PRACTICAL SURFACE LIGHT FIELDS

Greg Coombe

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2007

Approved by:

Anselmo Lastra

Radek Grzeszczuk

Leonard McMillan

Marc Pollefeys

Gary Bishop

© 2007
Greg Coombe
ALL RIGHTS RESERVED

ABSTRACT

GREG COOMBE: Practical Surface Light Fields
(Under the direction of Anselmo Lastra)

The rendering of photorealistic surface appearance is one of the main challenges facing modern computer graphics. Image-based approaches have become increasingly important because they can capture the appearance of a wide variety of physical surfaces with complex reflectance behavior. In this dissertation, I focus on Surface Light Fields, an image-based representation of view-dependent and spatially-varying appearance.

Constructing a Surface Light Field can be a time-consuming and tedious process. The data sizes are quite large, often requiring multiple gigabytes to represent complex reflectance properties. The result can only be viewed after a lengthy post-process is complete, so it can be difficult to determine when the light field is sufficiently sampled. Often, uncertainty about the sampling density leads users to capture many more images than necessary in order to guarantee adequate coverage.

To address these problems, I present several approaches to simplify the capture of Surface Light Fields. The first is a "human-in-the-loop" interactive feedback system based on the Online SVD. As each image is captured, it is incorporated into the representation in a streaming fashion and displayed to the user. In this way, the user receives direct feedback about the capture process, and can use this feedback to improve the sampling. To avoid the problems of discretization and resampling, I used Incremental Weighted Least Squares, a subset of Radial Basis Function which allows for incremental local construction and fast rendering on graphics hardware. Lastly, I address the limitation of fixed lighting by describing a system that captures the Surface Light Field of an object under synthetic lighting.

ACKNOWLEDGMENTS

Just as there is an African proverb that states “It takes a village to raise a child,” there should be an Academic proverb that states “It takes a big department to create a dissertation.” I am extremely lucky to have spent some of the best years of my life with the people of the University of North Carolina Computer Science Department. The people that I met through my time at UNC helped in numerous ways, which they may or may not know of, and which I will never be able to repay.

Most importantly, I would like to thank my advisor Anselmo Lastra, who has been a great inspiration to me both academically and personally. It is his ability to spot interesting research ideas (and discard unfruitful ones) that led me down the fruitful research path of this dissertation. I'd also like to thank him for being willing to help out when I was inevitably working at 4am the night before the paper deadline.

During a summer internship at Intel I had the privilege of working with Radek Grzeszczuk, who encouraged me to look at GPU implementations of Surface Light Fields. This became the first component of this dissertation and lead to all of the subsequent research. This has become somewhat of a trend for Radek, who also sparked the research of Wei-Chao Chen and Karl Hillesland before me. Working with him also gave me valuable insight into the world of industry research.

I would also like to thank Fred Brooks, Mary Whitton, and the members of the EVE Group for providing a unique and nurturing scientific environment for the exchange of ideas. I gained a tremendous amount through discussions. Leonard McMillan and the students of the Data Driven Modeling course gave encouragement and critiques of the research that would eventually become the second component of this dissertation.

I'd like to thank my first roommates in Chapel Hill; Mark Harris, David Marshburn, and Samir Naik, for demonstrating how to successfully navigate the stresses of graduate school by taking time to hang out and enjoy the social life of a great college town. Joining these three were often Andrew Nashel, Yuaxin Liu, Sharif Razzaque, Stefan Sain, Andrew Zaferakis, Ajith Mascarenas, Kelly Ward, Caroline Green, David Borland, Bill Baxter, Jeff Feasel, and too many others to name individually, although they know who they are. Without them, my years spent in Chapel Hill would have been dramatically less enjoyable.

I'd like to thank Bruce Gooch, whom I met in my undergraduate studies at the University of Utah. He encouraged me to get into research by joining the Alpha_1 research group, thus starting me down the long path that lead to this dissertation. Thanks to my friends and co-authors Josh Steinhurst and Justin Hensley for numerous research discussions about graphics hardware and global illumination. For several years, not a day would pass without a discussion of an interesting problem or new research topic. If only I had enough time to work on all of them!

I'd like to thank Jan-Micheal Frahm for helping me with the third component of this dissertation, and in particular helping me track down all of the possible sources of error in the capture device, and then devising creative and novel ways to quantify this error. I'd also like to thank Mark Pollefeys, Leonard McMillan, Radek Grzeszczuk, and Gary Bishop, for help during the research proposal and oral exam.

I would also like to thank the superb staff of the Computer Science Department, especially Bil Hays, Tim Quigg, Janet Jones, and Sandra Neely, who work every day to make our lives as grad students easier. I'm grateful to NVIDIA for funding my graduate career through both their International Graduate Student Fellowship and an interesting and enjoyable summer internship.

I would like to thank my parents for always supporting me and always believing in the importance of education. This is especially true of my wife Kavita, who endured numerous late nights and small paychecks over the course of the past 6 years. One of the few things that I could offer in return was a very flexible schedule, which allowed us to travel to Spain, Canada, Germany, Mexico, and India together. I'm extremely thankful that she has chosen me as her lifetime travel partner.

To my Dad, who always made sure I put education first

TABLE OF CONTENTS

LIST OF TABLES **x**

LIST OF FIGURES **xi**

LIST OF ABBREVIATIONS **xviii**

1 Introduction **1**

1.1 Surface Light Field Capture Difficulties 2

1.2 Thesis 5

1.3 Contributions 5

1.3.1 Incremental Construction 5

1.3.2 Scattered Data Approximation 7

1.3.3 Matching Desired Lighting 8

1.4 Thesis Organization 9

2 Background **10**

2.1 Introduction 10

2.2 Image-Based Modeling and Rendering 10

2.2.1 Non-Parametric Models 11

2.3 Surface Reflectance Fields 13

2.4 Surface Light Fields 15

2.5 OpenLF 17

2.5.1 Rendering 17

2.5.2 Visibility and Resampling 19

2.6 Online Methods 20

3 Incremental Construction of Surface Light Fields **22**

3.1 Introduction 22

3.2 Mathematical Background 24

3.2.1 Eigenvectors and Eigenvalues 24

3.2.2 Power Iteration Algorithm . . . 25
3.2.3 Principal Component Analysis . . . 26
3.2.4 Singular Value Decomposition . . . 29
3.3 Online SVD . . . 30
3.3.1 Error . . . 32
3.3.2 Missing Values . . . 32
3.4 Data-Driven Quality Heuristic . . . 34
3.5 Implementation . . . 36
3.5.1 The Surface Light Field Construction Process . . . 37
3.5.2 Pose Estimation . . . 39
3.5.3 Computer Hardware . . . 41
3.6 Results . . . 43
3.7 Conclusions . . . 45

4 Scattered Data Approximation 47
4.1 Introduction . . . 47
4.2 Scattered Data Approximation . . . 49
4.2.1 Piecewise Polynomial . . . 50
4.2.2 Shepard's Method . . . 50
4.2.3 Radial Basis Functions . . . 51
4.3 Least Squares . . . 52
4.3.1 Solving the Normal Equations . . . 54
4.3.1.1 QR Decomposition . . . 54
4.3.1.2 Singular Value Decomposition . . . 55
4.3.2 Local Least Squares Methods . . . 55
4.3.3 Moving Least Squares . . . 58
4.3.4 Weighted Least Squares . . . 58
4.4 Incremental Weighted Least Squares . . . 60
4.4.1 Adaptive Construction . . . 60
4.4.2 Hierarchical Construction . . . 61
4.4.3 Rendering . . . 62
4.5 Implementation . . . 63
4.5.1 Results . . . 64
4.6 Conclusion . . . 67
4.6.1 Incremental WLS and Online SVD . . . 67

5 Capturing a Surface Light Field Under Virtual Illumination 69
5.1 Introduction 70
5.2 System Calibration 74
5.2.1 Single Image Calibration 75
5.2.2 Reflective Object Calibration 76
5.2.2.1 Error 77
5.2.3 Projector Color Calibration 79
5.3 Multiple Cameras 80
5.3.1 Coverage 80
5.3.2 Integrating Multiple Cameras 80
5.3.3 Overlap 81
5.4 Multiplexed Illumination and High-Dynamic Range 82
5.4.1 Multiplexed Illumination 83
5.4.2 General Multiplexed Illumination 85
5.4.3 Multiplexed illumination for HDR images 86
5.5 Implementation and Results 88
5.5.1 Results 89
5.5.2 Error 89
5.6 Conclusion 90

