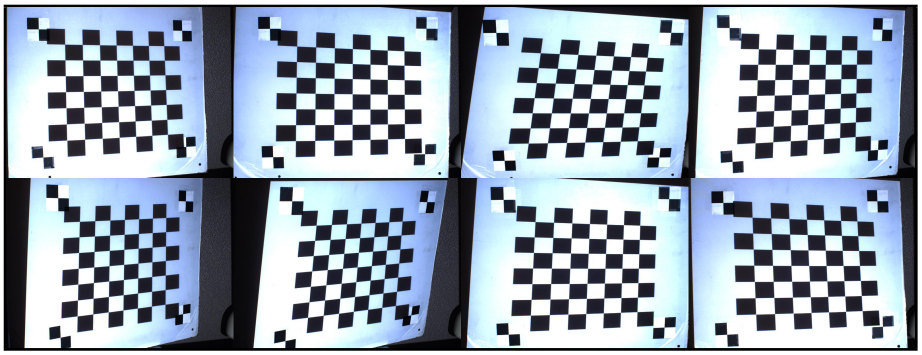
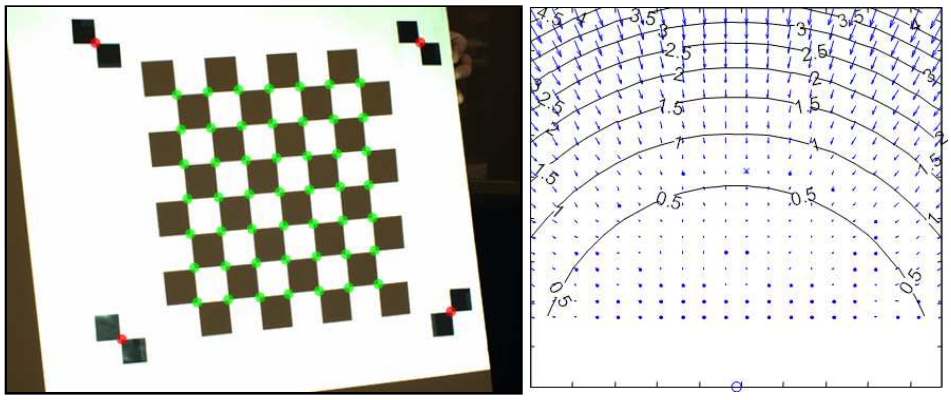


Figure 3.5: Projector calibration sequence containing multiple views of a checkerboard projected on a white plane marked with four printed fiducials in the corners. As for camera calibration, the plane must be moved to various positions and orientations throughout the scene.

Numerous methods have been proposed for estimating the parameters of this inverse camera model. However, community-developed tools are slow to emerge—most researchers keeping their tools in-house. It is our opinion that both OpenCV and the MATLAB calibration toolbox can be easily modified to allow projector calibration. We document our modifications to the latter in the following section. As noted in the OpenCV textbook [BK08], it is expected that similar modifications to OpenCV (possibly arising from this course's attendees) will be made available soon.

3.2.3 Calibration Procedure

In this section we describe, step-by-step, how to calibrate your projector using our software, which is built on top of the Camera Calibration Toolbox for MATLAB. Begin by calibrating your camera(s) using the procedure outlined in the previous section. Next, install the toolbox extensions available on the course website at `http://mesh.brown.edu/dlanman/scan3d`. Construct a calibration object similar to those in Figures 3.5 and 3.6. This object should be a diffuse white planar object, such as foamcore or a painted piece of particle board. Printed fiducials, possibly cut from a section of your camera calibration pattern, should be affixed to the surface. One option is to simply paste a section of the checkerboard pattern in one corner. In our implementation we place four checkerboard corners at the edges of the calibration object. The distances and angles between these points should be recorded.

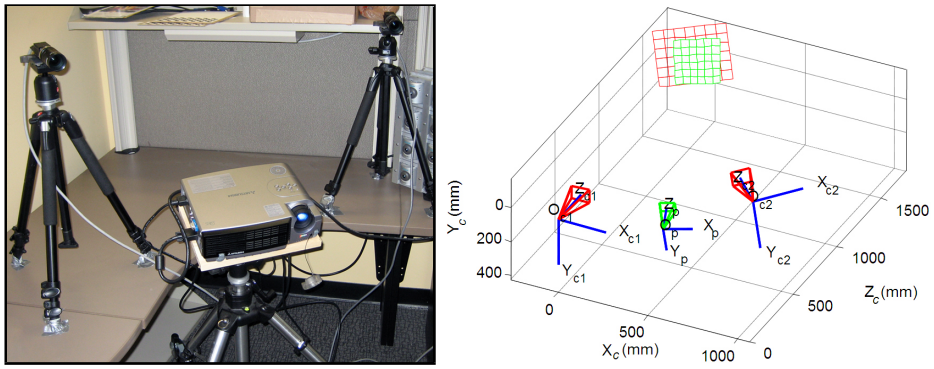


(a) projector calibration (b) projector lens distortion

Figure 3.6: Estimating the intrinsic parameters of the projector using a calibrated camera. (a) Calibration image collected using a white plane marked with four fiducials in the corners (denoted as red circles). (b) The resulting fourth-order lens distortion model for the projector.

A known checkerboard must be projected onto the calibration object. We have provided `run_capture` to generate the checkerboard pattern, as well as collect the calibration sequence. As previously mentioned, this script controls the projector using the Psychophysics Toolbox [Psy]. A series of 10–20 images should be recorded by projecting the checkerboard onto the calibration object. Suitable calibration images are shown in Figure 3.5. Note that the printed fiducials must be visible in each image and that the projected checkerboard should not obscure them. There are a variety of methods to prevent projected and printed checkerboards from interfering; one solution is to use color separation (e.g., printed and projected checkerboards in red and blue, respectively), however this requires the camera be color calibrated. We encourage you to try a variety of options and send us your results for documentation on the course website.

Your camera calibration images should be stored in the cam subdirectory of the provided projector calibration package. The `calib_data.mat` file, produced by running camera calibration, should be stored in this directory as well. The projector calibration images should be stored in the proj subdirectory. Afterwards, run the camera calibration toolbox by typing `calib` at the MATLAB prompt (in this directory). Since we're only using a few images, select "Standard (all the images are stored in memory)" when prompted. To load the images, select "Image names" and press return,



(a) projector-camera system (b) extrinsic calibration

Figure 3.7: Extrinsic calibration of a projector-camera system. (a) A structured light system with a pair of digital cameras and a projector. (b) Visualization of the extrinsic calibration results.

then "`j`". Now select "Extract grid corners", pass through the prompts without entering any options, and then follow the on-screen directions. Always skip any prompts that appear, unless you are more familiar with the toolbox options. Note that you should now select the *projected* checkerboard corners, not the printed fiducials. The detected corners should then be saved in `calib_data.mat` in the proj subdirectory.

To complete your calibration, run the `run_calibration` script. Note that you may need to modify which projector images are included at the top of the script (defined by the useProjImages vector), especially if you find that the script produces an optimization error message. The first time you run the script, you will be prompted to select the extrinsic calibration fiducials (i.e., the four printed markers in Figure 3.5). Follow any on-screen directions. Once calibration is complete, the script will visualize the recovered system parameters by plotting the position and field of view of the projector and camera (see Figure 3.7).

Our modifications to the calibration toolbox are minimal, reusing much of its functionality. We plan on adding a simple GUI to automate the manual steps currently needed with our software. Please check the course website for any updates. In addition, we will we post links to similar software tools produced by course attendees. In any case, we hope that the provided software is sufficient to overcome the "calibration hurdle" in their own 3D scanning projects.

Chapter 4

3D Scanning with Swept-Planes

In this chapter we describe how to build an inexpensive, yet accurate, 3D scanner using household items and a digital camera. Specifically, we'll describe the implementation of the "desktop scanner" originally proposed by Bouguet and Perona [BP]. As shown in Figure 4.1, our instantiation of this system is composed of five primary items: a digital camera, a point-like light source, a stick, two planar surfaces, and a calibration checkerboard. By waving the stick in front of the light source, the user can cast planar shadows into the scene. As we'll demonstrate, the depth at each pixel can then be recovered using simple geometric reasoning.

In the course of building your own "desktop scanner" you will need to develop a good understanding of camera calibration, Euclidean coordinate transformations, manipulation of implicit and parametric parameterizations of lines and planes, and efficient numerical methods for solving least-squares problems—topics that were previously presented in Chapter 2. We encourage the reader to also review the original project website [BP] and obtain a copy of the IJCV publication [BP99], both of which will be referred to several times throughout this chapter. Also note that the software accompanying this chapter was developed in MATLAB at the time of writing. We encourage the reader to download that version, as well as updates, from the course website at `http://mesh.brown.edu/dlanman/scan3d`.

4.1 Data Capture

As shown in Figure 4.1, the scanning apparatus is simple to construct and contains relatively few components. A pair of blank white foamcore boards are used as planar calibration objects. These boards can be purchased at an

(a) swept-plane scanning apparatus (b) frame from acquired video sequence

Figure 4.1: 3D photography using planar shadows. (a) The scanning setup, composed of five primary items: a digital camera, a point-like light source, a stick, two planar surfaces, and a calibration checkerboard (not shown). Note that the light source and camera must be separated so that cast shadow planes and camera rays do not meet at small incidence angles. (b) The stick is slowly waved in front of the point light to cast a planar shadow that translates from left to right in the scene. The position and orientation of the shadow plane, in the world coordinate system, are estimated by observing its position on the planar surfaces. After calibrating the camera, a 3D model can be recovered by triangulation of each optical ray by the shadow plane that first entered the corresponding scene point.

art supply store. Any rigid light-colored planar object could be substituted, including particle board, acrylic sheets, or even lightweight poster board. At least four fiducials, such as the printed checkerboard corners shown in the figure, should be affixed to known locations on each board. The distance and angle between each fiducial should be measured and recorded for later use in the calibration phase. These measurements will allow the position and orientation of each board to be estimated in the world coordinate system. Finally, the boards should be oriented approximately at a right angle to one another.

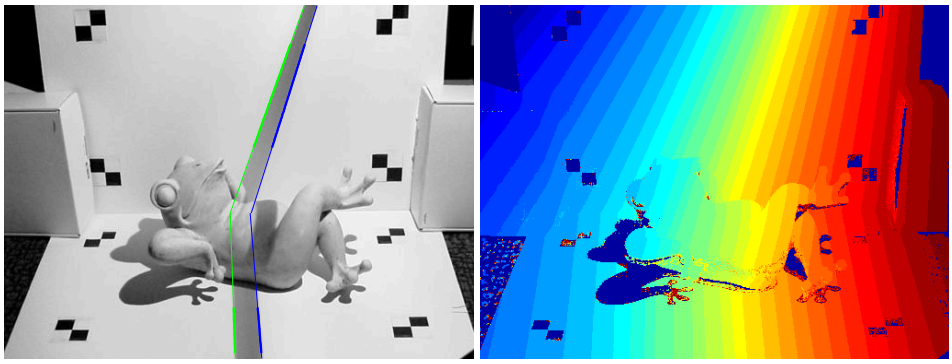
Next, a planar light source must be constructed. In this chapter we will follow the method of Bouguet and Perona [BP], in which a point source and a stick are used to cast planar shadows. Wooden dowels of varying diameter can be obtained at a hardware store, and the point light source can be fashioned from any halogen desk lamp after removing the reflector. Alternatively, a laser stripe scanner could be implemented by replacing the

point light source and stick with a modified laser pointer. In this case, a cylindrical lens must be affixed to the exit aperture of the laser pointer, creating a low-cost laser stripe projector. Both components can be obtained from Edmund Optics [Edm]. For example, a section of a lenticular array or cylindrical Fresnel lens sheet could be used. However, in the remainder of this chapter we will focus on the shadow-casting method.

Any video camera or webcam can be used for image acquisition. The light source and camera should be separated, so that the angle between camera rays and cast shadow planes is close to perpendicular (otherwise triangulation will result in large errors). Data acquisition is simple. First, an object is placed on the horizontal calibration board, and the stick is slowly translated in front of the light source (see Figure 4.1). The stick should be waved such that a thin shadow slowly moves across the screen in one direction. Each point on the object should be shadowed at some point during data acquisition. Note that the camera frame rate will determine how fast the stick can be waved. If it is moved too fast, then some pixels will not be shadowed in any frame—leading to reconstruction artifacts.

We have provided several test sequences with our setup, which are available on the course website [LT]. As shown in Figures 4.3–4.7, there are a variety of objects available, ranging from those with smooth surfaces to those with multiple self-occlusions. As we'll describe in the following sections, reconstruction requires accurate estimates of the shadow boundaries. As a result, you will find that light-colored objects (e.g., the chiquita, frog, and man sequences) will be easiest to reconstruct. Since you'll need to estimate the intrinsic and extrinsic calibration of the camera, we've also provided the calib sequence composed of ten images of a checkerboard with various poses. For each sequence we have provided both a high-resolution 1024×768 sequence, as well as a low-resolution 512×384 sequence for development.

When building your own scanning apparatus, briefly note some practical issues associated with this approach. First, it is important that every pixel be shadowed at some point in the sequence. As a result, you must wave the stick slow enough to ensure that this condition holds. In addition, the reconstruction method requires reliable estimates of the plane defined by the light source and the edge of the stick. Ambient illumination must be reduced so that a single planar shadow is cast by each edge of the stick. In addition, the light source must be sufficiently bright to allow the camera to operate with minimal gain, otherwise sensor noise will corrupt the final reconstruction. Finally, note that these systems typically use a single halogen desk lamp with the reflector removed. This ensures that the light source is



(a) spatial shadow edge localization (b) temporal shadow edge localization

Figure 4.2: Spatial and temporal shadow edge localization. (a) The shadow edges are determined by fitting a line to the set of zero crossings, along each row in the planar regions, of the difference image $\Delta I(x, y, t)$. (b) The shadow times (quantized to 32 values here) are determined by finding the zero-crossings of the difference image $\Delta I(x, y, t)$ for each pixel (x, y) as a function of time t. Early to late shadow times are shaded from blue to red.

sufficiently point-like to produce abrupt shadow boundaries.

4.2 Video Processing

Two fundamental quantities must be estimated from a recorded swept-plane video sequence: (1) the time that the shadow enters (and/or leaves) each pixel and (2) the spatial position of each leading (and/or trailing) shadow edge as a function of time. This section outlines the basic procedures for performing these tasks. Additional technical details are presented in Section 2.4 in [BP99] and Section 6.2.4 in [Bou99]. Our reference implementation is provided in the `videoProcessing` m-file. Note that, for large stick diameters, the shadow will be thick enough that two distinct edges can be resolved in the captured imagery. By tracking both the leading and trailing shadow edges, two independent 3D reconstructions are obtained—allowing robust outlier rejection and improved model quality. However, in a basic implementation, only one shadow edge must be processed using the following methods. In this section we will describe calibration of the leading edge, with a similar approach applying for the trailing edge.