6 Summary and Conclusion 91
6.1 Conclusion 91
6.1.1 Incremental Construction 91
6.1.2 Scattered Data Approximation 92
6.1.3 Virtual Illumination 92
6.2 Limitations 93
6.3 Future Work 94
6.3.1 Incremental Construction 94
6.3.2 Surface Light Fields 95

BIBLIOGRAPHY 98

LIST OF TABLES

4.1 RMS Error values from reconstructing a WLS approximation while varying the center layout. The error was computed with a training set of 64 images and an evaluation set of 74 images. The disk is a slight improvement in terms of error compared to the grid, and it has a large benefit in terms of reducing the variance of the error. 67

5.1 Angular error vs. relative radius error 77

LIST OF FIGURES

1.1 Left: Spiderman 2 by Sony Imageworks. Right: Matrix Reloaded by ESC Entertainment. The top row shows a few of the source images captured as a pre-process. The bottom row shows novel viewpoints reconstructed from the imagery. In order to animate the SLF in the Matrix Reloaded, the artists collected several sets of SLFs with different extreme facial expressions, and interpolated between them to generate the facial expressions used in the movie. 2

1.2 The first step in the SLF process is to capture a set of photographs of a real object from a variety of viewpoints. These images are processed to generate a compressed representation, which can be used to render novel images from this representation. Dataset courtesy of Intel Corp. . . . 3

1.3 A heart figurine, a marble pestle, and a copper pitcher captured and rendered with our online system. 6

1.4 A side-by-side comparison of the WLS reconstruction with an input image that was not included in the training set. 7

1.5 Left: pitcher model in St. Peter's light probe. Right: heart model in Uffizi light probe. 8

2.1 The Bidirectional Subsurface Scattering Distribution Function. The incident light arrives at the surface point (u_i, v_i) from direction (θ_i, ϕ_i). It is then scattered through the material and exits at surface point (u_e, v_e) in direction (θ_e, ϕ_e). 12

2.2 A taxonomy of reflectance functions with dimensionality. From Mueller (Mueller et al., 2004). 13

2.3 Surface light fields can be represented as the function $L(u, v, \theta, \phi)$, a 4D space composed of the product of the 2D space of surface points and the 2D space of view directions. 16

2.4 Discretizing the view directions and surface locations produces a matrix representation of the SLF. This matrix can then be factored into smaller matrices which represent the surface variation and view variation. . . . 18

2.5 The surface functions $g(u, v)$ are parameterized across triangle rings. To get the color for the triangle, each of the three triangle ring functions are weighted by $\beta(u, v, k)$ and summed. The weighting function sums to 1.0 everywhere within the triangle. From Chen et al. (Chen et al., 2002). 19

2.6 Resampling of views. Left: The camera positions are projected onto the local basis. Center: Delaunay triangulation of views. Right: Resampling to uniform grid. From Chen et al. (Chen et al., 2002). 20

3.1 A heart figurine, a marble pestle, and a copper pitcher captured and rendered with our online system. 23

3.2 This sequence shows the image acquisition process. The left frame shows an image from the camera. In the middle is the reconstruction before this image is incorporated, with highlights interpolated from nearby views. The right image shows the reconstruction after it is incorporated, with the correct highlights. 25

3.3 The eigenvalues of a 655x75 matrix from a SLF dataset. The mass of the eigenvalues are concentrated in the first dozen terms, which implies that a good approximation of the full matrix can be computed using only a subset of the eigenvectors and eigenvalues. 28

3.4 The Online SVD. A new point c is incorporated into the existing SVD. The orthogonal component p is computed by projecting onto U. If $\|p\|$ is below a threshold, then we can incorporate this new sample by simply rotating U. Otherwise, we must increase the rank of the approximation by adding another eigenvector that is orthogonal to the current set of eigenvectors. 31

3.5 The Mean Squared Error of a reconstructed light field as a function of the rank. The majority of the light field is captured after 4-6 terms. For the same number of terms, the Online SVD has more error than the batch SVD. 33

3.6 Left: The bust dataset with areas of missing data highlighted in red. Right: Imputation of the missing data fills in the areas with plausible values. 35

3.7 The data-driven quality heuristic $\psi(s, t, \theta, \phi)$ displayed on the star model. Left: The heuristic is dark in areas where images have already been captured, indicating that the area has been well-sampled. Right: In areas where more data is needed, the heuristic is bright to draw attention to the area. 36

3.8 A side-by-side comparison of the Buddha model constructed with the OpenLF system (left) and our incremental approach (right). There are some slight differences noticeable in the chest area. 37

3.9 Acquiring the geometry of a marble pestle. Left: The object is scanned using a contact scanner. Right: The resulting mesh after triangulation and editing. 39

3.10 Top: The OpenLF system. Bottom: Our online system. The two systems share several components, including the visibility, resampling, and rendering. We introduce several new components to convert light field construction into an online process. 40

3.11 The pose estimation output. Left: The output from the video camera, showing the object and the tracking fiducials. Right: The tracking fiducials are segmented from the image and the pose estimation is computed. 40

3.12 To improve the efficiency of the texture lookup on the graphics hardware, the numerous small textures are packed into larger textures. Left: The viewing functions $h(\theta, \phi)$ are parameterized over the hemisphere and stored in view maps. Right: The surface functions $g(u, v)$ are parameterized over the triangles and stored in surface maps. 42

3.13 Left: A captured image of the star model. Right: The star model rendered in our system. Dataset courtesy of Intel Corporation. 42

3.14 Timing results in seconds for several models. The bust has 7K triangles, the Buddha has 12K triangles, the star has 5K triangles, the heart has 2.7K triangles, and the pitcher has 3K triangles. 43

3.15 Measuring the convergence of surface light field construction using the data-driven heuristic. Compared to random selection, the use of the data-driven heuristic dramatically speeds convergence. The PSNR of the reconstruction was compared to a reference SVD implementation at 10 random viewpoints. 44

3.16 The total reconstruction error of the Online SVD from 20 random permutations of the original dataset. The reference PCA is shown in the lower left corner. A permutation of the SVD where the samples were sorted by norm is shown in the upper right corner. This indicates that the error is greatest when the samples are correlated. 45

4.1 A side-by-side comparison of the WLS reconstruction with an input image that was not included in the training set. 49

4.2 A diagram of the weighted least squares approach to function representation. Each center constructs a low-degree polynomial approximation based on samples in their neighborhood. These local approximations are then combined to form a global approximation. 56

4.3 Two possible distance-weighting functions, with a variable support. Both functions are 1.0 at the center, and taper off to zero as the distance increases. While they are well-behaved within the support interval, they must be clamped outside this interval to avoid spurious weighting. . . 57

4.4 A diagram of the Adaptive construction of WLS. Initially, there are a fixed number of centers, each with a large domain. As new points are added to the data structure, they are classified according to which domains they fall within. When the number of points within a domain crosses a threshold, the domains are shrunk. 60

4.5 A diagram of the Hierarchical construction of WLS. Initially, there is a single center with a large domain. As points are added to the data structure, they are classified according to the domains they fall within. When the number of points within a domain crosses a threshold, the domain is subdivided into four smaller domains. 61

4.6 The heart object, rendered in our system. 62

4.7 A diagram of the SLF capture and rendering system. Images are captured using a handheld video camera, and passed to the system. The pose of the camera is estimated using fiducials in the environment. Using the mesh information, visibility is computed and the surface locations are back-projected into the image. Each of these samples are incorporated into the Incremental Weighted Least Squares approximation, and sent to the card for rendering. The user can use this direct feedback to decide where to move the video camera to capture more images. . . . 63

4.8 Timing results (in seconds per image) for the Incremental WLS construction. We measured three quantities; the time to compute the visibility and reproject the vertices into the image, the Least Squares fitting times, and the time to transfer the computed results to the graphics card for rendering. The dominant term is the Least Squares fitting. 65

4.9 A description of the models and construction methods used for timing data. 65

4.10 The reconstruction error of the hierarchical construction versus a batch construction for a single surface patch of the bust model. Each method used only the input samples available, and the error was measured against the full set of samples. The hierarchical algorithm is initially superior to the batch algorithm, and continues to be similar in error behavior while also being much faster to compute. 66

5.1 Sample light probes. Left and middle: Real light probes captured from St. Peter's Cathedral and Uffizi Gallery, courtesy of Paul Debevec (from *debevec.org*). Right: Synthetic light probe created by the artist Crinity. 69