4.2.1 Spatial Shadow Edge Localization

To reconstruct a 3D model, we must know the equation of each shadow plane in the world coordinate system. As shown in Figure 4.2, the cast shadow will create four distinct lines in the camera image, consisting of a pair of lines on both the horizontal and vertical calibration boards. These lines represent the intersection of the 3D shadow planes (both the leading and trailing edges) with the calibration boards. Using the notation of Figure 2 in [BP99], we need to estimate the 2D shadow lines $\lambda_h(t)$ and $\lambda_v(t)$ projected on the horizontal and vertical planar regions, respectively. In order to perform this and subsequent processing, a *spatio-temporal* approach can be used. As described in by Zhang et al. [ZCS03], this approach tends to produce better reconstruction results than traditional edge detection schemes (e.g., the Canny edge detector [MSKS05]), since it is capable of preserving sharp surface discontinuities.

Begin by converting the video to grayscale (if a color camera was used), and evaluate the maximum and minimum brightness observed at each camera pixel $\bar{x}_c = (x, y)$ over the stick-waving sequence.

$$I_{\text{max}}(x, y) \triangleq \max_t I(x, y, t)$$
$$I_{\text{min}}(x, y) \triangleq \min_t I(x, y, t)$$

To detect the shadow boundaries, choose a per-pixel detection threshold which is the midpoint of the dynamic range observed in each pixel. With this threshold, the shadow edge can be localized by the zero crossings of the difference image

$$\Delta I(x, y, t) \triangleq I(x, y, t) - I_{\text{shadow}}(x, y),$$

where the shadow threshold image is defined to be

$$I_{\text{shadow}}(x, y) \triangleq \frac{I_{\text{max}}(x, y) + I_{\text{min}}(x, y)}{2}.$$

In practice, you'll need to select an occlusion-free image patch for each planar region. Afterwards, a set of sub-pixel shadow edge samples (for each row of the patch) are obtained by interpolating the position of the zero-crossings of $\Delta I(x, y, t)$. To produce a final estimate of the shadow edges $\lambda_h(t)$ and $\lambda_v(t)$, the best-fit line (in the least-squares sense) must be fit to the set of shadow edge samples. The desired output of this step is illustrated in Figure 4.2(a), where the best-fit lines are overlaid on the original

image. Keep in mind that you should convert the provided color images to grayscale; if you're using MATLAB, the function `rgb2gray` can be used for this task.

4.2.2 Temporal Shadow Edge Localization

After calibrating the camera, the previous step will provide all the information necessary to recover the position and orientation of each shadow plane as a function of time in the world coordinate system. As we'll describe in Section 4.4, in order to reconstruct the object you'll also need to know when each pixel entered the shadowed region. This task can be accomplished in a similar manner as spatial localization. Instead of estimating zero-crossing along each row for a fixed frame, the per-pixel shadow time is assigned using the zero crossings of the difference image $\Delta I(x, y, t)$ for each pixel (x, y) as a function of time t. The desired output of this step is illustrated in Figure 4.2(b), where the shadow crossing times are quantized to 32 values (with blue indicating earlier times and red indicated later ones). Note that you may want to include some additional heuristics to reduce false detections. For instance, dark regions cannot be reliably assigned a shadow time. As a result, you can eliminate pixels with insufficient contrast (e.g., dark blue regions in the figure).

4.3 Calibration

As described in Chapters 2 and 3, intrinsic and extrinsic calibration of the camera is necessary to transfer image measurements into the world coordinate system. For the swept-plane scanner, we recommend using either the Camera Calibration Toolbox for MATLAB [Bou] or the calibration functions within OpenCV [Opea]. As previously described, these packages are commonly used within the computer vision community and, at their core, implement the widely adopted calibration method originally proposed by Zhang [Zha99]. In this scheme, the intrinsic and extrinsic parameters are estimated by viewing several images of a planar checkerboard with various poses. In this section we will briefly review the steps necessary to calibrate the camera using the MATLAB toolbox. We recommend reviewing the documentation on the toolbox website [Bou] for additional examples; specifically, the first calibration example and the description of calibration parameters are particularly useful to review for new users.

4.3.1 Intrinsic Calibration

Begin by adding the toolbox to your MATLAB path by selecting "File → Set Path...". Next, change the current working directory to one of the calibration sequences (e.g., to the calib or calib-lr examples downloaded from the course website). Type `calib` at the MATLAB prompt to start. Since we're only using a few images, select "Standard (all the images are stored in memory)" when prompted. To load the images, select "Image names" and press return, then "`j`". Now select "Extract grid corners", pass through the prompts without entering any options, and then follow the on-screen directions. (Note that, for the provided examples, a calibration target with the default 30mm×30mm squares was used). Always skip any prompts that appear, unless you are more familiar with the toolbox options. Once you've finished selecting corners, choose "Calibration", which will run one pass though the calibration algorithm discussed in Chapter 3. Next, choose "Analyze error". Left-click on any outliers you observe, then right-click to continue. Repeat the corner selection and calibration steps for any remaining outliers (this is a manually-assisted form of bundle adjustment). Once you have an evenly-distributed set of reprojection errors, select "Recomp. corners" and finally "Calibration". To save your intrinsic calibration, select "Save".

4.3.2 Extrinsic Calibration

From the previous step you now have an estimate of how pixels can be converted into normalized coordinates (and subsequently optical rays in world coordinates, originating at the camera center). In order to assist you with your implementation, we have provided a MATLAB script called `extrinsicDemo`. As long as the calibration results have been saved in the calib and calib-lr directories, this demo will allow you to select four corners on the "horizontal" plane to determine the Euclidean transformation from this ground plane to the camera reference frame. (Always start by selecting the corner in the bottom-left and proceed in a counter-clockwise order. For your reference, the corners define a 558.8mm×303.2125mm rectangle in the provided test sequences.) In addition, observe that the final section of `extrinsicDemo` uses the utility function `pixel2ray` to determine the optical rays (in camera coordinates), given a set of user-selected pixels.

4.4 Reconstruction

At this point, the system is fully calibrated. Specifically, optical rays passing through the camera's center of projection can be expressed in the same world coordinate system as the set of temporally-indexed shadow planes. Ray-plane triangulation can now be applied to estimate the per-pixel depth (at least for those pixels where the shadow was observed). In terms of Figure 2 in [BP99], the camera calibration is used to obtain a parametrization of the ray defined by a true object point P and the camera center O_c. Given the shadow time for the associated pixel $\bar{x}_c$, one can lookup (and potentially interpolate) the position of the shadow plane at this time. The resulting ray-plane intersection will provide an estimate of the 3D position of the surface point. Repeating this procedure for every pixel will produce a 3D reconstruction. For complementary and extended details on the reconstruction process, please consult Sections 2.5 and 2.6 in [BP99] and Sections 6.2.5 and 6.2.6 in [Bou99].

4.5 Post-processing and Visualization

Once you have reconstructed a 3D point cloud, you'll want to visualize the result. Regardless of the environment you used to develop your solution, you can write a function to export the recovered points as a VRML file containing a single indexed face set with an empty coordIndex array. Additionally, a per-vertex color can be assigned by sampling the maximum-luminance color, observed over the video sequence. In Chapter 6 we document further post-processing that can be applied, including merging multiple scans and extracting watertight meshes. However, the simple colored point clouds produced at this stage can be rendered using the Java-based point splatting software provided on the course website.

To give you some expectation of reconstruction quality, Figures 4.3–4.7 show results obtained with our reference implementation. Note that there are several choices you can make in your implementation; some of these may allow you to obtain additional points on the surface or increase the reconstruction accuracy. For example, using both the leading and trailing shadow edges will allow outliers to be rejected (by eliminating points whose estimated depth disagrees between the leading vs. trailing shadow edges).

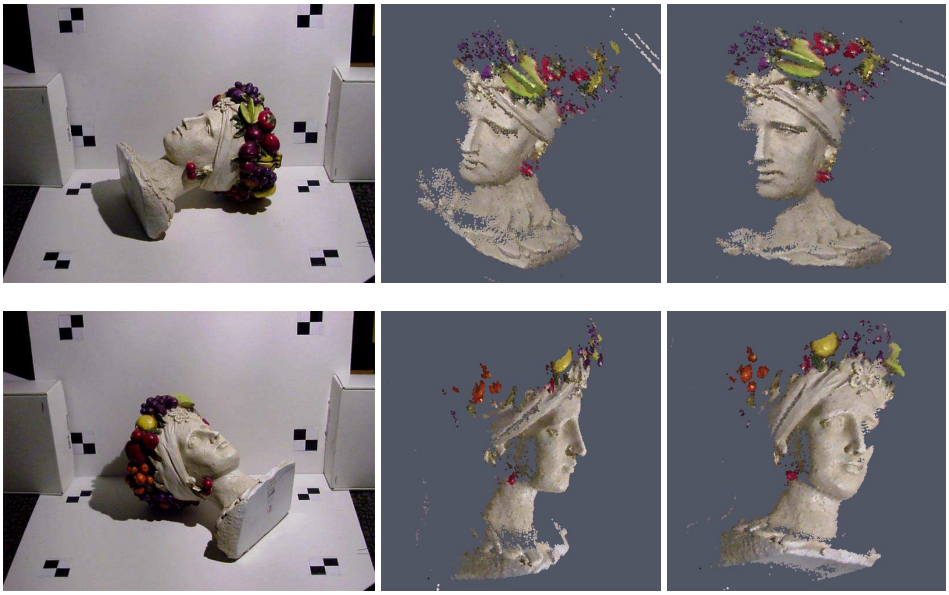


Figure 4.3: Reconstruction of the chiquita-v1 and chiquita-v2 sequences.

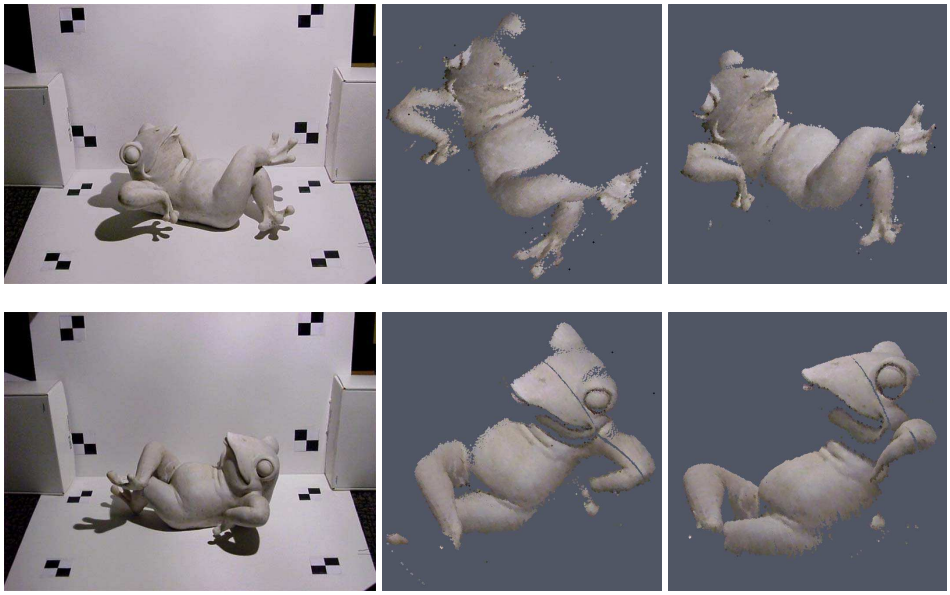


Figure 4.4: Reconstruction of the frog-v1 and frog-v2 sequences.

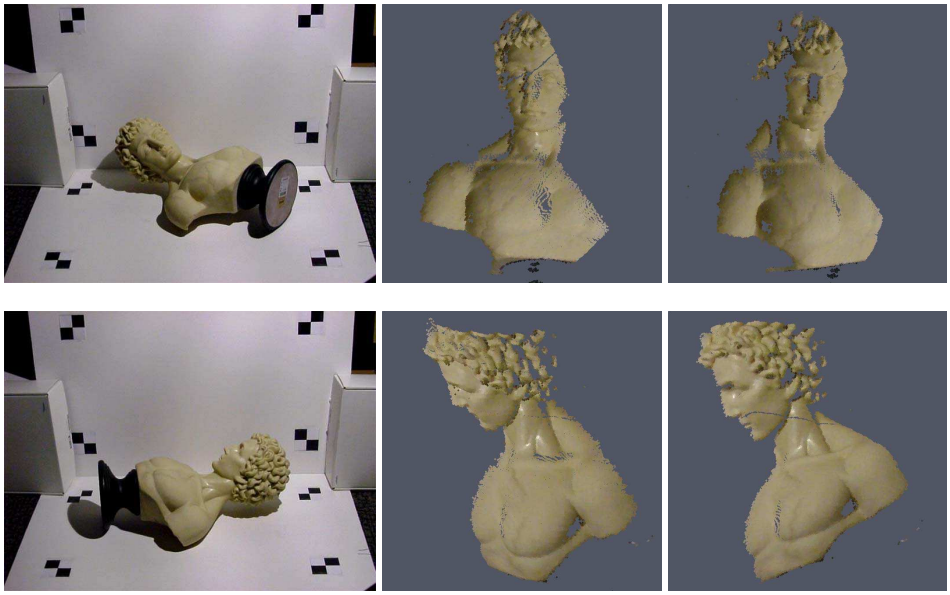


Figure 4.5: Reconstruction of the man-v1 and man-v2 sequences.

Figure 4.6: Reconstruction of the schooner sequence.

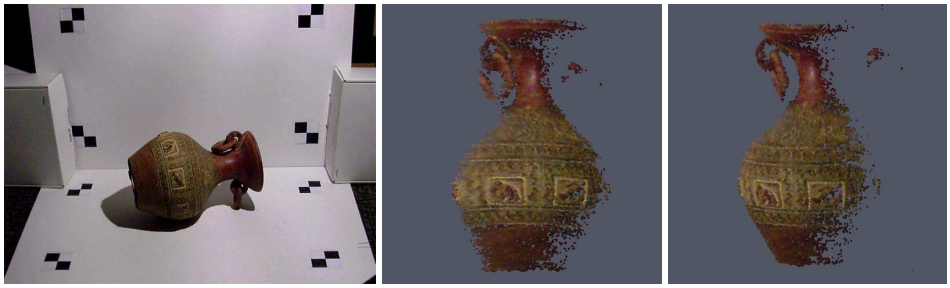


Figure 4.7: Reconstruction of the urn sequence.