5.2 Left: pitcher model in St. Peter's light probe. Right: heart model in Uffizi light probe. 70

5.3 A diagram of the capture system. Light is projected onto a large screen, which is then reflected onto the object. The object is mounted on a pan/tilt device. 72

5.4 A picture of the Surface Light Field capture system. The light is projected onto the screen and reflects down onto the object. The object is mounted on a tracking board, which is mounted on a pan/tilt device. The fiducial markers are used to estimate the object's position and orientation relative to the screen. The black curtains on the walls minimize light scattering in the lab. 73

5.5 Mapping the virtual illumination environment to our physical setup. Left: The virtual lighting environment. Right: The image that is projected onto the screen. Note that the image is stretched upward and outward to represent the rays of light from the virtual illumination environment. 74

5.6 Calibration method for planar screens. Left: A diagram of the physical setup for geometric calibration of the screen. Right: The calibration image, which shows both the projected fiducial board and the physical fiducial board. This is used to establish a correspondence between the two coordinate systems. 75

5.7 The reflective object calibration process, shown for a planar screen on the left and a corner screen on the right. Top: A reflective calibration object is placed in the same location as the object. Pixels are projected onto the screen and segmented from the images. Bottom: The reflected rays are used to index into the environment map, producing an image which is correct from the point-of-view of the object. 78

5.8 The relationship between the RGB color values that are sent to the projector and the color values that are actually displayed on the projector. We fit a polynomial to these curves and inverted it to correct for the error. 79

5.9 How much of the environment does the screen cover? Left: A single camera position covers only a portion of the environment, resulting in an incorrect SLF. Right: Four cameras cover the entire hemisphere of light above an object, leaving only a small hole in the top. 81

5.10 Multiplexing the St. Peter's Cathedral light probe (light probe courtesy of debevec.org). Left: Low-dynamic range multiplexing. The screen is divided into 9 regions, where each region is either "on" or "off". Approximately half of the regions are "on" at a time. Right: High-dynamic range multiplexing. Each region represents a different exposure level. . 87

5.11 Different HDR levels of the heart model illuminated with the St. Peter's Cathedral light probe. 89

5.12 An example image from the capture system which shows the camera blocking part of the screen. 90

6.1 The effect of tracking quality on SLF reconstruction. Left: High-quality tracking and calibration from Chen et. al (Chen et al., 2002) enables the SLF to accurately represent surface properties. Right: Errors in real-time tracking and calibration cause the SLF of a similar object to blur out interesting surface properties. 94

LIST OF ABBREVIATIONS

BTF	Bidirectional Texture Function
BRDF	Bidirectional Reflectance DistributionFunction
GPUs	Graphics Processing Units
HDR	High-Dynamic Range
IBR	Image-Based Rendering
kNN	k-Nearest Neighbor
LoT	List of Tables
LoF	List of Figures
RBFs	Radial Basis Functions
SLF	Surface Light Field
SRF	Surface Reflectance Field
SVD	Singular Value Decomposition
ToC	Table of Contents
WLS	Weighted Least Squares

CHAPTER 1

Introduction

The richness I achieve comes from Nature, the source of my inspiration.
– Claude Monet

The beauty and complexity of the natural world holds an enduring fascination for artists. From the earliest painters to modern-day digital artists, humans have looked to nature for subject material. One component of this is the desire to incorporate natural objects into modern artistic expressions such as movies and games. These elements heighten the sense of realism by providing familiar touchstones.

As a consequence, digital artists have developed a number of techniques to represent the appearance of surfaces. Traditional approaches often require the artists to hand-create a set of shaders that match the desired surface properties. However, the demand for photorealistic images in movies and games has outpaced the ability of artists to create the tens (or hundreds) of complex shaders per surface that are needed.

Driven by these limitations, image-based methods have become increasingly important. Image-Based Rendering (IBR) is a set of techniques that use photographs to generate representations of geometry and surface appearance. The appeal of IBR is that it is often easier to *acquire* the visual phenomena than to *simulate* it. Simulating physical processes such as subsurface scattering and interreflection can be a daunting task requiring hours of computer simulation. On the other hand, advances in digital cameras, 3D laser scanners, and other imaging technology enable us to easily capture large amounts of geometric and radiance data. The challenge is to represent all of this data in such a way that novel images can be generated at interactive rates.

In this dissertation I focus on the Surface Light Field (SLF) (Miller et al., 1998) as a representation of radiance data. A SLF is a function which assigns a color value to every ray leaving every point on a surface. It is constructed by acquiring images from different viewpoints and mapping these images onto the surface of a known geometric model, as shown in Figure 1.2. This representation enables the display of complex, view-dependent illumination of real-world objects at interactive rates. Surface Light

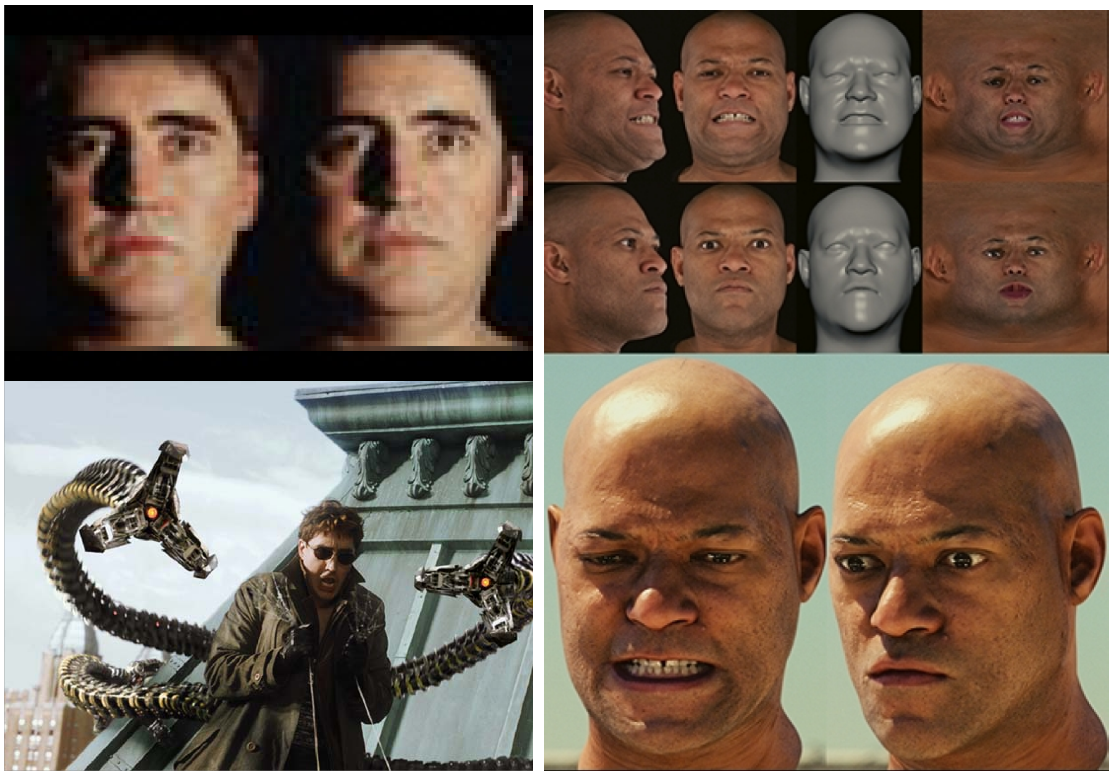


Figure 1.1: Left: Spiderman 2 by Sony Imageworks. Right: Matrix Reloaded by ESC Entertainment. The top row shows a few of the source images captured as a pre-process. The bottom row shows novel viewpoints reconstructed from the imagery. In order to animate the SLF in the Matrix Reloaded, the artists collected several sets of SLFs with different extreme facial expressions, and interpolated between them to generate the facial expressions used in the movie.

Fields (SLFs) can represent a wide variety of complex surface properties, including anisotropy, Fresnel reflection, forward/backward scattering, off-specular scattering, and diffuse inter-reflection.

SLFs have recently been used in a number of games and movies. Examples from two recent movies are shown in Figure 1.1. The artists were prompted to use image-based techniques due to the difficulty of representing the complex reflectance properties of the human face using traditional CG techniques. SLFs enabled the artists to create photorealistic characters that could be integrated into virtual and real environments.