Chapter 5

Structured Lighting

In this chapter we describe how to build a structured light scanner using one or more digital cameras and a single projector. While the "desktop scanner" [BP] implemented in the previous chapter is inexpensive, it has limited practical utility. The scanning process requires manual manipulation of the stick, and the time required to sweep the shadow plane across the scene limits the system to reconstructing static objects. Manual translation can be eliminated by using a digital projector to sequentially display patterns (e.g., a single stipe translated over time). Furthermore, various *structured light* illumination sequences, consisting of a series of projected images, can be used to efficiently solve for the camera pixel to projector column (or row) correspondences.

By implementing your own structured light scanner, you will directly extending the algorithms and software developed for the swept-plane systems in the previous chapter. Reconstruction will again be accomplished using ray-plane triangulation. The key difference is that correspondences will now be established by decoding certain structured light sequences. At the time of writing, the software accompanying this chapter was developed in MATLAB. We encourage the reader to download that version, as well as any updates, from the course website at `http://mesh.brown.edu/dlanman/scan3d`.

5.1 Data Capture

5.1.1 Scanner Hardware

As shown in Figure 5.1, the scanning apparatus contains one or more digital cameras and a single digital projector. As with the swept-plane systems,

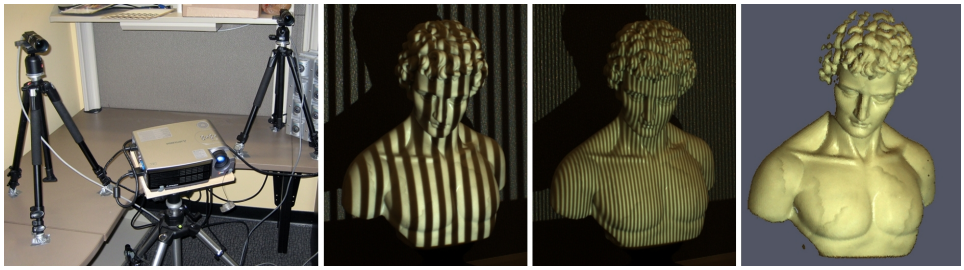


Figure 5.1: Structured light for 3D scanning. From left to right: a structured light scanning system containing a pair of digital cameras and a single projector, two images of an object illuminated by different bit planes of a Gray code structured light sequence, and a reconstructed 3D point cloud.

the object will eventually be reconstructed by ray-plane triangulation, between each camera ray and a plane corresponding to the projector column (and/or row) illuminating that point on the surface. As before, the cameras and projector should be arranged to ensure that no camera ray and projector plane meet at small incidence angles. A "diagonal" placement of the cameras, as shown in the figure, ensures that both projector rows and columns can be used for reconstruction.

As briefly described in Chapter 3, a wide variety of digital cameras and projectors can be selected for your implementation. While low-cost webcams will be sufficient, access to raw imagery will eliminate decoding errors introduced by compression artifacts. You will want to select a camera that is be supported by your preferred development environment. For example, if you plan on using the MATLAB Image Acquisition Toolbox, then any DCAM-compatible FireWire camera or webcam with a Windows Driver Model (WDM) or Video for Windows (VFW) driver will work [Mat]. If you plan on developing in OpenCV, a list compatible cameras is maintained on the wiki [Opeb]. Almost any digital projector can be used, since the operating system will simply treat it as an additional display.

As a point of reference, our implementation contains a single Mitsubishi XD300U DLP projector and a pair of Point Grey GRAS-20S4M/C Grasshopper video cameras. The projector is capable of displaying 1024×768 24-bit RGB images at 50-85 Hz [Mit]. The cameras capture 1600×1200 24-bit RGB images at up to 30 Hz [Poia]. Although lower-resolution modes can be used if higher frame rates are required. The data capture was implemented in MATLAB. The cameras were controlled using custom wrappers for the FlyCapture SDK [Poib], and fullscreen control of the projector was achiev-

ing using the Psychophysics Toolbox [Psy] (see Chapter 3).

5.1.2 Structured Light Sequences

The primary benefit of introducing the projector is to eliminate the mechanical motion required in swept-plane scanning systems (e.g., laser striping or the "desktop scanner"). Assuming minimal lens distortion, the projector can be used to display a single column (or row) of white pixels translating against a black background; thus, 1024 (or 768) images would be required to assign the correspondences, in our implementation, between camera pixels and projector columns (or rows). After establishing the correspondences and calibrating the system, a 3D point cloud is reconstructed using familiar ray-plane triangulation. However, a simple swept-plane sequence does not fully exploit the projector. Since we are free to project arbitrary 24-bit color images, one would expect there to exist a sequence of coded patterns, besides a simple translation of a single stripe, that allow the projector-camera correspondences to be assigned in relatively few frames. In general, the identity of each plane can be encoded spatially (i.e., within a single frame) or temporally (i.e., across multiple frames), or with a combination of both spatial and temporal encodings. There are benefits and drawbacks to each strategy. For instance, purely spatial encodings allow a single static pattern to be used for reconstruction, enabling dynamic scenes to be captured. Alternatively, purely temporal encodings are more likely to benefit from redundancy, reducing reconstruction artifacts. A comprehensive assessment of such codes is presented by Salvi et al. [SPB04].

In this chapter we will focus on purely temporal encodings. While such patterns are not well-suited to scanning dynamic scenes, they have the benefit of being easy to decode and are robust to surface texture variation, producing accurate reconstructions for static objects (with the normal prohibition of transparent or other problematic materials). A simple binary structured light sequence was first proposed by Posdamer and Altschuler [PA82] in 1981. As shown in Figure 5.2, the binary encoding consists of a sequence of binary images in which each frame is a single bit plane of the binary representation of the integer indices for the projector columns (or rows). For example, column 546 in our prototype has a binary representation of 1000100010 (ordered from the most to the least significant bit). Similarly, column 546 of the binary structured light sequence has an identical bit sequence, with each frame displaying the next bit.

Considering the projector-camera arrangement as a communication system, then a key question immediately arises; what binary sequence is most

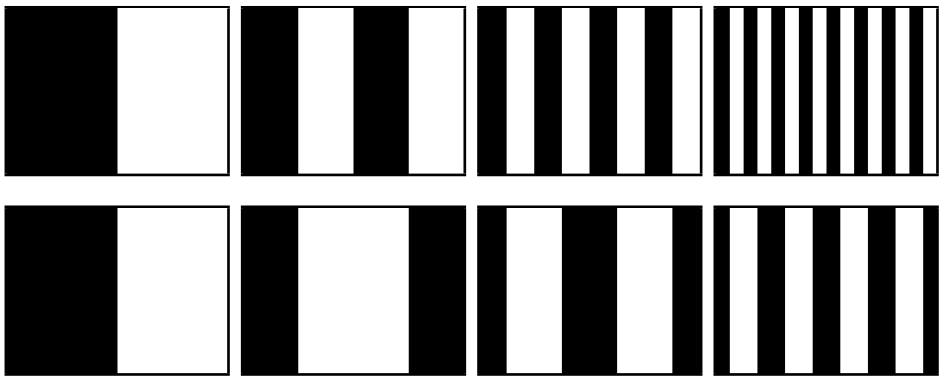
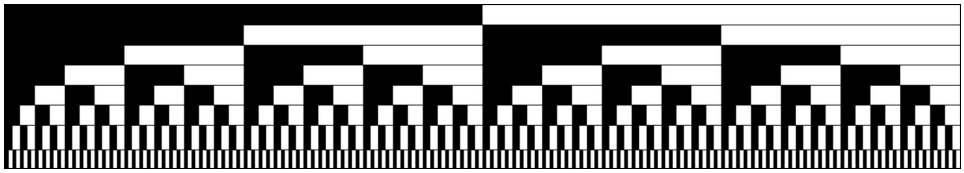


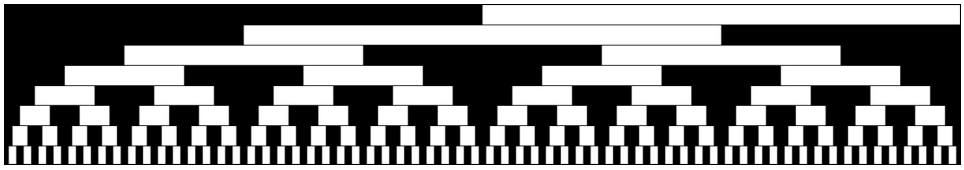
Figure 5.2: Structured light illumination sequences. (Top row, left to right) The first four bit planes of a binary encoding of the projector columns, ordered from most to least significant bit. (Bottom row, left to right) The first four bit planes of a Gray code sequence encoding the projector columns.

robust to the known properties of the channel noise process? At a basic level, we are concerned with assigning an accurate projector column/row to camera pixel correspondence, otherwise triangulation artifacts will lead to large reconstruction errors. Gray codes were first proposed as one alternative to the simple binary encoding by Inokuchi et al. [ISM84] in 1984. The *reflected binary code* was introduced by Frank Gray in 1947 [Wik]. As shown in Figure 5.3, the Gray code can be obtained by reflecting, in a specific manner, the individual bit-planes of the binary encoding. Pseudocode for converting between binary and Gray codes is provided in Table 5.1. For example, column 546 in our in our implementation has a Gray code representation of 1100110011, as given by BIN2GRAY. The key property of the Gray code is that two neighboring code words (e.g., neighboring columns in the projected sequence) only differ by one bit (i.e., adjacent codes have a Hamming distance of one). As a result, the Gray code structured light sequence tends to be more robust to decoding errors than a simple binary encoding.

In the provided MATLAB code, the m-file `bincode` can be used to generate a binary structured light sequence. The inputs to this function are the width w and height h of the projected image. The output is a sequence of $2\lceil \log_2 w \rceil + 2\lceil \log_2 h \rceil + 2$ uncompressed images. The first two images consist of an all-white and an all-black image, respectively. The next $2\lceil \log_2 w \rceil$ images contain the bit planes of the binary sequence encoding the projec-



(a) binary structured light sequence



(b) Gray code structured light sequence

Figure 5.3: Comparison of binary (top) and Gray code (bottom) structured light sequences. Each image represents the sequence of bit planes displayed during data acquisition. Image rows correspond to the bit planes encoding the projector columns, assuming a projector resolution of 1024×768, ordered from most to least significant bit (from top to bottom).

tor columns, interleaved with the binary inverse of each bit plane (to assist in decoding). The last $2\lceil\log_2 h\rceil$ images contain a similar encoding for the projector rows. A similar m-file named `graycode` is provided to generate Gray code structured light sequences.

5.2 Image Processing

The algorithms used to decode the structured light sequences described in the previous section are relatively straightforward. For each camera, it must be determined whether a given pixel is directly illuminated by the projector in each displayed image. If it is illuminated in any given frame, then the corresponding code bit is set high, otherwise it is set low. The decimal integer index of the corresponding projector column (and/or row) can then be recovered by decoding the received bit sequences for each camera pixel. A user-selected intensity threshold is used to determine whether a given pixel is illuminated. For instance, $\lceil\log_2 w\rceil + 2$ images could be used to encode the projector columns, with the additional two images consisting of all-white and all-black frames. The average intensity of the all-white and all-black frames could be used to assign a per-pixel threshold; the in-

```
BIN2GRAY(B)
1  n ← length[B]
2  G[1] ← B[1]
3  for i ← 2 to n
4     do G[i] ← B[i − 1] xor B[i]
5  return G
```

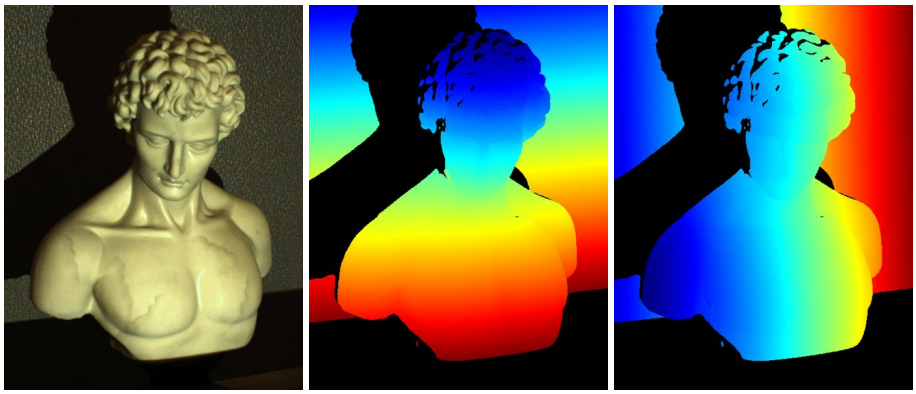
```
GRAY2BIN(G)
1  n ← length[G]
2  B[1] ← G[1]
3  for i ← 2 to n
4     do B[i] ← B[i − 1] xor G[i]
5  return B
```

Table 5.1: Pseudocode for converting between binary and Gray codes. (Left) BIN2GRAY accepts an n-bit Boolean array, encoding a decimal integer, and returns the Gray code G. (Right) Conversion from a Gray to a binary sequence is accomplished using GRAY2BIN.

dividual bit planes of the projected sequence could then be decoded by comparing the received intensity to the threshold.

In practice, a single fixed threshold results in decoding artifacts. For instance, certain points on the surface may only receive indirect illumination scattered from directly-illuminated points. In certain circumstances the scattered light may cause a bit error, in which an unilluminated point appears illuminated due to scattered light. Depending on the specific structured light sequence, such bit errors may produce significant reconstruction errors in the 3D point cloud. One solution is to project each bit plane and its inverse, as was done in Section 5.1. While $2\lceil \log_2 w \rceil$ frames are now required to encode the projector columns, the decoding process is less sensitive to scattered light, since a variable per-pixel threshold can be used. Specifically, a bit is determined to be high or low depending on whether a projected bit-plane or its inverse is brighter at a given pixel. Typical decoding results are shown in Figure 5.4.

As with any communication system, the design of structured light sequences must account for anticipated artifacts introduced by the communication channel. In a typical projector-camera system decoding artifacts are introduced from a wide variety of sources, including projector or camera defocus, scattering of light from the surface, and temporal variation in the scene (e.g., varying ambient illumination or a moving object). We have provided a variety of data sets for testing your decoding algorithms. In particular, the man sequence has been captured using both binary and Gray code structured light sequences. Furthermore, both codes have been applied when the projector is focused and defocused at the average depth of the sculpture. We encourage the reader to study the decoding artifacts produced under these non-ideal, yet commonly encountered, circumstances.