1.1 Surface Light Field Capture Difficulties

The two primary goals of SLF research are high-quality representation and interactive rendering, and much of the previous research has focused on these goals. However, the

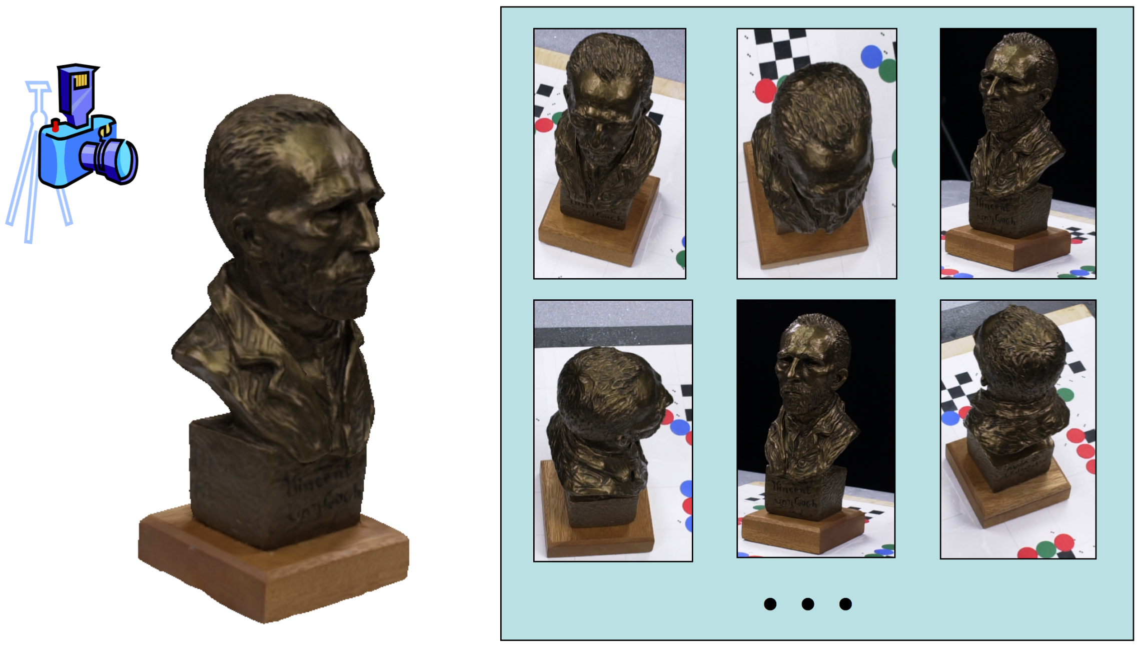


Figure 1.2: The first step in the SLF process is to capture a set of photographs of a real object from a variety of viewpoints. These images are processed to generate a compressed representation, which can be used to render novel images from this representation. Dataset courtesy of Intel Corp.

process of capturing all of the necessary data has remained difficult and time-consuming. It is my belief that the capture process is one of the major hindrances towards wider adoption of SLFs. In this dissertation, I attempt to address this difficulty by presenting several techniques to simplify the capture of SLFs. I focus on three problems; the lack of feedback during capture, handling missing data, and the problem of matching a desired virtual lighting environment.

The first difficulty with SLF capture is the lack of feedback. Collecting the numerous images needed for the construction of surface light fields is a time-consuming and tedious process. Since the result can be viewed only after a lengthy post-process is complete, it can be difficult to determine when the light field is sufficiently sampled. It is not enough to uniformly sample the hemisphere, as this may miss high-frequency information such as highlights. Often, uncertainty about the sampling density leads users to capture many more images than necessary in order to guarantee adequate coverage. If undersampling artifacts are visible in the result, more images must be acquired and the entire factorization post-process must be repeated. These data-acquisition problems can be traced to the lack of feedback during the image capture process, as humans are quite adept at recognizing undersampling errors.

The second difficulty with SLF capture is handling missing and irregularly-sampled data. An optimal capture process would acquire radiance data from every direction at every point on the surface. This is impossible due to the sheer amount of data that this would require. As a consequence, the data samples are often sparse and scattered. The data is resampled to fit the mathematical representation and compressed, and novel viewpoints are reconstructed by interpolating or extrapolating from nearby points in the compressed representation. Each of these steps: resampling, compression, and reconstruction, can create artifacts in the final result. To minimize these artifacts, the mathematical representation must be designed with the data characteristics in mind. The representation must also be compact, so that it can rendered efficiently. This often means that the reconstruction algorithm must run on graphics hardware to exploit the computational power and bandwidth of these cards.

The third difficulty with SLF capture is matching desired lighting. An artist may have a desired lighting environment, such as an environment map from a game or a light probe from a remote location. For example, a game developer writing a DaVinci Code thriller may have several scenes that are set in famous cathedrals, and it is important that objects in the game match the cathedral lighting. It would be expensive (or perhaps impossible) to travel to these actual locations to capture SLFs of the objects. This is a problem since source images for the SLF are captured under specific illumination conditions, and the SLF is constructed from these images. This fixed lighting restriction makes the SLF compact and fast to render, but it means that the SLF retains the appearance of the lighting environment under which it was captured. This makes it hard to capture an object as lit by a desired lighting environment. Being unable to match this lighting is a serious limitation, as it precludes the use of SLFs in movies or games.

In this dissertation, I present a series of approaches to deal with these limitations.

1.2 Thesis

Three major problems with Surface Light Field construction, (1) lack of feedback, (2) difficulty handling missing data, and (3) matching desired illumination, can be addressed by (1) enabling incremental construction, (2) employing scattered data approximation techniques, and (3) capturing under virtual lighting environments.

1.3 Contributions

To address these three limitations of the SLF capture process, I have developed three novel techniques.

1.3.1 Incremental Construction

To address the lack of feedback, I discuss a system for incrementally capturing, constructing, and rendering directionally-varying illumination. As each image is captured, it is incorporated into the SLF by incrementally building a low rank linear approximation. These partial results are rendered interactively on graphics hardware. This real-time feedback enables the user to preview the lighting model and direct the image acquisition towards undersampled areas of the object. The incremental construction method is based on the Online SVD (Brand, 2002).

We also develop a novel data-driven heuristic that highlights undersampled areas of the SLF and directs the user towards effective camera views. This heuristic is used to aid the user in identifying undersampled regions. Our system is an order of magnitude faster than previous systems, and reduces the time necessary to capture the images and construct a surface light field from hours to minutes. Example images from the system are shown in Figure 1.3. I presented these methods at the 2005 Eurographics Symposium on Rendering (Coombe et al., 2005).

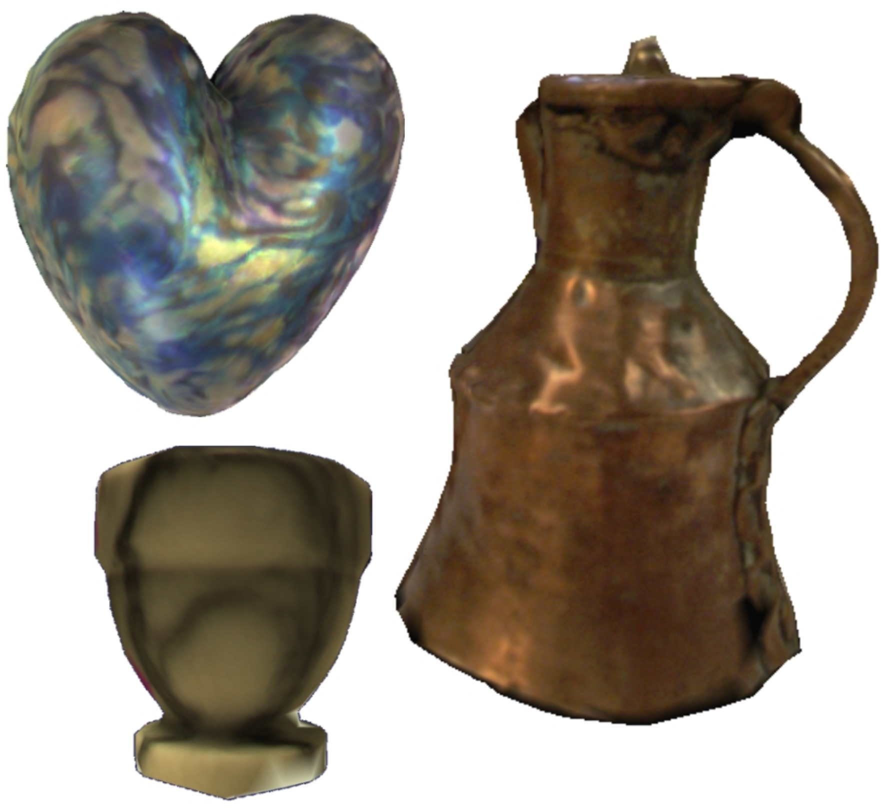


Figure 1.3: A heart figurine, a marble pestle, and a copper pitcher captured and rendered with our online system.

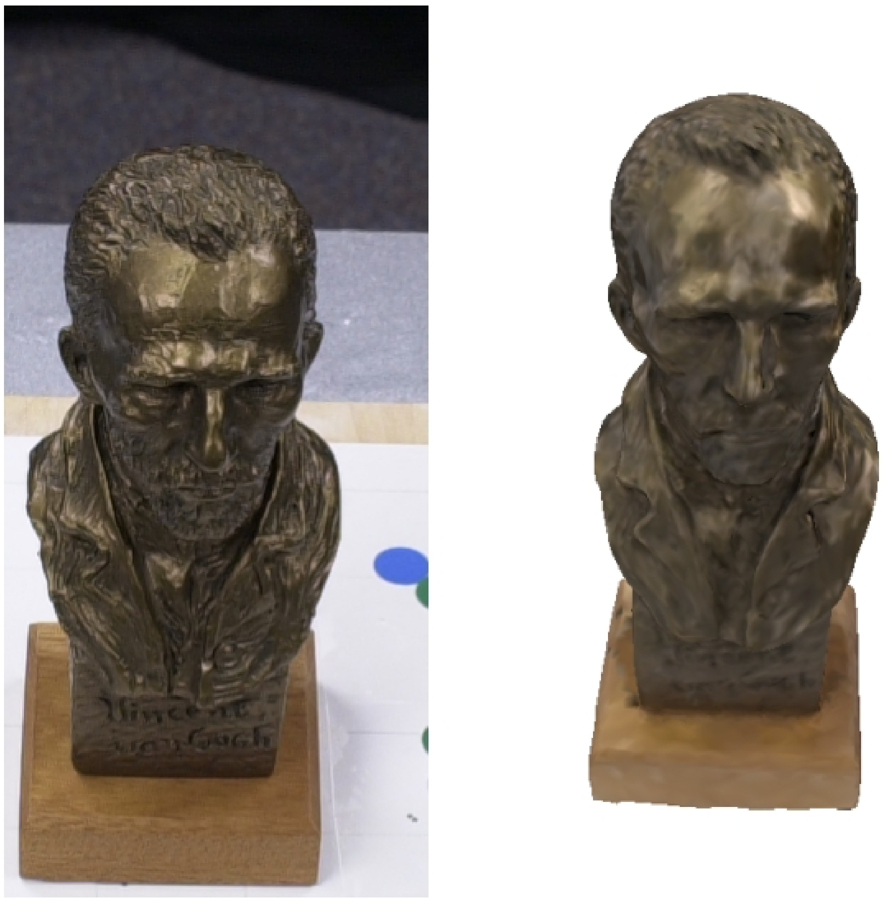


Figure 1.4: A side-by-side comparison of the WLS reconstruction with an input image that was not included in the training set.

1.3.2 Scattered Data Approximation

The Incremental SLF construction technique is a powerful technique for capturing spatially- and directionally-varying surface appearance. However, it requires that the data lie on a regular grid. Since SLF data can be sparse and irregularly distributed, it requires extensive resampling which can introduce artifacts such as blurring. One way to avoid this resampling is to use data approximation techniques which were developed for scattered data. I discuss one particular technique, Weighted Least Squares, and our extension to an incremental framework. Each surface patch consists of a set of Weighted Least Squares (WLS) node centers, which are low-degree polynomial representations of the anisotropic exitant radiance. During rendering, the representations are combined in a non-linear fashion to generate a full reconstruction of the exitant radiance. We developed a rendering algorithm that is fast, efficient, and implemented entirely on the GPU. An example of the result is shown in Figure 1.4.

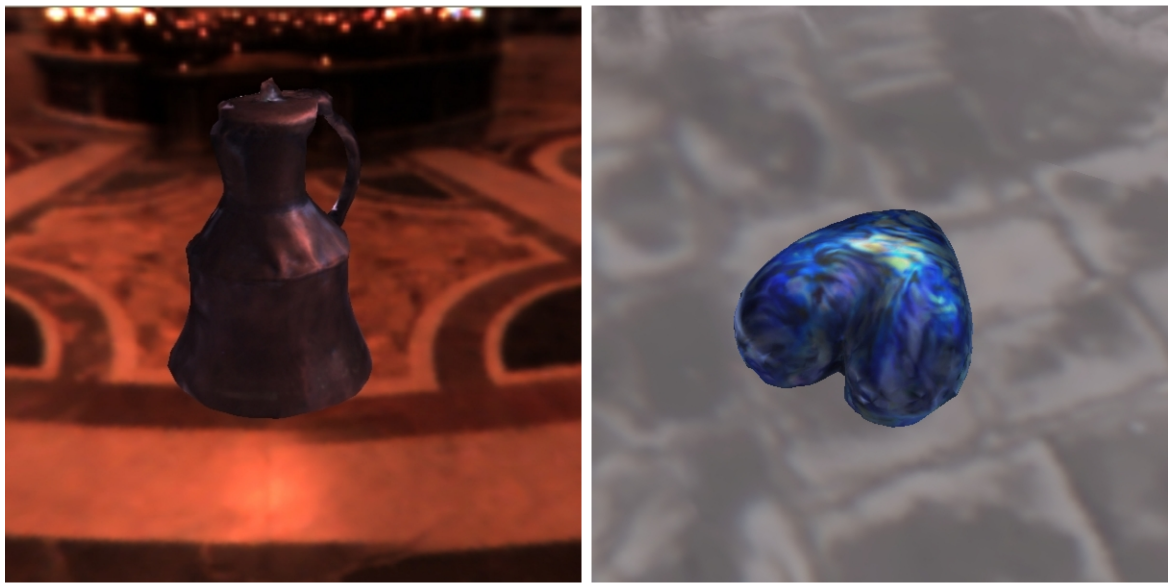


Figure 1.5: Left: pitcher model in St. Peter's light probe. Right: heart model in Uffizi light probe.

Building off our work in incremental construction, we designed the algorithm to process images incrementally instead of in the traditional batch fashion. We present two approaches for incremental construction, a hierarchical refinement and an adaptive refinement. This human-in-the-loop process enables the user to preview the model as it is being constructed and to adapt to over-sampling and undersampling of the surface appearance. These techniques were presented at the 2006 Conference on Computer Graphics Theory and Applications (Coombe and Lastra, 2006).

1.3.3 Matching Desired Lighting

Another limitation of SLFs is that they can only represent the fixed lighting conditions of the environment where the model was captured. If the artists wishes to capture the object as if it were lit using a desired lighting environment, then there are several possible approaches. One approach is to illuminate the object using a combination of physical lights, but this is a low-resolution approximation which is difficult to calibrate. Another approach is to capture a full 6D Bidirectional Texture Function (BTF) and only use the portion that corresponds to the desired lighting. This approach is high-quality and allows arbitrary lighting, but requires several orders of magnitude more data and is unnecessary if the lighting environment is known.

I describe a third approach: a method for capturing a SLF as if it were lit from a virtual illumination environment. I use a simple setup consisting of a projector, a camera, a pan-tilt unit, and tracking fiducials to recreate the desired lighting environment. This enables the artist to capture objects lit using any desired lighting environment, either real or synthetic. Some examples of objects captured under virtual illumination are shown in Figure 1.5.

There are several challenges to recreating the virtual lighting environment. We determine the correspondence between surface points and rays in the virtual environment using two calibration methods; a fast method for planar screens, and a slower technique for screens with arbitrary geometry. To decrease noise and improve the quality of the capture under low- and high-dynamic range environment maps, I extend Multiplexed Illumination to handle High-Dynamic Range images. These techniques were presented in (Coombe et al., 2007) and (Frahm et al., 2006) .

1.4 Thesis Organization

The rest of the dissertation is organized as follows. The next chapter gives an overview of research in the area of image-based modeling and rendering, and specifically in the area of image-based representations of surface appearance. Chapter 3 discusses a method for incrementally constructing a SLF using the Online Singular Value Decomposition (SVD). Chapter 4 details how to represent the SLF as a scattered data approximation problem and use WLS to represent the radiance values. Chapter 5 describes a physical setup to capture to SLF under virtual illumination. Finally, in Chapter 6, I summarize the major results of this dissertation and conclude with a look at areas of future improvement.