(a) all-white image (b) decoded row indices (c) decoded column indices

Figure 5.4: Decoding structured light illumination sequences. (a) Camera image captured while projecting an all white frame. Note the shadow cast on the background plane, prohibiting reconstruction in this region. (b) Typical decoding results for a Gray code structured light sequence, with projector row and camera pixel correspondences represented using a `jet` colormap in MATLAB. Points that cannot be assigned a correspondence with a high confidence are shown in black. (c) Similar decoding results for projector column correspondences.

The support code includes the m-file `bindecode` to decode the provided binary structured light sequences. This function accepts as input the directory containing the encoded sequences, following the convention of the previous section. The output is a pair of unsigned 16-bit grayscale images containing the decoded decimal integers corresponding to the projector column and row that illuminated each camera pixel (see Figure 5.4). A value of zero indicates a given pixel cannot be assigned a correspondence, and the projector columns and rows are indexed from one. The m-file `graydecode` is also provided to decode Gray code structured light sequences. Note that our implementation of the Gray code is shifted to the left, if the number of columns (or rows) is not a power of two, such that the projected patterns are symmetric about the center column (or row) of the image. The sample script `slDisplay` can be used to load and visualize the provided data sets.

5.3 Calibration

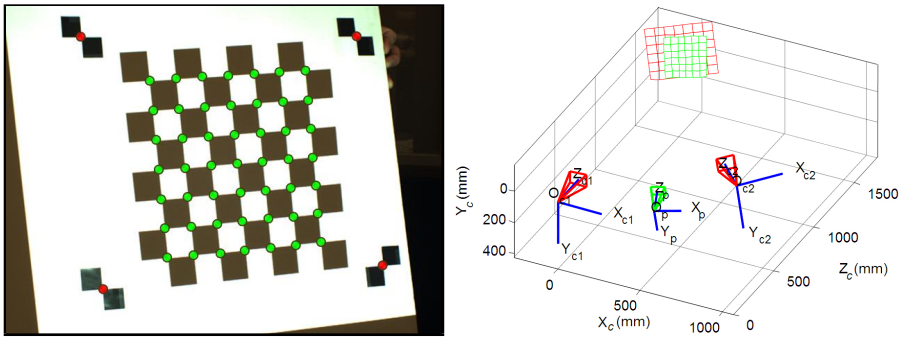
As with the swept-plane scanner, calibration is accomplished using any of the tools and procedures outlined in Chapter 3. In this section we briefly review the basic procedures for projector-camera calibration. In our implementation, we used the Camera Calibration Toolbox for MATLAB [Bou] to first calibrate the cameras, following the approach used in the previous chapter. An example sequence of 15 views of a planar checkerboard pattern, composed of 38mm×38mm squares, is provided in the accompanying test data for this chapter. The intrinsic and extrinsic camera calibration parameters, in the format specified by the toolbox, are also provided.

Projector calibration is achieved using our extensions to the Camera Calibration Toolbox for MATLAB, as outlined in Chapter 3. As presented, the projector is modeled as a pinhole imaging system containing additional lenses that introduce distortion. As with our cameras, the projector has an intrinsic model involving the principal point, skew coefficients, scale factors, and focal length.

To estimate the projector parameters, a static checkerboard is projected onto a diffuse planar pattern with a small number of printed fiducials located on its surface. In our design, we used a piece of foamcore with four printed checkerboard corners. As shown in Figure 5.5, a single image of the printed fiducials is used to recover the implicit equation of the calibration plane in the camera coordinate system. The 3D coordinate for each projected checkerboard corner is then reconstructed. The 2D camera pixel to 3D point correspondences are then used to estimate the intrinsic and extrinsic calibration from multiple views of the planar calibration object.

A set of 20 example images of the projector calibration object are included with the support code. In these examples, the printed fiducials were horizontally separated by 406mm and vertically separated by 335mm. The camera and projector calibration obtained using our procedure are also provided; note that the projector intrinsic and extrinsic parameters are in the same format as camera calibration outputs from the Camera Calibration Toolbox for MATLAB. The provided m-file `slCalib` can be used to visualize the calibration results.

A variety of Matlab utilities are provided to assist the reader in implementing their own structured light scanner. The m-file `campixel2ray` converts from camera pixel coordinates to an optical ray expressed in the coordinate system of the first camera (if more than one camera is used). A similar m-file `projpixel2ray` converts from projector pixel coordinates to an optical ray expressed in the common coordinate system of the first



(a) projector-camera system (b) extrinsic calibration

Figure 5.5: Extrinsic calibration of a projector-camera system. (a) A planar calibration object with four printed checkerboard corners is imaged by a camera. A projected checkerboard is displayed in the center of the calibration plane. The physical and projected corners are manually detected and indicated with red and green circles, respectively. (b) The extrinsic camera and projector calibration is visualized using `slCalib`. Viewing frusta for the cameras are shown in red and the viewing frustum for the projector is shown in green. Note that the reconstruction of the first image of a single printed checkerboard, used during camera calibration, is shown with a red grid, whereas the recovered projected checkerboard is shown in green. Also note that the recovered camera and projector frusta correspond to the physical configuration shown in Figure 5.1.

camera. Finally, `projcol2plane` and `projrow2plane` convert from projected column and row indices, respectively, to an implicit parametrization of the plane projected by each projector column and row in the common coordinate system.

5.4 Reconstruction

The decoded set of camera and projector correspondences can be used to reconstruct a 3D point cloud. Several reconstruction schemes can be implemented using the sequences described in Section 5.1. The projector column correspondences can be used to reconstruct a point cloud using ray-plane triangulation. A second point cloud can be reconstructed using the projector row correspondences. Finally, the projector pixel to camera pixel correspondences can be used to reconstruct the point cloud using ray-ray

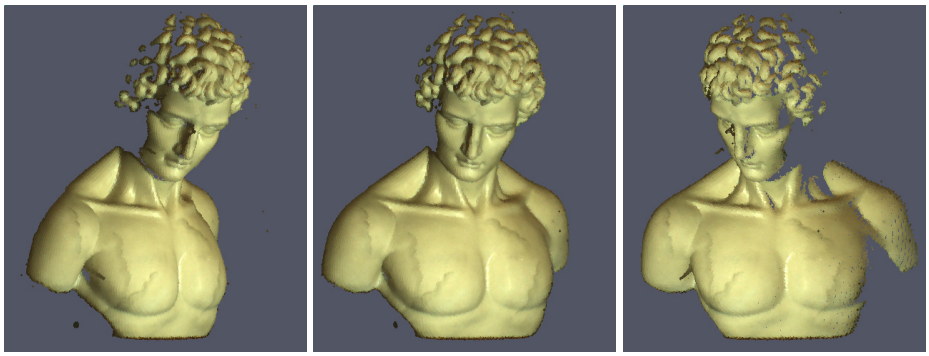


Figure 5.6: Gray code reconstruction results for first man sequence.

triangulation (i.e., by finding the closest point to the optical rays defined by the projector and camera pixels). A simple per-point RGB color can be assigned by sampling the color of the all-white camera image for each 3D point. Reconstruction artifacts can be further reduced by comparing the reconstruction produced by each of these schemes.

We have provided our own implementation of the reconstruction equations in the included m-file `slReconstruct`. This function can be used, together with the previously described m-files, to reconstruct a 3D point cloud for any of the provided test sequences. Furthermore, VRML files are also provided for each data set, containing a single indexed face set with an empty coordIndex array and a per-vertex color.

5.5 Post-processing and Visualization

As with the swept-plane scanner, the structured light scanner produces a colored 3D point cloud. Only points that are both imaged by a camera and illuminated by the projector can be reconstructed. As a result, a complete 3D model of an object would typically require merging multiple scans obtained by moving the scanning apparatus or object (e.g., by using a turntable). These issues are considered in Chapter 6. We encourage the reader to implement their own solution so that measurements from multiple cameras, projectors, and 3D point clouds can be merged. Typical results produced by or reference implementation are shown in Figure 5.6, with additional results shown in Figures 5.7–5.10.

Figure 5.7: Reconstruction of the chiquita Gray code sequence.

Figure 5.8: Reconstruction of the schooner Gray code sequence.

Figure 5.9: Reconstruction of the urn Gray code sequence.

Figure 5.10: Reconstruction of the drummer Gray code sequence.

Chapter 6

Surfaces from Point Clouds

The objects scanned in the previous examples are solid, with a well-defined boundary surface separating the inside from the outside. Since computers have a finite amount of memory and operations need to be completed in a finite number of steps, algorithms can only be designed to manipulate surfaces described by a finite number of parameters. Perhaps the simplest surface representation with a finite number of parameters is produced by a finite sampling scheme, where a process systematically chooses a set of points lying on the surface.

The triangulation-based 3D scanners described in previous chapters produce such a finite sampling scheme. The so-called *point cloud*, a dense collection of surface samples, has become a popular representation in computer graphics. However, since point clouds do not constitute surfaces, they cannot be used to determine which 3D points are inside or outside of the solid object. For many applications, being able to make such a determination is critical. For example, without closed bounded surfaces, volumes cannot be measured. Therefore, it is important to construct so-called *watertight* surfaces from point clouds. In this chapter we consider these issues.

6.1 Representation and Visualization of Point Clouds

In addition to the 3D point locations, the 3D scanning methods described in previous chapters are often able to estimate a color per point, as well as a surface normal vector. Some methods are able to measure both color and surface normal, and some are able to estimate other parameters which can be used to describe more complex material properties used to generate complex renderings. In all these cases the data structure used to represent

a point cloud in memory is a simple array. A minimum of three values per point are needed to represent the point locations. Colors may require one to three more values per point, and normals vectors three additional values per point. Other properties may require more values, but in general it is the same number of parameters per point that need to be stored. If M is the number of parameters per point and N is the number of points, then point cloud can be represented in memory using an array of length NM.

6.1.1 File Formats

Storing and retrieving arrays from files is relatively simple, and storing the raw data either in ASCII format or in binary format is a valid solution to the problem. However, these solutions may be incompatible with many software packages. We want to mention two standards which have support for storing point clouds with some auxiliary attributes.

Storing Point Clouds as VRML Files

The Virtual Reality Modeling Language (VRML) is an ISO standard published in 1997. A VRML file describes a scene graph comprising a variety of nodes. Among geometry nodes, `PointSet` and `IndexedFaceSet` are used to store point clouds. The `PointSet` node was designed to store point clouds, but in addition to the 3D coordinates of each point, only colors can be stored. No other attributes can be stored in this node. In particular, normal vectors cannot be recorded. This is a significant limitation, since normal vectors are important both for rendering point clouds and for reconstructing watertight surfaces from point clouds.

The `IndexedFaceSet` node was designed to store polygon meshes with colors, normals, and/or texture coordinates. In addition to vertex coordinates, colors and normal vectors can be stored bound to vertices. Even though the `IndexedFaceSet` node was not designed to represent point clouds, the standard allows for this node to have vertex coordinates and properties such as colors and/or normals per vertex, but no faces. The standard does not specify how such a node should be rendered in a VRML browser, but since they constitute valid VRML files, they can be used to store point clouds.

The SFL File Format

The SFL file format was introduced with Pointshop3D [ZPKG02] to provide a versatile file format to import and export point clouds with color, normal vectors, and a radius per vertex describing the local sampling density. A SFL file is encoded in binary and features an extensible set of surfel attributes, data compression, upward and downward compatibility, and transparent conversion of surfel attributes, coordinate systems, and color spaces. Pointshop3D is a software system for interactive editing of point-based surfaces, developed at the Computer Graphics Lab at ETH Zurich.

6.1.2 Visualization

A well-established technique to render dense point clouds is point splatting. Each point is regarded as an oriented disk in 3D, with the orientation determined by the surface normal evaluated at each point, and the radius of the disk usually stored as an additional parameter per vertex. As a result, each point is rendered as an ellipse. The color is determined by the color stored with the point, the direction of the normal vector, and the illumination model. The radii are chosen so that the ellipses overlap, resulting in the perception of a continuous surface being rendered.

6.2 Merging Point Clouds

The triangulation-based 3D scanning methods described in previous chapters are able to produce dense point clouds. However, due to visibility constraints these point clouds may have large gaps without samples. In order for a surface point to be reconstructed, it has to be illuminated by a projector, and visible by a camera. In addition, the projected patterns needs to illuminate the surface transversely for the camera to be able to capture a sufficient amount of reflected light. In particular, only points on the front-facing side of the object can be reconstructed (i.e., on the same side as the projector and camera). Some methods to overcome these limitations are discussed in Chapter 7. However, to produce a complete representation, multiple scans taken from various points of view must be integrated to produce a point cloud with sufficient sampling density over the whole visible surface of the object being scanned.

6.2.1 Computing Rigid Body Matching Transformations

The main challenge to merging multiple scans is that each scan is produced with respect to a different coordinate system. As a result, the rigid body transformation needed to register one scan with another must be estimated. In some cases the object is moved with respect to the scanner under computer control. In those cases the transformations needed to register the scans are known within a certain level of accuracy. This is the case when the object is placed on a computer-controlled turntable or linear translation stage. However, when the object is repositioned by hand, the matching transformations are not known and need to be estimated from point correspondences.

We now consider the problem of computing the rigid body transformation $q = Rp + T$ to align two shapes from two sets of N points, $\{p_1, \ldots, p_N\}$ and $\{q_1, \ldots, q_N\}$. That is, we are looking for a rotation matrix R and a translation vector T so that

$$q_1 = Rp_1 + T \quad \ldots \quad q_N = Rp_N + T \ .$$

The two sets of points can be chosen interactively or automatically. In either case, being able to compute the matching transformation in closed form is a fundamental operation.