CHAPTER 2

Background

2.1 Introduction

In order to compactly represent the view-dependent appearance of an object, the SLF makes particular assumptions about the light paths. In order to understand the SLF algorithm, it is important to understand why these assumptions were made, and the assumptions that other researchers have made in designing their algorithms. This chapter gives a short introduction to the field of IBR, with an emphasis on the degrees of freedom of the radiance representations. I then discuss SLFs and a particular SLF implementation, OpenLF, which was the basis for much of my research.

2.2 Image-Based Modeling and Rendering

Image-Based Rendering (IBR) is a powerful approach for generating real-time photorealistic computer graphics. While it started with pure lightfield algorithms (Levoy and Hanrahan, 1996; McMillan and Bishop, 1995) it quickly became clear that augmenting the representation with geometry information could improve the quality of the reconstruction (Gortler et al., 1996; Debevec et al., 1996; Miller et al., 1998). Buehler et al. (Buehler et al., 2001) established a set of criteria that IBR algorithms should possess. These criteria address geometric concerns such as ray coherency, continuity, and epipole consistency, as well as behavioral characteristics such as real-time behavior, the use of geometric proxies, and the ability to handle unstructured input.

One of the noticeable characteristics of IBR algorithms is that they tend to involve large datasets. For example, the Lafortune (Lafortune et al., 1997) factorization presented in Hillesland (Hillesland et al., 2003) involves solving 10^6 nonlinear optimizations of 10^3 data samples. The tensor product computation used by Furukawa (Furukawa et al., 2002) requires 10^4 factorizations of 3D matrices of $10^3 \times 10^3 \times 10^2$. Consequently, much of algorithm design revolves around issues of compression and representation.

The choice of representation of this captured data is crucial for interactive rendering. Appearance modeling approaches can be categorized into *parametric* and *non-parametric*. Parametric approaches assume a particular model for the surface appearance (such as the Lafortune (Lafortune et al., 1997) model used by McAllister (McAllister et al., 2002)). However, parametric models have difficulty representing the wide variety of objects that occur in real scenes, as observed in Hawkins et al. (Hawkins et al., 2001). Some of the physical properties that are difficult to represent are anisotropic materials, Fresnel reflection, forward/backward scattering, off-specular scattering, diffuse inter-reflection, and subsurface scattering.

Non-parametric (also called data-driven) approaches use the captured data to estimate the underlying function and make minimal assumptions about the behavior of the reflectance. Thus non-parametric models are capable of representing a larger class of surfaces, which accounts for their recent popularity in image-based modeling (Chen et al., 2002; Furukawa et al., 2002; Zickler et al., 2005). In this dissertation I focus on non-parametric SLF models, so I will describe the background research in this area in further detail.

2.2.1 Non-Parametric Models

The Eurographics State of the Art Report on Acquisition, Synthesis and Rendering of Bidirectional Texture Functions (Mueller et al., 2004) presents a taxonomy of image-based modeling representations, shown in Figure 2.2. The most general appearance representation is the Bi-directional Surface Scattering Reflectance Distribution Function (BSSRDF) (Nicodemus et al., 1977) represented as:

$$L(u_i, v_i, \theta_i, \phi_i, u_e, v_e, \theta_e, \phi_e)$$

The subscript i denotes incident radiance, and the subscript e denotes exitant radiance. This function, shown in Figure 2.1, is an 8D function which models the scattering of light from the surface. The incident light arrives at the surface point (u_i, v_i) from direction (θ_i, ϕ_i). It is then scattered through the material and exits at surface point (u_e, v_e) in direction (θ_e, ϕ_e).

The term “Surface Scattering” in BSSRDF encompasses physical behavior such as translucency, transparency, and subsurface scattering. Since measuring these properties is quite complex (Goesele et al., 2004), the light that travels through the surface is often ignored and the points (u_i, v_i) and (u_e, v_e) are treated as a single point (u, v). This assumption leads to the BTF, a model of the entire set of incident lighting directions

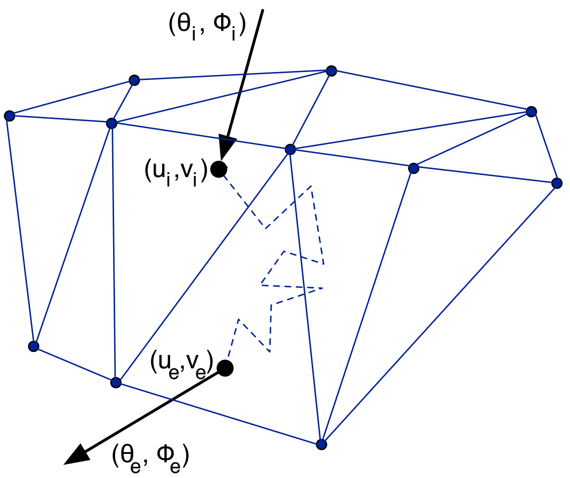


Figure 2.1: The Bidirectional Subsurface Scattering Distribution Function. The incident light arrives at the surface point (u_i, v_i) from direction (θ_i, ϕ_i). It is then scattered through the material and exits at surface point (u_e, v_e) in direction (θ_e, ϕ_e).

and exitant lighting directions (viewpoints)

$$L(s, t, \theta_i, \phi_i, \theta_e, \phi_e)$$

The BTF is also known as the spatially-varying BRDF or view-dependent texture maps (Debevec et al., 1998). Lensch et al. (Lensch et al., 2001) use the Lafortune (Lafortune et al., 1997) representation and clustered BRDFs from sparse acquired images in order to create spatially-varying BRDFs. This is inspired by the observation that while objects rarely have a uniform BRDF, many objects consist of a small set of BRDFs. The object is split into clusters with different properties, a set of basis BRDFs is generated for each cluster, and the original samples are reprojected onto the space. This factorization is expensive and cannot be done in real-time.

McAllister et al. (McAllister et al., 2002) describe a system for capturing the BTF and methods for fitting the data to a Lafortune representation. The compact form of the Lafortune model allows the authors to process a large amount of data and render at interactive rates. Gardner (Gardner et al., 2003) describe a BRDF capture device that uses a linear light source (as opposed to a point source), which can also estimate surface normals and a height field.

Surface Reflectance Fields, Bidirectional Reflectance Distribution Functions (BRDFs) and Surface Light Fields are models of reflectance that are derived from the BTF by further restricting the domain. The Surface Reflectance Field (SRF) models the inci-

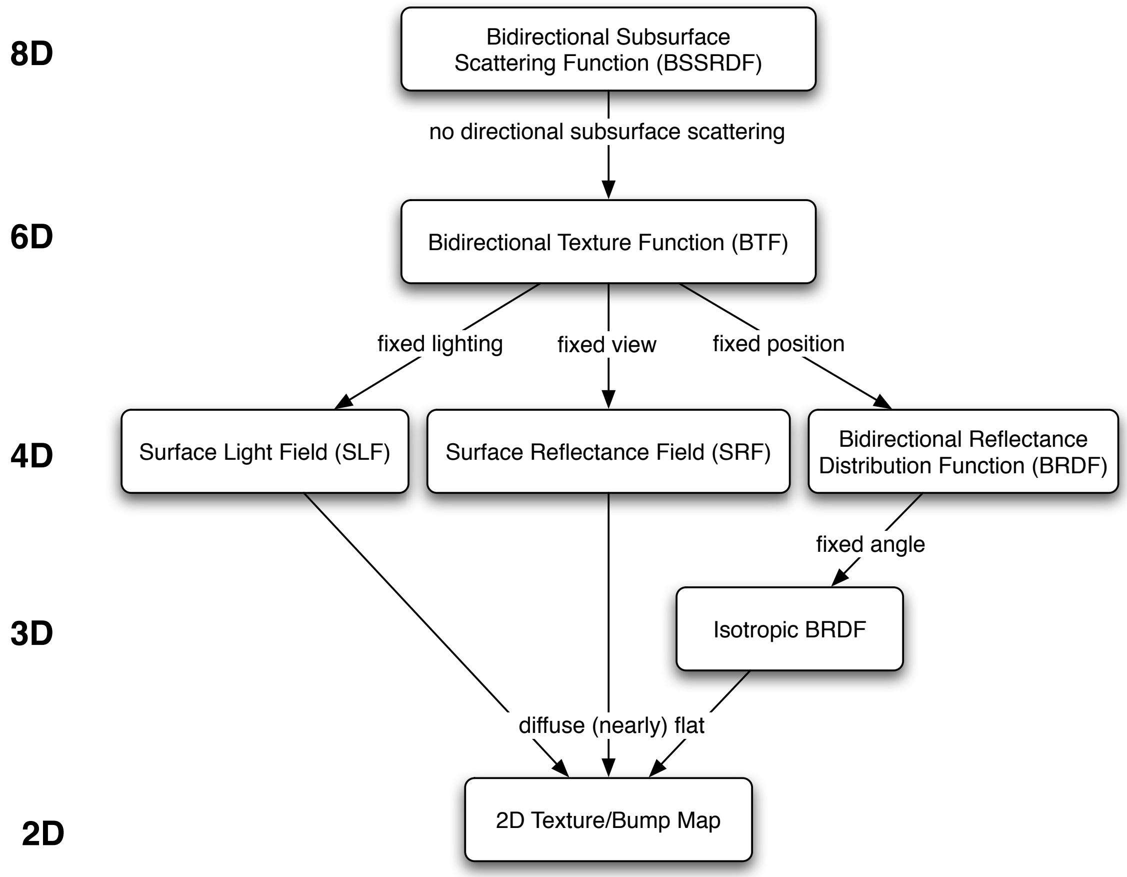

8D
6D
4D
3D
2D
Bidirectional Subsurface Scattering Function (BSSRDF)
no directional subsurface scattering
Bidirectional Texture Function (BTF)
fixed lighting
fixed view
fixed position
Surface Light Field (SLF)
Surface Reflectance Field (SRF)
Bidirectional Reflectance Distribution Function (BRDF)
fixed angle
Isotropic BRDF
diffuse (nearly) flat
2D Texture/Bump Map

Figure 2.2: A taxonomy of reflectance functions with dimensionality. From Mueller (Mueller et al., 2004).

dent radiance from a fixed viewpoint, while the SLF models the exitant radiance from arbitrary viewpoints under fixed lighting. Both Surface Reflectance Fields (SRFs) and SLFs allow surface variation, while the Bidirectional Reflectance DistributionFunction (BRDF) assumes uniform surface material but models incident and exitant radiance.

2.3 Surface Reflectance Fields

SRFs represents directionally-varying incident radiance for a fixed viewpoint. These are also known as *image-based relighting* methods (Debevec et al., 2000). The Surface Reflectance Field is represented as a 4D function $L(s, t, \theta_l, \phi_l)$. As shown in Figure 2.2, the SRFs have the same dimensionality as SLFs, which allows similar compression and rendering techniques to be used.

These methods typically capture a set of images of the scene lit by basis functions.

Since light is linear, images under arbitrary lighting can be reconstructed by projecting the desired lighting onto the light basis. Debevec (Debevec et al., 2000) acquire radiance samples of a human face using a custom-built light stage, and compute a reflectance function that can be used to generate images under novel lighting conditions. A similar system was used to capture reflectance fields with high-frequency BRDFs (Koudelka et al., 2001).

Malzbender (Malzbender et al., 2001) use a simple basis system and focus on fast rendering of surface reflectance fields. A camera is placed on a rig which is positioned over the object to be scanned. The rig is constructed in such a way that lights are positioned uniformly over the hemisphere, and a set of images are captured under different illumination conditions. The captured images are fit to a bi-quadratic polynomial at each pixel:

$$L(u, v; l_u, l_v) = a_0(u, v) + a_1(u, v)l_u + a_2(u, v)l_v + a_3(u, v)l_u^2 + a_4(u, v)l_u l_v + a_5(u, v)l_v^2$$

where l_u and l_v are the projections of the incident lighting vector (θ_i, ϕ_i) onto the local u, v basis. Fixing the camera position means that no intrinsic or extrinsic calibration of the camera is necessary, and the authors are able to capture a high-spatial frequency but low-lighting frequency. The coefficients can be stored in texture maps and rendered at interactive rates on graphics hardware. One advantage of this representation is that it is fairly general; the authors show how it can be used to encode different per-pixel lighting information such as bump-maps and depth-of-field.

Matusik (Matusik et al., 2004) represent the SRF as a large transport matrix and progressively refine this matrix as new images are acquired. The transport matrix maps the known input illumination to the captured output illumination. For a known illumination L_k and a captured image B_k, we get an equation of the form:

$$B_k = T L_k$$

where T is the unknown transport matrix. The goal of this method is to estimate T from the set of input/output pairs. Then, given an arbitrary illumination, an image of the scene under that illumination can be rendered by multiplying by the transport matrix. Since this matrix T is quite large (# of pixels in illumination $\times$ # of pixels in output image), it is stored as the sum of a set of 2D kernels:

$$T_i \approx \sum_w w_{k,i} R_{k,i}$$

These kernels can be progressively refined by representing the kernels as the leaf nodes of a kd-tree basis. Following the suggestion of Buehler (Buehler et al., 2001), they augment the transport database with geometric proxy information to improve the reconstruction.

The progressive refinement of the kernels provides several advantages. It allows the user to recognize areas of complex reflectance behavior and to capture more photographs in these areas. It also produces a more compact representation by only refining those areas which require more detail. I discuss the idea of user feedback in Chapter 3 and progressive refinement in Chapter 4.

Most lighting models only represent the directional variation of the incident light, assuming that the light comes from infinitely far away. This does not allow for spatial variation in the lighting such as spotlights and patterned lighting. Masselus (Masselus et al., 2003) present a system for capturing a SRF with spatially-varying light. This can be thought of as an extension of SRFs to the equation:

$$L(u, v, \theta_{(i,u)}, \theta_{(i,v)}, \phi_{(i,u)}, \phi_{(i,v)})$$

They fix a camera to a turntable with an object at the center, and rotate both under a projector. A set of lighting patterns is projected onto the object. These patterns are used to compute a set of basis functions which represent the lighting. These basis functions can be recombined to produce a relighted version of the scene from a single viewpoint. The system can represent phenomena which are not possible with many other SRF capture systems. However, the increased dimensionality significantly increases the acquisition time. To speed this up, the authors use a form of multiplexed illumination (Schechner et al., 2003), where several patterns are projected at once and the individual patterns are demultiplexed from the result. Even with this speedup, the capture times reported in the paper are around 40 hours.

2.4 Surface Light Fields

SLFs (Miller et al., 1998; Wood et al., 2000) are image-based representations of lighting which are mapped onto the surface of a geometric model. They represent the exitant radiance under fixed illumination conditions. This parameterization results in a compact representation that enables the capture and display of complex, view-dependent illumination of real-world objects. Constructing these representations can be a time-consuming and tedious process. The data sizes can be large, often requiring multiple

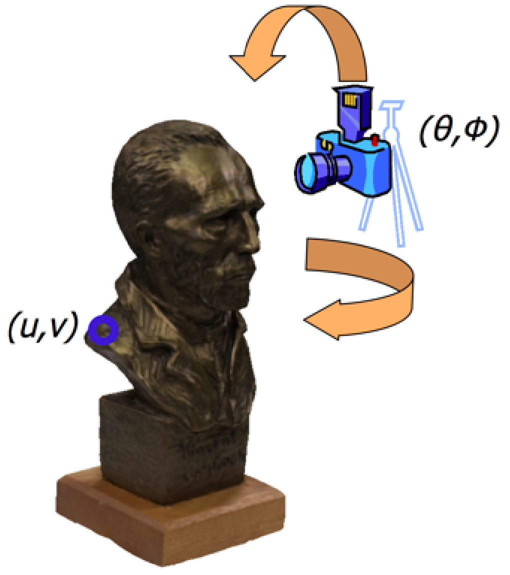


Figure 2.3: Surface light fields can be represented as the function $L(u, v, \theta, \phi)$, a 4D space composed of the product of the 2D space of surface points and the 2D space of view directions.

gigabytes to represent complex reflectance properties.

SLFs can be represented as the function $L(u, v, \theta_e, \phi_e)$. The variables u and v represent surface location on the mesh, and θ_e and ϕ_e represent view directions (to simplify notation, the subscript e will be implied in subsequent equations). The SLF parameterization is 4-dimensional, composed of the product of the 2D space of surface points and the 2D space of view directions. An example is shown in Figure 2.3.

Instruments designed to capture the BRDF or BTF of objects can collect surface light fields also. The capture stage of Matusik, et al. (Matusik et al., 2002) was used to model objects that are difficult to represent, including objects with indeterminate silhouettes such as fuzzy toy figures. They used two plasma panels, six cameras, and four Halogen lamps in their instrument.