This registration problem is, in general, not solvable due to measurement errors. A common approach in such a case is to seek a least-squares solution. In this case, we desire a closed-form solution for minimizing the mean squared error

$$\phi(R, T) = \frac{1}{N} \sum_{i=1}^{N} \|Rp_i + T - q_i\|^2 \ , \tag{6.1}$$

over all rotation matrices R and translation vectors T. This yields a quadratic function of 12 components in R and T; however, since R is restricted to be a valid rotation matrix, there exist additional constraints on R. Since the variable T is unconstrained, a closed-form solution for T, as a function of R, can be found by solving the linear system of equations resulting from differentiating the previous expression with respect to T.

$$\frac{1}{2}\frac{\partial \phi}{\partial T} = \frac{1}{N} \sum_{i=1}^{N} (Rp_i + T - q_i) \Rightarrow T = \overline{q} - R\overline{p} = 0$$

In this expression $\overline{p}$ and $\overline{q}$ are the geometric centroids of the two sets of matching points, given by

$$\overline{p} = \left(\frac{1}{N} \sum_{i=1}^{N} p_i \right) \qquad \overline{q} = \left(\frac{1}{N} \sum_{i=1}^{N} q_i \right) .$$

Substituting for T in Equation 6.1, we obtain the following equivalent error function which depends only on R.

$$\psi(R) = \frac{1}{N} \sum_{i=1}^{N} \| R(p_i - \overline{p}) - (q_i - \overline{q}) \|^2 \tag{6.2}$$

If we expand this expression we obtain

$$\psi(R) = \frac{1}{N} \sum_{i=1}^{N} \| p_i - \overline{p} \|^2 - \frac{2}{N} \sum_{i=1}^{N} (q_i - \overline{q})^t R(p_i - \overline{p}) + \frac{1}{N} \sum_{i=1}^{N} \| q_i - \overline{q} \|^2 ,$$

since $\|Rv\|^2 = \|v\|^2$ for any vector v. As the first and last terms do not depend on R, maximizing this expression is equivalent to maximizing

$$\eta(R) = \frac{1}{N} \sum_{i=1}^{N} (q_i - \overline{q})^t R(p_i - \overline{p}) = \text{trace}(RM) ,$$

where M is the 3×3 matrix

$$M = \frac{1}{N} \sum_{i=1}^{N} (p_i - \overline{p})(q_i - \overline{q})^t .$$

Recall that, for any pair of matrices A and B of the same dimensions, $\text{trace}(A^t B) = \text{trace}(B A^t)$. We now consider the singular value decomposition (SVD) $M = U \Delta V^t$, where U and V are orthogonal 3×3 matrices, and Δ is a diagonal 3×3 matrix with elements $\delta_1 \geq \delta_2 \geq \delta_3 \geq 0$. Substituting, we find

$$\text{trace}(RM) = \text{trace}(RU\Delta V^t) = \text{trace}((V^t RU)\Delta) = \text{trace}(W\Delta) ,$$

where $W = V^t RU$ is orthogonal. If we expand this expression, we obtain

$$\text{trace}(W\Delta) = w_{11}\delta_1 + w_{22}\delta_2 + w_{33}\delta_3 \leq \delta_1 + \delta_2 + \delta_3 ,$$

where $W = (w_{ij})$. The last inequality is true because the components of an orthogonal matrix cannot be larger than one. Note that the last inequality

is an equality only if $w_{11} = w_{22} = w_{33} = 1$, which is only the case when $W = I$ (the identity matrix). It follows that if V^tU is a rotation matrix, then $R = V^tU$ is the minimizer of our original problem. The matrix V^tU is an orthogonal matrix, but it may not have a negative determinant. In that case, an upper bound for trace$(W\Delta)$, with W restricted to have a negative determinant, is achieved for $W = J$, where

$$J = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} .$$

In this case it follows that the solution to our problem is $R = V^tJU$.

6.2.2 The Iterative Closest Point (ICP) Algorithm

The Iterative Closest Point (ICP) is an algorithm employed to match two surface representations, such as points clouds or polygon meshes. This matching algorithm is used to reconstruct 3D surfaces by registering and merging multiple scans. The algorithm is straightforward and can be implemented in real-time. ICP iteratively estimates the transformation (i.e., translation and rotation) between two geometric data sets. The algorithm takes as input two data sets, an initial estimate for the transformation, and an additional criterion for stopping the iterations. The output is an improved estimate of the matching transformation. The algorithm comprises the following steps.

1. Select points from the first shape.
2. Associate points, by nearest neighbor, with those in the second shape.
3. Estimate the closed-form matching transformation using the method derived in the previous section.
4. Transform the points using the estimated parameters.
5. Repeat previous steps until the stopping criterion is met.

The algorithm can be generalized to solve the problem of registering multiple scans. Each scan has an associated rigid body transformation which will register it with respect to the rest of the scans, regarded as a single rigid object. An additional external loop must be added to the previous steps to pick one transformation to be optimized with each pass, while the others are kept constant—either going through each of the scans in sequence, or randomizing the choice.

6.3 Surface Reconstruction from Point Clouds

Watertight surfaces partition space into two disconnected regions so that every line segment joining a point in one region to a point in the other must cross the dividing surface. In this section we discuss methods to reconstruct watertight surfaces from point clouds.

6.3.1 Continuous Surfaces

In mathematics surfaces are represented in parametric or implicit form. A parametric surface $S = \{x(u) : u \in U\}$ is defined by a function $x : U \to \mathbb{R}^3$ on an open subset U of the plane. An implicit surface is defined as a level set $S = \{p \in \mathbb{R}^3 : f(p) = \lambda\}$ of a continuous function $f : V \to \mathbb{R}$, where V is an open subset in 3D. These functions are most often smooth or piecewise smooth. Implicit surfaces are called *watertight* because they partition space into the two disconnected sets of points, one where $f(p) > \lambda$ and a second where $f(p) < \lambda$. Since the function f is continuous, every line segment joining a point in one region to a point in the other must cross the dividing surface. When the boundary surface of a solid object is described by an implicit equation, one of these two sets describes the inside of the object, and the other one the outside. Since the implicit function can be evaluated at any point in 3D space, it is also referred to as a scalar field. On the other hand, parametric surfaces may or may not be watertight. In general, it is difficult to determine whether a parametric surface is watertight or not. In addition, implicit surfaces are preferred in many applications, such as reverse engineering and interactive shape design, because they bound a solid object which can be manufactured; for example, using rapid prototyping technologies or numerically-controlled machine tools, such representations can define objects of arbitrary topology. As a result, we focus our remaining discussion on implicit surfaces.

6.3.2 Discrete Surfaces

A discrete surface is defined by a finite number of parameters. We only consider here polygon meshes, and in particular those polygon meshes representable as `IndexedFaceSet` nodes in VRML files. Polygon meshes are composed of *geometry* and topological *connectivity*. The geometry includes vertex coordinates, normal vectors, and colors (and possibly texture coordinates). The connectivity is represented in various ways. A popular representation used in many isosurface algorithms is the *polygon soup*,

where polygon faces are represented as loops of vertex coordinate vectors. If two or more faces share a vertex, the vertex coordinates are repeated as many times as needed. Another popular representation used in isosurface algorithms is the `IndexedFaceSet` (IFS), describing polygon meshes with simply-connected faces. In this representation the geometry is stored as arrays of floating point numbers. In these notes we are primarily concerned with the array `coord` of vertex coordinates, and to a lesser degree with the array `normal` of face normals. The connectivity is described by the total number V of vertices, and F faces, which are stored in the `coordIndex` array as a sequence of loops of vertex indices, demarcated by values of -1.

6.3.3 Isosurfaces

An isosurface is a polygonal mesh surface representation produced by an isosurface algorithm. An isosurface algorithm constructs a polygonal mesh approximation of a smooth implicit surface $S = \{x : f(x) = 0\}$ within a bounded three-dimensional volume, from samples of a defining function $f(x)$ evaluated on the vertices of a volumetric grid. Marching Cubes [LC87] and related algorithms operate on function values provided at the vertices of hexahedral grids. Another family of isosurface algorithms operate on functions evaluated at the vertices of tetrahedral grids [DK91]. Usually, no additional information about the function is provided, and various interpolation schemes are used to evaluate the function within grid cells, if necessary. The most natural interpolation scheme for tetrahedral meshes is linear interpolation, which we also adopt here.

6.3.4 Isosurface Construction Algorithms

An isosurface algorithm producing a polygon soup output must solve three key problems: (1) determining the quantity and location of isosurface vertices within each cell, (2) determining how these vertices are connected forming isosurface faces, and (3) determining globally consistent face orientations. For isosurface algorithms producing IFS output, there is a fourth problem to solve: identifying isosurface vertices lying on vertices and edges of the volumetric grid. For many visualization applications, the polygon soup representation is sufficient and acceptable, despite the storage overhead. Isosurface vertices lying on vertices and edges of the volumetric grid are independently generated multiple times. The main advantage of this approach is that it is highly parallelizable. But, since most of these boundary vertices are represented at least twice, it is not a compact representation.

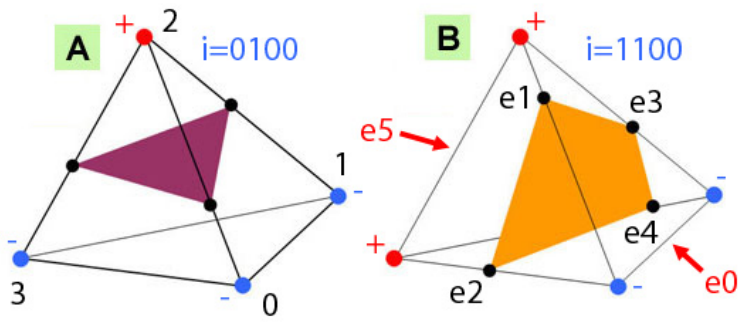


Figure 6.1: In isosurface algorithms, the sign of the function at the grid vertices determines the topology and connectivity of the output polygonal mesh within each tetrahedron. Mesh vertices are located on grid edges where the function changes sign.

Researchers have proposed various solutions and design decisions (e.g., cell types, adaptive grids, topological complexity, interpolant order) to address these four problems. The well-known Marching Cubes (MC) algorithm uses a fixed hexahedral grid (i.e., cube cells) with linear interpolation to find zero-crossings along the edges of the grid. These are the vertices of the isosurface mesh. Second, polygonal faces are added connecting these vertices using a table. The crucial observation made with MC is that the possible connectivity of triangles in a cell can be computed independently of the function samples and stored in a table. Out-of-core extensions, where sequential layers of the volume are processed one at a time, are straightforward.

Similar tetrahedral-based algorithms [DK91, GH95, TPG99], dubbed Marching Tetrahedra (MT), have also been developed (again using linear interpolation). Although the cell is simpler, MT requires maintaining a tetrahedral sampling grid. Out-of-core extensions require presorted traversal schemes, such as in [CS97]. For an unsorted tetrahedral grid, hash tables are used to save and retrieve vertices lying on edges of the volumetric grid. As an example of an isosurface algorithm, we discuss MT in more detail.

Marching Tetrahedra

MT operates on the following input data: (1) a tetrahedral grid and (2) one piecewise linear function $f(x)$, defined by its values at the grid vertices. Within the tetrahedron with vertices $x_0, x_1, x_2, x_3 \in \mathbb{R}^3$, the func-

i	$(i_3 i_2 i_1 i_0)$	face
0	0000	[-1]
1	0001	[2,1,0,-1]
2	0010	[0,3,4,-1]
3	0011	[1,3,4,2,-1]
4	0100	[1,5,3,-1]
5	0101	[0,2,5,3,-1]
6	0110	[0,3,5,4,-1]
7	0111	[1,5,2,-1]
8	1000	[2,5,1,-1]
9	1001	[4,5,3,0,-1]
10	1010	[3,5,2,0,-1]
11	1011	[3,5,1,-1]
12	1100	[2,4,3,1,-1]
13	1101	[4,3,0,-1]
14	1110	[0,1,2,-1]
15	1111	[-1]

e	edge
0	(0,1)
1	(0,2)
2	(0,3)
3	(1,2)
4	(1,3)
5	(2,3)

Table 6.1: Look-up tables for tetrahedral mesh isosurface evaluation. Note that consistent face orientation is encoded within the table. Indices stored in the first table reference tetrahedron edges, as indicated by the second table of vertex pairs (and further illustrated in Figure 6.1). In this case, only edge indices $\{1, 2, 3, 4\}$ have associated isosurface vertex coordinates, which are shared with neighboring cells.

tion is linear and can be described in terms of the barycentric coordinates $b = (b_0, b_1, b_2, b_3)^t$ of an arbitrary internal point $x = b_0\, x_0 + b_1\, x_1 + b_2\, x_2 + b_3\, x_3$ with respect to the four vertices: $f(x) = b_0\, f(x_0) + b_1\, f(x_1) + b_2\, f(x_2) + b_3\, f(x_3)$, where $b_0, b_1, b_2, b_3 \geq 0$ and $b_0 + b_1 + b_2 + b_3 = 1$. As illustrated in Figure 6.1, the sign of the function at the four grid vertices determines the connectivity (e.g., triangle, quadrilateral, or empty) of the output polygonal mesh within each tetrahedron. There are actually $16 = 2^4$ cases, which modulo symmetries and sign changes reduce to only three. Each grid edge, whose end vertex values change sign, corresponds to an isosurface mesh vertex. The exact location of the vertex along the edge is determined by linear interpolation from the actual function values, but note that the 16 cases can be precomputed and stored in a table indexed by a 4-bit integer $i = (i_3 i_2 i_1 i_0)$, where $i_j = 1$ if $f(x_j) > 0$ and $i_j = 0$, if $f(x_j) < 0$. The full table is shown in Table 6.1. The cases $f(x_j) = 0$ are singular and require special treatment. For example, the index is $i = (0100) = 4$ for Figure 6.1(a),

and $i = (1100) = 12$ for Figure 6.1(b). Orientation for the isosurface faces, consistent with the orientation of the containing tetrahedron, can be obtained from connectivity alone (and are encoded in the look-up table as shown in Table 6.1). For IFS output it is also necessary to stitch vertices as described above.

Algorithms to polygonize implicit surfaces [Blo88], where the implicit functions are provided in analytic form, are closely related to isosurface algorithms. For example, Bloomenthal and Ferguson [BF95] extract non-manifold isosurfaces produced from trimming implicits and parameterics using a tetrahedral isosurface algorithm. [WvO96] polygonize boolean combinations of skeletal implicits (Boolean Compound Soft Objects), applying an iterative solver and face subdivision for placing vertices along feature edges and points. Suffern and Balsys [SB03] present an algorithm to polygonize surfaces defined by two implicit functions provided in analytic form; this same algorithm can compute bi-iso-lines of pairs of implicits for rendering.

Isosurface Algorithms on Hexahedral Grids

An isosurface algorithm constructs a polygon mesh approximation of a level set of a scalar function defined in a finite 3D volume. The function $f(p)$ is usually specified by its values $f_\alpha = f(p_\alpha)$ on a regular grid of three dimensional points

$$G = \{p_\alpha : \alpha = (\alpha_0, \alpha_1, \alpha_2) \in [\![n_0]\!] \times [\![n_1]\!] \times [\![n_2]\!]\} \ ,$$

where $[\![n_j]\!] = \{0, \ldots, n_j - 1\}$, and by a method to interpolate in between these values. The surface is usually represented as a polygon mesh, and is specified by its *isovalue* f_0. Furthermore, the interpolation scheme is assumed to be linear along the edges of the grid, so that the isosurface cuts each edge in no more than one point. If p_α and p_β are grid points connected by an edge, and $f_\alpha > f_0 > f_\beta$, the location of the point $p_{\alpha\beta}$ where the isosurface intersects the edge is

$$p_{\alpha\beta} = \frac{f_\alpha - f_0}{f_\alpha - f_\beta}\, p_\beta + \frac{f_\beta - f_0}{f_\beta - f_\alpha}\, p_\alpha \ . \tag{6.3}$$

Marching Cubes

One of the most popular isosurface extraction algorithms is *Marching Cubes* [LC87]. In this algorithm the points defined by the intersection of the isosurface with the edges of the grid are the vertices of the polygon mesh.

These vertices are connected forming polygon faces according to the following procedure. Each set of eight neighboring grid points define a small cube called a *cell*

$$C_\alpha = \{p_{\alpha+\beta} : \beta \in \{0,1\}^3\}.$$

Since the function value associated with each of the eight corners of a cell may be either above or below the isovalue (isovalues equal to grid function values are called singular and should be avoided), there are $2^8 = 256$ possible configurations. A polygonization of the vertices within each cell, for each one of these configurations, is stored in a static look-up table. When symmetries are taken into account, the size of the table can be reduced significantly.

6.3.5 Algorithms to Fit Implicit Surfaces to Point Clouds

Let U be a relatively open and simply-connected subset of $\mathbb{R}^3$, and $f : U \to \mathbb{R}$ a smooth function. The gradient ∇f is a vector field defined on U. Given an *oriented point cloud*, i.e., a finite set $\mathcal{D}$ of point-vector pairs (p, n), where p is an interior point of U, and n is a unit length 3D vector, the problem is to find a smooth function f so that $f(p) \approx 0$ and $\nabla(p) \approx n$ for every oriented point (p, n) in the data set $\mathcal{D}$. We call the zero iso-level set of such a function $\{p : f(p) = 0\}$ a *surface fit*, or *surface reconstruction*, for the data set $\mathcal{D}$.

We are particularly interested in fitting isosurfaces to oriented point points. For the sake of simplicity, we assume that the domain is the unit cube $U = [0,1]^3$, the typical domain of an isosurface defined on an hexahedral mesh, and the isolevel is zero, i.e., the isosurface to be fitted to the data points is $\{p : f(p) = 0\}$, but of course, the argument applies in more general cases.

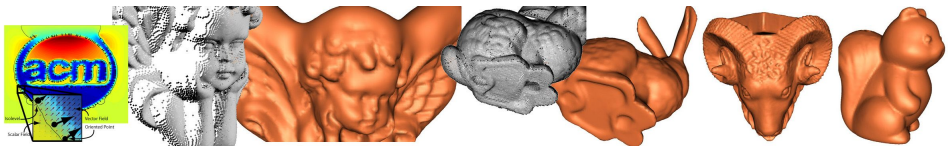


Figure 6.2: Early results of Vector Field Isosurface reconstruction from oriented point clouds introduced in [ST05].

Figure 6.2 shows results of surface reconstruction from an oriented point cloud using the simple variational formulation presented in [ST05], where oriented data points are regarded as samples of the gradient vector field of

an implicit function, which is estimated by minimizing this energy function

$$E_1(f) = \sum_{i=1}^{m} f(p_i)^2 + \lambda_1 \sum_{i=1}^{m} \|\nabla f(p_i) - n_i\|^2 + \lambda_2 \int_V \|Hf(x)\|^2 \, dx \;, \quad (6.4)$$

where $f(x)$ is the implicit function being estimated, $\nabla f(x)$ is the gradient of f, $Hf(x)$ is the Hessian of $f(x)$, $(p_1, n_1), \ldots, (p_m, n_m)$ are point-normal data pairs, V is a bounding volume, and λ_1 and λ_2 are regularization parameters. Minimizing this energy requires the solution of a simple large and sparse least squares problem. The result is usually unique modulo an additive constant. Given that, for rendering or post-processing, isosurfaces are extracted from scalar functions defined over regular grids (e.g., via Marching Cubes), it is worth exploring representations of implicit functions defined as a regular scalar fields. Finite difference discretization is used in [ST05], with the volume integral resulting in a sum of gradient differences over the edges of the regular grid, yet Equation 6.4 can be discretized in many other ways.

Chapter 7

Applications and Emerging Trends

Previous chapters outlined the basics of building custom 3D scanners. In this chapter we turn our attention to late-breaking work, promising directions for future research, and a summary of recent projects motivated by the course material. We hope course attendees will be inspired to implement and extend some of these more advanced systems, using the basic mathematics, software, and methodologies we have presented up until this point.

7.1 Extending Swept-Planes and Structured Light

This course was previously taught by the organizers at Brown University in 2007 and 2009. On-line archives, complementing these course notes, are available at http://mesh.brown.edu/3dpgp-2007 and http://mesh.brown.edu/3dpgp-2009. In this section we briefly review two course projects developed by students. The first project can be viewed as a direct extension of 3D slit scanning, similar to the device and concepts presented in Chapter 4. Rather than using a single camera, this project explores the benefits and limitations of using a stereoscopic camera in tandem with laser striping. The second project can be viewed as a direct extension of structured lighting, in fact utilizing the software that eventually led to that presented in Chapter 5. Through a combination of planar mirrors and a Fresnel lens, a novel imaging condition is achieved allowing a single digital projector and camera to recover a complete 3D object model without moving parts. We hope to add more projects to the updated course notes as we hear from attendees about their own results.

7.1.1 3D Slit Scanning with Planar Constraints

Leotta et al. [LVT08] propose "3D slit scanning with planar constraints" as a novel 3D point reconstruction algorithm for multi-view swept-plane scanners. A planarity constraint is proposed, based on the fact that all observed points on a projected stripe must lie on the same 3D plane. The plane linearly parameterizes a homography [HZ04] between any pair of images of the laser stripe, which can be recovered from point correspondences derived from epipolar geometry [SCD*06]. This planarity constraint reduces reconstruction outliers and allows for the reconstruction of points seen in only one view.

As shown in Figure 7.1, a catadioptric stereo rig is constructed to remove artifacts from camera synchronization errors and non-uniform projection. The same physical setup was originally suggested by Davis and Chen [DC01]. It maintains many of the desirable traits of other laser range scanners while eliminating several actuated components (e.g., the translation and rotation stages in Figure 1.2), thereby reducing calibration complexity and increasing maintainability and scan repeatability. Tracing backward from the camera, the optical rays first encounter a pair of primary mirrors forming a "V" shape. The rays from the left half of the image are reflected to the left, and those from the right half are reflected to the right. Next, the rays on each side encounter a secondary mirror that reflects them back toward the center and forward. The viewing volumes of the left and right sides of the image intersect near the target object. Each half of the resulting image may be considered as a separate camera for image processing. The standard camera calibration techniques for determining camera position and orientation still apply to each half.

Similar to Chapter 4, a user scans an object by manually manipulating a hand-held laser line projector. Visual feedback is provided at interactive rates in the form of incremental 3D reconstructions, allowing the user to sweep the line across any unscanned regions. Once the plane of light has been estimated, there are several ways to reconstruct the 3D location of the points. First, consider the non-singular case when a unique plane of light can be determined. If a point is visible from only one camera (due to occlusion or indeterminate correspondence), it can still be reconstructed by ray-plane intersection. For points visible in both views, it is beneficial to use the data from both views. One approach is to triangulate the points using ray-ray intersection. This is the approach taken by Davis and Chen [DC01]. While both views are used, the constraint that the resulting 3D point lies on the laser stripe plane is not strictly enforced.

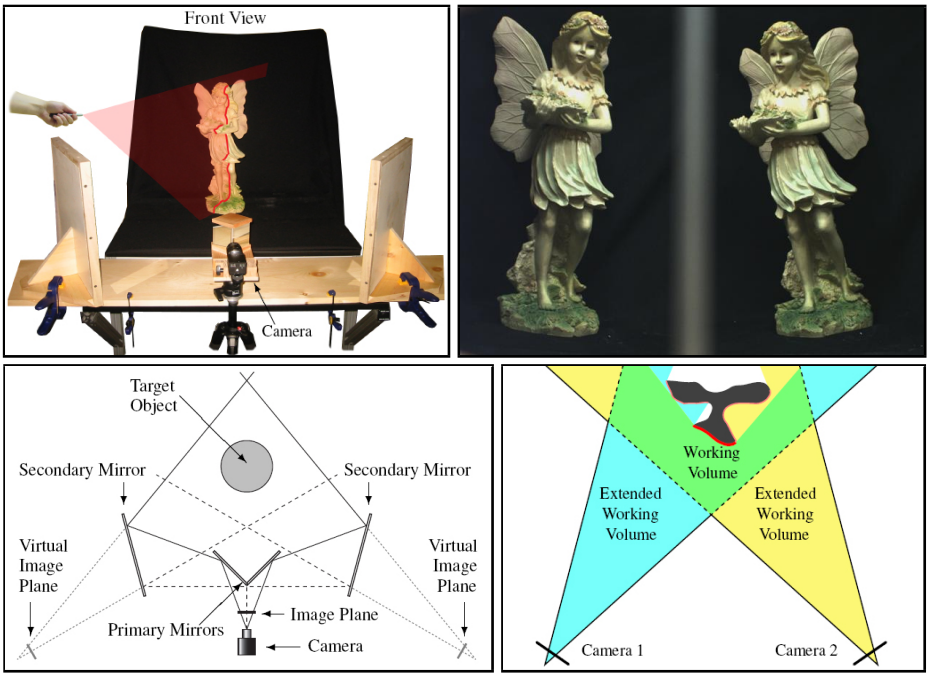
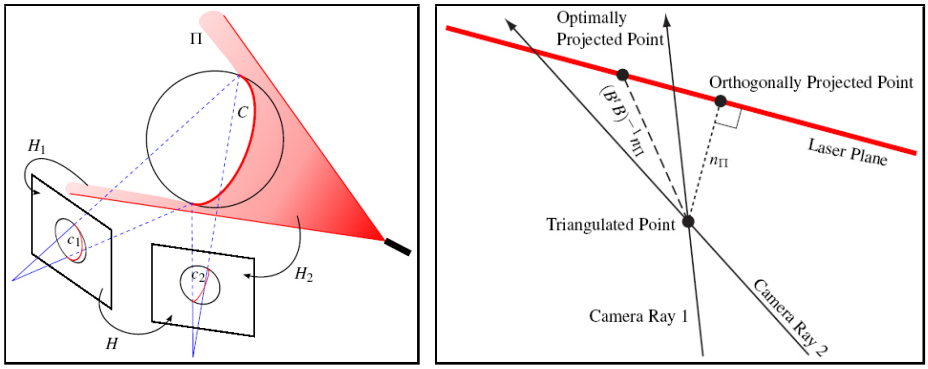


Figure 7.1: 3D slit scanning with planar constraints. (Top left) The catadioptric scanning rig. (Top right) A sample image. (Bottom left) Diagram of the scanning system. Note that the camera captures a stereo image, each half originating from a virtual camera produced by mirror reflection. (Bottom right) Working volume and scannable surfaces of a T-shaped object. Note that the working volume is the union of the virtual camera pair, excluding occluded regions. (Figure from [LVT08].)

Leotta et al. [LVT08] examine how such a planarity constraint can enhance the reconstruction. Their first approach involves triangulating a point (using ray-ray intersection) and then projecting it onto the closest point on the corresponding 3D laser plane. While such an approach combines the stability of triangulation with the constraint of planarity, there is no guarantee that the projected point is the optimal location on the plane. In their second approach, they compute an optimal triangulation constrained to the plane. The two projection approaches are compared in Figure 7.2. We refer the reader to the paper for additional technical details on the derivation of the optimally projected point. Illustrative results are shown in Figure 7.3. Note the benefit of adding points seen in only one view, as well as the result of applying optimal planar projection.



(a) laser plane to image homographies (b) projection onto the laser plane

Figure 7.2: (a) Homographies between the laser plane and image planes. (b) A 2D view of triangulation and both orthogonal and optimal projection onto the laser plane. (Figure from [LVT08].)

(a) catadioptric stereo view (b) triangulated (c) optimal (d) all points

Figure 7.3: Reconstruction results. The catadioptric image (a) and output point clouds using triangulation (b) and optimal planar projection (c). Note that points in (c) are reconstructed from both views, whereas (d) shows the additional of points seen from only one view. (Figure from [LVT08].)

7.1.2 Surround Structured Lighting

Lanman et al. [LCT07] propose "surround structured lighting" as a novel modification of a traditional single projector-camera structured light system that allows full 360° surface reconstructions, without requiring turntables or multiple scans. As shown in Figure 7.4, the basic concept is to illuminate the object from all directions with a structured pattern consisting of horizontal planes of light, while imaging the object from multiple views using a single camera and mirrors. A key benefit of this design is to ensure that each point on the object surface can be assigned an unambiguous Gray code sequence, despite the possibility of being illuminated from multiple directions.

Traditional structured light projectors, for example those using Gray code sequences, cannot be used to simultaneously illuminate an object from all sides (using more than one projector) due to interference. If such a configuration was used, then there is a high probability that certain points would be illuminated by multiple projectors. In such circumstances, multiple Gray codes would interfere, resulting in erroneous reconstruction due to decoding errors. Rather than using multiple projectors (each with a single center of projection), the proposed "surround structured lighting" system uses a single *orthographic* projector and a pair of planar mirrors.

As shown from above and in profile in Figure 7.4, the key components of the proposed scanning system are an orthographic projector, two planar mirrors aligned such that their normal vectors are contained within the plane of light created by each projector row, and a single high-resolution digital camera. If any structured light pattern consisting of horizontal binary stripes is implemented, then the object can be fully illuminated on all sides due to direct and reflected projected light. Furthermore, if the camera's field of view contains the object and both mirrors, then it will record five views of the illuminated object: one direct view, two first reflections, and two second reflections [FNdJV06]. By carefully aligning the mirrors so that individual projector rows are always reflected back upon themselves, only a single Gray code sequence will be assigned to each projector row—ensuring each vertically-space plane in the reconstruction volume receives a unique code. The full structured light pattern combined with the five views (see Figure 7.5) provides sufficient information for a nearly complete surface reconstruction from a single camera position, following methods similar to those in Chapter 5.

The required orthographic projector can be implemented using a standard off-the-shelf DLP projector and a Fresnel lens, similar to that used by

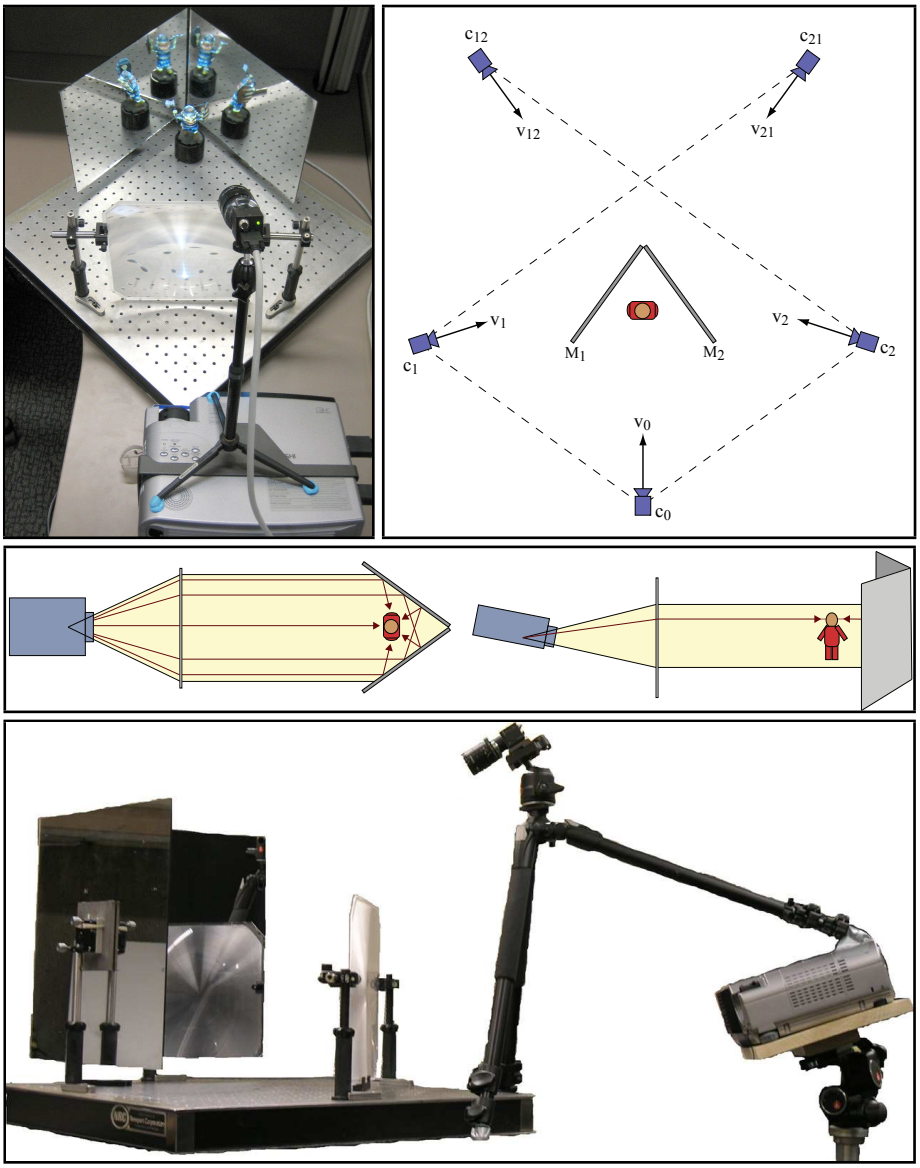


Figure 7.4: Surround structured lighting for full object scanning. (Top left) Surround structured lighting prototype. (Top right) The position of the real c_0 and virtual $\{c_1, c_2, c_{12}, c_{21}\}$ cameras with respect to mirrors $\{M_1, M_2\}$. (Middle) After reflection, multiple rays from a single projector row illuminate the object from all sides while remaining co-planar. (Bottom) Prototype, from left to right: the first-surface planar mirrors, Fresnel lens, high-resolution digital camera, and DLP projector. (Figure from [LCT07].)

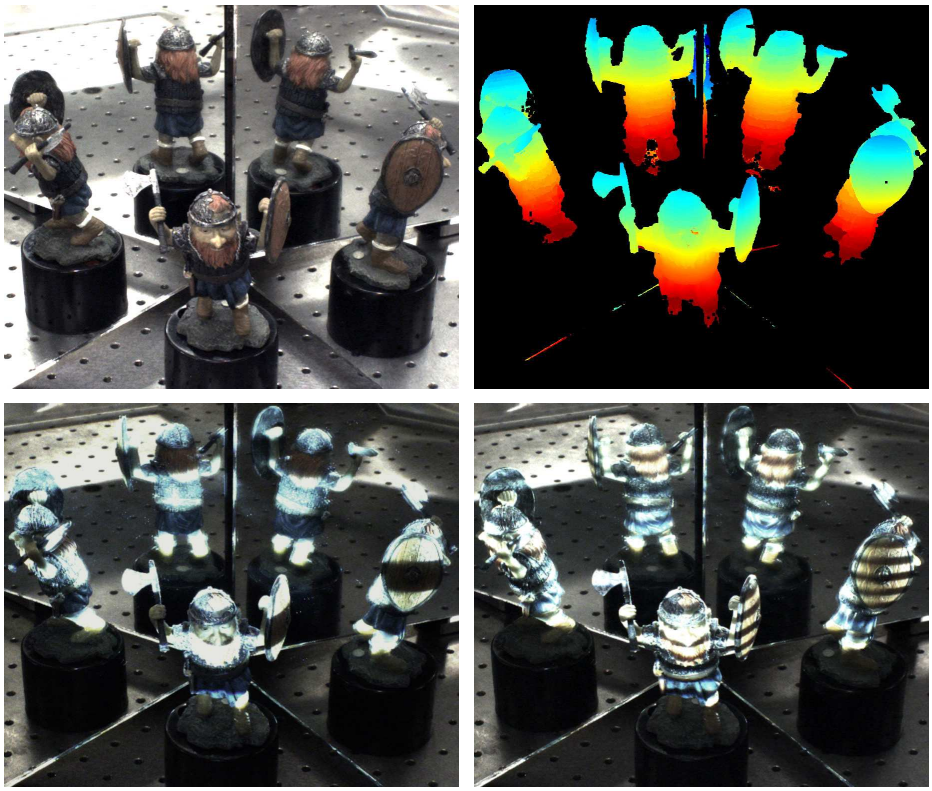


Figure 7.5: Example of an orthographic Gray code pattern and recovered projector rows. (Top left) Scene, as viewed under ambient illumination, for use in texture mapping. (Top right) Per-pixel projector row indices recovered by decoding the projected Gray code sequence (shaded by increasing index, from red to blue). (Bottom left) Fourth projected Gray code. (Bottom right) Sixth projected Gray code. (Figure from [LCT07].)

Nayar and Anand [NA06] for volumetric display. The Fresnel lens converts light rays diverging from the focal point to parallel rays and can be manufactured in large sizes, while remaining lightweight and inexpensive. Although the projector can be modeled as a pinhole projector (see Chapters 2 and 3), practically it will have a finite aperture lens and a corresponding finite depth of field. This makes conversion to a perfectly-orthographic set of rays impossible with the Fresnel lens, yet an acceptable approximation is still feasible.

The planar mirrors are positioned, following Forbes et al. [FNdJV06],

Figure 7.6: Summary of reconstruction results. From left to right: the input images used for texture mapping and four views of the 3-D point cloud recovered using the proposed method with a single camera and orthographic projector. (Figure from [LCT07].)

such that their surface normals are roughly 72° apart and perpendicular to the projected planes of light. This ensures that five equally-spaced views are created. The mirrors are mounted on gimbals with fine tuning knobs in order to facilitate precise positioning. Furthermore, first-surface mirrors are used to eliminate refraction from the protective layer of glass covering the reflective surface in lower-cost second-surface mirrors (e.g., those one would typically buy at a hardware store).

Because of the unique design of the scanning system, calibration of the multiple components is a non-trivial task. Lanman et al. [LCT07] address these issues by developing customized calibration procedures divided across three stages: (1) configuration of the orthographic projector, (2) alignment of the planar mirrors, and (3) calibration of the camera/mirror system. Key results include the use of Gray codes and homographies to calibrate the orthographic imaging system, procedures to ensure precise mechanical alignment of the scanning apparatus, and optimization techniques for calibrating catadioptic systems containing a single digital camera and one or more planar mirrors. We refer readers to the paper for additional calibration details.

Reconstruction proceeds in a similar manner to that used in conventional structured light scanners. A key modification, however, is the requirement that each of the five sub-images must be assigned to a specific real or virtual camera. Afterwards, each optical ray is intersected with its associated projector plane (corresponding to an individual orthographic projector row) in order to reconstruct a dense 3-D point cloud. Illustrative results are tabulated in Figure 7.6.

While the current prototype can only scan relatively small volumes, this system has already demonstrated practical benefits for telecollaboration applications, allowing for rapid acquisition of nearly complete object models. We hope the reader is inspired to pursue similar non-traditional optical configurations using their own 3D scanners. To this end, we also recommend reviewing related work by Epstein et al. [EGPP04], on incorporating planar mirrors with structured lighting, as well as the work of Nayar and Anand [NA06] on creating orthographic projectors using similar configurations of Fresnel lenses.

7.2 Recent Advances and Further Reading

3D scanning remains a very active area of computer graphics and vision research. While numerous commercial products are available, few achieve the ease of use, visual fidelity, and reliability of simple point-and-shoot cameras. As briefly reviewed in Chapter 1, a myriad collection of both passive and active non-contact optical metrology methods have emerged. Many have not withstood the test of time, yielding to more flexible, lower-cost alternatives. Some, like 3D slit scanning and structured lighting, have become widespread—in equal parts due to their performance, as well as their conceptual and practical accessibility.

In this section we briefly review late-breaking work and other advances that are shaping the field of optical 3D scanning. Before continuing, we encourage the reader to consider the materials from closely-related SIGGRAPH 2009 courses. Heidrich and Ihrke present *Acquisition of Optically Complex Objects and Phenomena*, discussing methods to scan problematic objects with translucent and specular materials. In a similar vein, Narasimhan et al. present *Scattering*, a course on imaging through participating media. Several additional courses focus on specialized scanning applications; Debevec et al. cover face scanning in *The Digital Emily Project: Photoreal Facial Modeling and Animation*, whereas Cain et al. discuss scanning applications in archeology and art history in *Computation & Cultural Heritage:*

Fundamentals and Applications. Finally, we recommend Gross' *Point Based Graphics-State of the Art and Recent Advances* for a further tutorial on point-based rendering.

Applying 3D scanning in practical situations requires carefully considering the complexities of the object you are measuring. These complexities are explored, to great detail, by two pioneering efforts: the Digital Michelangelo Project [LPC*00] and the Pietà Project [BRM*02]. In the former, the subsurface scattering properties of marble were considered; in the later, lightweight commercial laser scanners and photometric stereo were deployed to create a portable solution. Building low-cost, reliable systems for rapidly scanning such complex objects remains a challenging task.

Passive methods continue to evolve. Light fields [LH96, GGSC96], captured using either camera arrays [WJV*05] or specialized plenoptic cameras [AW92, NLB*05, VRA*07], record the spatial and angular variation of the irradiance passing between two planes in the world (assuming no attenuation within the medium). Vaish et al. [VLS*06] recover depth maps from light fields. Such imagery can be used to synthesize a virtual focal stack, allowing the method of Nayar and Nakagawa [NN94] to estimate shape-from-focus. Lanman et al. [LRAT08] extend light field capture to allow single-shot visual hull reconstruction of opaque objects.

The limitations of most active scanning methods are well-known. Specifically, scanning diffuse surfaces is straightforward; however, scenes that contain translucence, subsurface scattering, or strong reflections lead to artifacts and often require additional measures to be taken—often involving dappling the surface with certain Lambertian powders. Scanning such hard-to-scan items has received significant attenuation over the last few years. Hullin et al. [HFI*08] immerse transparent objects in a florescent liquid. This liquid creates an inverse image as that produced by a traditional 3D slit scanner, where the laser sheet is visible up to the point of contact. In a similar vein, tomographic methods are used to reconstruct transparent objects; first, by immersion in an index-matching liquid [TBH06], and second through the use of background-oriented Schlieren imaging [AIH*08].

In addition to scanning objects with complex material properties, active imaging is beginning to be applied to challenging environments with anisotropic participating media. For example, in typical underwater imaging conditions, visible light is strongly scattered by suspended particulates. Narasimhan et al. consider laser striping and photometric stereo in underwater imaging [NN05], as well as structured light in scattering media [NNSK08]. Such challenging environments represent the next horizon for active 3D imaging.

7.3 Conclusion

As low-cost mobile projectors enter the market, we expect students and hobbyists to begin incorporating them into their own 3D scanning systems. Such projector-camera systems have already received a great deal of attention in recent academic publications. Whether for novel human-computer interaction or ad-hoc tiled displays, consumer digital projection is set to revolutionize the way we interact with both physical and virtual assets.

This course was designed to lower the barrier of entry to novices interested in trying 3D scanning in their own projects. Through the course notes, on-line materials, and open source software, we have endeavored to eliminate the most difficult hurdles facing beginners. We encourage attendees to email the authors with questions or links to their own 3D scanning projects that draw on the course material. Revised course notes, updated software, recent publications, and similar do-it-yourself projects are maintained on the course website at `http://mesh.brown.edu/dlanman/scan3d`. We encourage you to take a look and see what your fellow attendees have built for themselves!

Bibliography

[AIH*08] ATCHESON B., IHRKE I., HEIDRICH W., TEVS A., BRADLEY D., MAGNOR M., SEIDEL H.-P.: Time-resolved 3d capture of non-stationary gas flows. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) 27*, 5 (2008), 132. 78

[AW92] ADELSON T., WANG J.: Single lens stereo with a plenoptic camera. *IEEE TPAMI 2*, 14 (1992), 99–106. 78

[BF95] BLOOMENTHAL J., FERGUSON K.: Polygonization of non-manifold implicit surfaces. In *SIGGRAPH '95: ACM SIGGRAPH 1995 papers* (1995), pp. 309–316. 66

[BK08] BRADSKI G., KAEHLER A.: *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008. 32

[Bla04] BLAIS F.: Review of 20 years of range sensor development. *Journal of Electronic Imaging 13*, 1 (2004), 231–240. 5

[Blo88] BLOOMENTHAL J.: Polygonization of Implicit Surfaces. *Computer Aided Geometric Design 5*, 4 (1988), 341–355. 66

[Bou] BOUGUET J.-Y.: Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/. 28, 40, 52

[Bou99] BOUGUET J.-Y.: *Visual methods for three-dimensional modeling*. PhD thesis, California Institute of Technology, 1999. 38, 42

[BP] BOUGUET J.-Y., PERONA P.: 3d photography on your desk. http://www.vision.caltech.edu/bouguetj/ICCV98/. 7, 35, 36, 45

[BP99] BOUGUET J.-Y., PERONA P.: 3d photography using shadows in dual-space geometry. *Int. J. Comput. Vision 35*, 2 (1999), 129–149. 35, 38, 39, 42

[BRM*02] BERNARDINI F., RUSHMEIER H., MARTIN I. M., MITTLEMAN J., TAUBIN G.: Building a digital model of michelangelo's florentine pietà. *IEEE Computer Graphics and Applications 22* (2002), 59–67. 78

[CG00] CIPOLLA R., GIBLIN P.: *Visual Motion of Curves and Surfaces*. Cambridge University Press, 2000. 4

[CMU] CMU IEEE 1394 digital camera driver, version 6.4.5. http://www.cs.cmu.edu/ iwan/1394/. 25

[Cre] CREAFORM: Handyscan 3D. http://www.creaform3d.com/en/handyscan3d/products/exascan.aspx. 6

[CS97] CHIANG Y., SILVA C. T.: I/O Optimal Isosurface Extraction. In *IEEE Visualization 1997, Conference Proceedings* (1997), pp. 293–300. 64

[CTMS03] CARRANZA J., THEOBALT C., MAGNOR M. A., SEIDEL H.-P.: Free-viewpoint video of human actors. *ACM Trans. Graph. 22*, 3 (2003), 569–577. 4

[dAST*08] DE AGUIAR E., STOLL C., THEOBALT C., AHMED N., SEIDEL H.-P., THRUN S.: Performance capture from sparse multi-view video. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (2008), pp. 1–10. 4

[DC01] DAVIS J., CHEN X.: A laser range scanner designed for minimum calibration complexity. In *Proceedings of the International Conference on 3-D Digital Imaging and Modeling (3DIM)* (2001), p. 91. 70

[Deb97] DEBEVEC P. E.: Facade: modeling and rendering architecture from photographs and the campanile model. In *SIGGRAPH '97: ACM SIGGRAPH 97 Visual Proceedings* (1997), p. 254. 4

[DK91] DOI A., KOIDE A.: An Efficient Method of Triangulating Equivalued Surfaces by Using Tetrahedral Cells. *IEICE Transactions on Communications and Electronics Information Systems E74*, 1 (Jan. 1991), 214–224. 63, 64

[Edm] EDMUND OPTICS:. http://www.edmundoptics.com. 37

[EGPP04] EPSTEIN E., GRANGER-PICHÉ M., POULIN P.: Exploiting mirrors in interactive reconstruction with structured light. In *Vision, Modeling, and Visualization 2004* (2004), pp. 125–132. 77

[Far97] FARID H.: *Range Estimation by Optical Differentiation*. PhD thesis, University of Pennsylvania, 1997. 4

[FNdJV06] FORBES K., NICOLLS F., DE JAGER G., VOIGT A.: Shape-from-silhouette with two mirrors and an uncalibrated camera. In *ECCV 2006* (2006), pp. 165–178. 73, 75

[FWM98] FERRYMAN J. M., WORRALL A. D., MAYBANK S. J.: Learning enhanced 3d models for vehicle tracking. In *BMVC* (1998), pp. 873–882. 4

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. In *SIGGRAPH '96: ACM SIGGRAPH 1996 papers* (1996), pp. 43–54. 78

[GH95] GUÉZIEC A., HUMMEL R.: Exploiting Triangulated Surface Extraction Using Tetrahedral Decomposition. *IEEE Transactions on Visualization and Computer Graphics 1*, 4 (1995). 64

[GSP06] GREENGARD A., SCHECHNER Y. Y., PIESTUN R.: Depth from diffracted rotation. *Opt. Lett. 31*, 2 (2006), 181–183. 4

[HARN06] HSU S., ACHARYA S., RAFII A., NEW R.: Performance of a time-of-flight range camera for intelligent vehicle safety applications. *Advanced Microsystems for Automotive Applications* (2006). 6

[Hec01] HECHT E.: *Optics (4th Edition)*. Addison Wesley, 2001. 4

[HFI*08] HULLIN M. B., FUCHS M., IHRKE I., SEIDEL H.-P., LENSCH H. P. A.: Fluorescent immersion range scanning. *ACM Trans. Graph. 27*, 3 (2008), 1–10. 78

[HVB*07] HERNÁNDEZ C., VOGIATZIS G., BROSTOW G. J., STENGER B., CIPOLLA R.: Non-rigid photometric stereo with colored lights. In *Proc. of the 11th IEEE Intl. Conf. on Comp. Vision (ICCV)* (2007). 7

[HZ04] HARTLEY R. I., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, 2004. 3, 26, 70

[ISM84] INOKUCHI S., SATO K., MATSUDA F.: Range imaging system for 3-d object recognition. In *Proceedings of the International Conference on Pattern Recognition* (1984), pp. 806–808. 48

[IY01] IDDAN G. J., YAHAV G.: Three-dimensional imaging in the studio and elsewhere. *Three-Dimensional Image Capture and Applications IV 4298*, 1 (2001), 48–55. 6

[KM00] KAKADIARIS I., METAXAS D.: Model-based estimation of 3d human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 12 (2000), 1453–1459. 4

[Las] LASER DESIGN INC.: Surveyor DT-2000 desktop 3D laser scanner. http://www.laserdesign.com/quick-attachments/hardware/low-res/dt-series.pdf. 6

[Lau94] LAURENTINI A.: The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE TPAMI 16*, 2 (1994), 150–162. 3

[LC87] LORENSEN W. L., CLINE H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Siggraph'87, Conference Proceedings* (1987), ACM Press, pp. 163–169. 63, 66

[LCT07] LANMAN D., CRISPELL D., TAUBIN G.: Surround structured lighting for full object scanning. In *Proceedings of the International Conference on 3-D Digital Imaging and Modeling (3DIM)* (2007), pp. 107–116. 73, 74, 75, 76

[LFDF07] LEVIN A., FERGUS R., DURAND F., FREEMAN W. T.: Image and depth from a conventional camera with a coded aperture. *ACM Trans. Graph. 26*, 3 (2007), 70. 4

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *Proc. of ACM SIGGRAPH* (1996), pp. 31–42. 78

[LN04] LAND M. F., NILSSON D.-E.: *Animal Eyes*. Oxford University Press, 2004. 2

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital michelangelo project: 3D scanning of large statues. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 131–144. 78

[LRAT08] LANMAN D., RASKAR R., AGRAWAL A., TAUBIN G.: Shield fields: modeling and capturing 3d occluders. *ACM Trans. Graph. 27*, 5 (2008), 1–10. 78

[LT] LANMAN D., TAUBIN G.: Build your own 3d scanner: 3d photography for beginners (course website). http://mesh.brown.edu/dlanman/scan3d. 37

[LVT08] LEOTTA M. J., VANDERGON A., TAUBIN G.: 3d slit scanning with planar constraints. *Computer Graphics Forum 27*, 8 (Dec. 2008), 2066–2080. 70, 71, 72

[Mat] MATHWORKS, INC: Image acquisition toolbox. http://www.mathworks.com/products/imaq/. 25, 46

[MBR*00] MATUSIK W., BUEHLER C., RASKAR R., GORTLER S. J., MCMILLAN L.: Image-based visual hulls. In *SIGGRAPH '00: ACM SIGGRAPH 2000 papers* (2000), pp. 369–374. 4

[Mit] MITSUBISHI ELECTRIC CORP.: XD300U user manual. http://www.projectorcentral.com/pdf/projector_manual_1921.pdf. 46

[MPL04] MARC R. Y., POLLEFEYS M., LI S.: Improved real-time stereo on commodity graphics hardware. In *In IEEE Workshop on Real-time 3D Sensors and Their Use* (2004). 3

[MSKS05] MA Y., SOATTO S., KOSECKA J., SASTRY S. S.: *An Invitation to 3-D Vision*. Springer, 2005. 26, 39

[NA06] NAYAR S. K., ANAND V.: *Projection Volumetric Display Using Passive Optical Scatterers*. Tech. rep., July 2006. 75, 77

[Nex] NEXTENGINE: 3D Scanner HD. https://www.nextengine.com/indexSecure.htm. 6

[NLB*05] NG R., LEVOY M., BREDIF M., DUVAL G., HOROWITZ M., HANRAHAN P.: Light field photography with a hand-held plenoptic camera. *Tech Report, Stanford University* (2005). 78

[NN94] NAYAR S. K., NAKAGAWA Y.: Shape from focus. *IEEE Trans. Pattern Anal. Mach. Intell. 16*, 8 (1994), 824–831. 4, 78

[NN05] NARASIMHAN S. G., NAYAR S.: Structured light methods for underwater imaging: light stripe scanning and photometric stereo. In *Proceedings of 2005 MTS/IEEE OCEANS* (September 2005), vol. 3, pp. 2610 – 2617. 78

[NNSK08] NARASIMHAN S. G., NAYAR S. K., SUN B., KOPPAL S. J.: Structured light in scattering media. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 courses* (2008), pp. 1–8. 78

[Opea] Open source computer vision library. http://sourceforge.net/projects/opencvlibrary/. 26, 40

[Opeb] OpenCV wiki. http://opencv.willowgarage.com/wiki/. 46

[OSS*00] ORMONEIT D., SIDENBLADH H., SIDENBLADH H., BLACK M. J., HASTIE T., FLEET D. J.: Learning and tracking human motion using functional analysis. In *IEEE Workshop on Human Modeling, Analysis and Synthesis* (2000), pp. 2–9. 4

[PA82] POSDAMER J., ALTSCHULER M.: Surface measurement by space encoded projected beam systems. *Computer Graphics and Image Processing 18* (1982), 1–17. 47

[Poia] POINT GREY RESEARCH, INC.: Grasshopper IEEE-1394b digital camera. http://www.ptgrey.com/products/grasshopper/index.asp. 26, 46

[Poib] POINT GREY RESEARCH, INC.: Using matlab with point grey cameras. http://www.ptgrey.com/support/kb/index.asp?a=4&q=218. 46

[Pol] POLHEMUS: FastSCAN. http://www.polhemus.com/?page=Scanning_Fastscan. 6

[Psy] PSYCHOPHYSICS TOOLBOX:. http://psychtoolbox.org. 31, 33, 47

[RWLB01] RASKAR R., WELCH G., LOW K.-L., BANDYOPADHYAY D.: Shader lamps: Animating real objects with image-based illumination. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag, pp. 89–102. 4

[SB03] SUFFERN K. G., BALSYS R. J.: Rendering the intersections of implicit surfaces. *IEEE Comput. Graph. Appl. 23*, 5 (2003), 70–77. 66

[SCD*06] SEITZ S., CURLESS B., DIEBEL J., SCHARSTEIN D., SZELISKI R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR 2006* (2006). 3, 70

[SH03] STARCK J., HILTON A.: Model-based multiple view reconstruction of people. In *Proceedings of the Ninth IEEE International Conference on Computer Vision* (2003), p. 915. 4

[SMP05] SVOBODA T., MARTINEC D., PAJDLA T.: A convenient multicamera self-calibration for virtual environments. *PRESENCE: Teleoperators and Virtual Environments 14*, 4 (August 2005), 407–422. 27

[SPB04] SALVI J., PAGÈS J., BATLLE J.: Pattern codification strategies in structured light systems. In *Pattern Recognition* (April 2004), vol. 37, pp. 827–849. 6, 47

[ST05] SIBLEY P. G., TAUBIN G.: Vectorfield Isosurface-based Reconstruction from Oriented points. In *SIGGRAPH'05 Sketch* (2005). 67, 68

[Sul95] SULLIVAN G.: Model-based vision for traffic scenes using the ground-plane constraint. 93–115. 4

[TBH06] TRIFONOV B., BRADLEY D., HEIDRICH W.: Tomographic reconstruction of transparent objects. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches* (2006), p. 55. 78

[TPG99] TREECE G. M., PRAGER R. W., GEE A. H.: Regularised Marching Tetrahedra: Improved Iso-Surface Extraction. *Computers and Graphics 23*, 4 (1999), 583–598. 64

[VF92] VAILLANT R., FAUGERAS O. D.: Using extremal boundaries for 3-d object modeling. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (1992), 157–173. 3

[VLS*06] VAISH V., LEVOY M., SZELISKI R., ZITNICK C. L., KANG S. B.: Reconstructing occluded surfaces using synthetic apertures: Stereo, focus and robust measures. In *Proc. IEEE Computer Vision and Pattern Recognition* (2006), pp. 2331–2338. 78

[VRA*07] VEERARAGHAVAN A., RASKAR R., AGRAWAL R., MOHAN A., TUMBLIN J.: Dappled photography: Mask enhanced cameras for heterodyned light fields and coded aperture refocusing. *ACM Trans. Graph. 26*, 3 (2007), 69. 78

[Wik] WIKIPEDIA: Gray code. http://en.wikipedia.org/wiki/Gray_code. 48

[WJV*05] WILBURN B., JOSHI N., VAISH V., TALVALA E.-V., ANTUNEZ E., BARTH A., ADAMS A., HOROWITZ M., LEVOY M.: High performance imaging using large camera arrays. *ACM Trans. Graph. 24*, 3 (2005), 765–776. 78

[WN98] WATANABE M., NAYAR S. K.: Rational filters for passive depth from defocus. *Int. J. Comput. Vision 27*, 3 (1998), 203–225. 4

[Woo89] WOODHAM R. J.: Photometric method for determining surface orientation from multiple images. 513–531. 7

[WvO96] WYVILL B., VAN OVERVELD K.: Polygonization of Implicit Surfaces with Constructive Solid Geometry. *Journal of Shape Modelling 2*, 4 (1996), 257–274. 66

[ZCS03] ZHANG L., CURLESS B., SEITZ S. M.: Spacetime stereo: Shape recovery for dynamic scenes. In *IEEE Conference on Computer Vision and Pattern Recognition* (June 2003), pp. 367–374. 39

[Zha99] ZHANG Z.: Flexible camera calibration by viewing a plane from unknown orientations. In *International Conference on Computer Vision (ICCV)* (1999). 40

[Zha00] ZHANG Z.: A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell. 22*, 11 (2000), 1330–1334. 24, 25, 27

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: an interactive system for point-based surface editing. *ACM Trans. Graph. 21*, 3 (2002), 322–329. 58