Wood et al. (Wood et al., 2000) use a generalization of Vector Quantization and Principal Component Analysis to compress surface light fields, and introduce a 2-pass rendering algorithm that displays compressed light fields at interactive rates. They construct a representation that they call a *lumisphere*, which is a sphere of exitant lighting directions. The surface is scanned at a high-resolution using a laser-scanner, and each point on the surface has a corresponding lumisphere. They develop generalizations of both PCA and vector-quantization to compress these lumispheres. To improve the compression, the lumisphere is re-parameterized around the reflection direction. As a side-effect of the parameterization, the lumisphere can be rotated within the lighting environment (in a plausible but not physically-based manner).

The SLF describes the exitant radiance in every direction at every point on the surface. We can discretize this function over the surface patches and solid angles and represent it as a matrix. The columns of this matrix are the camera views, and the rows are the surface locations. Since storing these full data matrices would be impractical, several techniques have been developed to compress the data. Factorization approaches represent the 4D surface light field $L(u, v, \theta, \phi)$ as a sum of products of lower-dimensional functions

$$L(u, v, \theta, \phi) \approx \sum_{r=1}^{rank} g^r(u, v) h^r(\theta, \phi)$$

The number of terms r is the rank of the approximation. This is shown in Figure 2.4. This factorization attempts to decouple the variation in surface texture from the variation in lighting. These functions can be constructed by using Principal Component Analysis (Wood et al., 2000; Chen et al., 2002; Nishino et al., 2001) or non-linear optimization (Hillesland et al., 2003; McCool et al., 2001). The function parameters can be stored in texture maps and rendered in real-time (Chen et al., 2002). I discuss this approach in Chapter 3.

2.5 OpenLF

The approach that I have been describing was implemented in an open source software package called OpenLF (OpenLF,), developed by Intel and based on the research by Chen et. al. (Chen et al., 2002). We have based our research on the framework of OpenLF. This section describes several components of OpenLF, including visibility, resampling, and rendering.

2.5.1 Rendering

One of the advantages of the factorization approach to surface light fields is that the functions $g(u, v)$ and $h(\theta, \phi)$ can be stored in texture maps and rendered in real-time on graphics hardware (Chen et al., 2002). The functions $g(u, v)$ are known as *surface maps* and $h(\theta, \phi)$ are known as *view maps.*

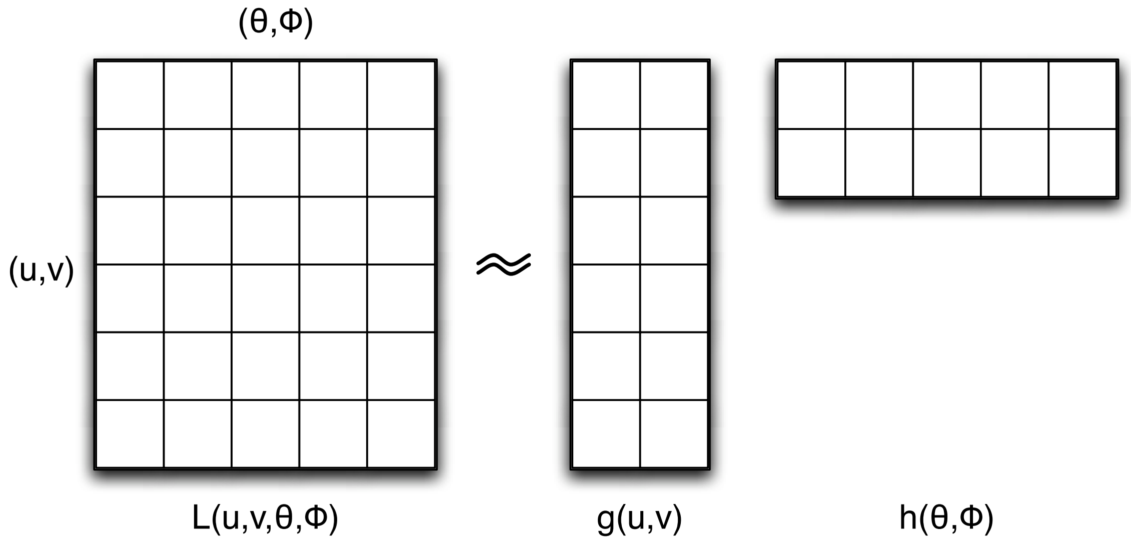


Figure 2.4: Discretizing the view directions and surface locations produces a matrix representation of the SLF. This matrix can then be factored into smaller matrices which represent the surface variation and view variation.

The exitant radiance directions (θ, ϕ) are parameterized over the hemisphere above each vertex. The surface locations (u, v) are parameterized over the triangle rings centered at each vertex. A triangle ring is the set of triangles which share a vertex. Using triangle rings instead of individual triangles avoids discontinuities at the edges (Chen et al., 2002), but requires that each triangle be represented by three surface maps (one per vertex k). A diagram is shown in Figure 2.5.

The rendering algorithm, which is executed on graphics hardware, evaluates the following function:

$$L_k(u, v, \theta, \phi) = \sum_{r=1}^{rank} \sum_{k=1}^{3} \beta(u, v, k) g_k^r(u, v) h_k^r(\theta, \phi)$$

at each of the triangles. A view vector is computed from the eye to each of the three vertices of the triangle. These vectors are projected onto the local basis system, and interpolated across the triangle. At each fragment, the vectors are normalized and used to index into the view maps $h^r_{(1,2,3)}(\theta, \phi)$. The functions $g^r_{(1,2,3)}(u, v)$ are computed by linearly interpolating the surface maps across the triangle. The two functions are multiplied together and weighted by a barycentric function $\beta(u, v, k)$ to get the final exitant radiance at this point. Each term r of the approximation is rendered in one pass and accumulated in a floating-point frame buffer.

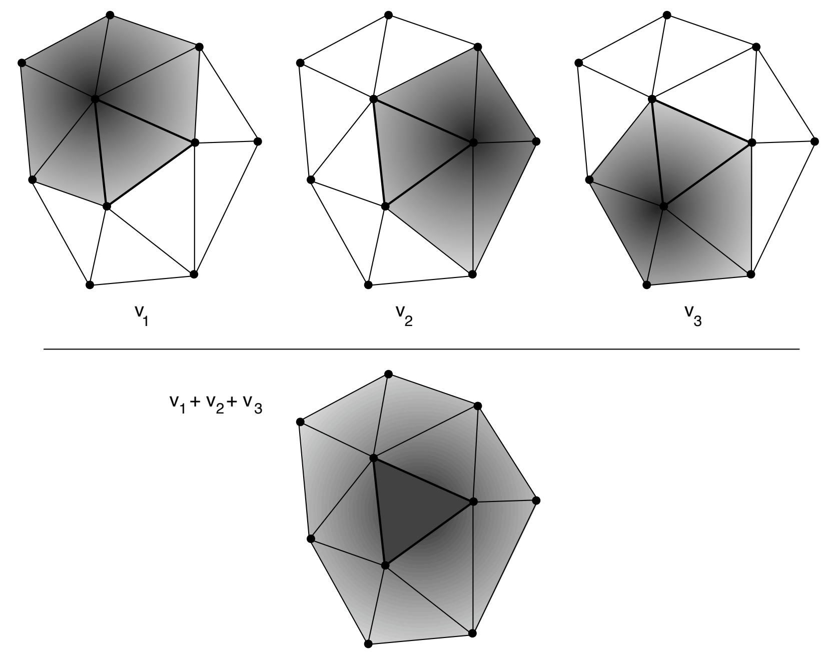


Figure 2.5: The surface functions $g(u, v)$ are parameterized across triangle rings. To get the color for the triangle, each of the three triangle ring functions are weighted by $\beta(u, v, k)$ and summed. The weighting function sums to 1.0 everywhere within the triangle. From Chen et al. (Chen et al., 2002).

2.5.2 Visibility and Resampling

The visibility is tested by projecting the triangle mesh using the camera's position and orientation. Visibility is computed on a per-vertex basis by back-projecting the vertex into the camera. If a point is determined to be visible, the colors are sampled from the input image using bilinear interpolation.

The matrix factorization approach to compression of surface light fields requires that the image data be resampled from the input images to fit into the rows and columns of the data matrices. This resampling occurs in both dimensions; not only are the camera locations at arbitrary locations in the hemisphere, but the projected areas of the triangles vary in each image.

Resampling the images to fill the columns of the data matrix is straightforward. If a triangle ring is determined to be visible, the colors are sampled from the input image using bilinear interpolation.

Resampling the camera views to cover the hemisphere is more complicated. In the OpenLF system, camera locations are treated as points in a Delaunay triangulation. This triangulation is used to weight each of the camera positions, which are stored in

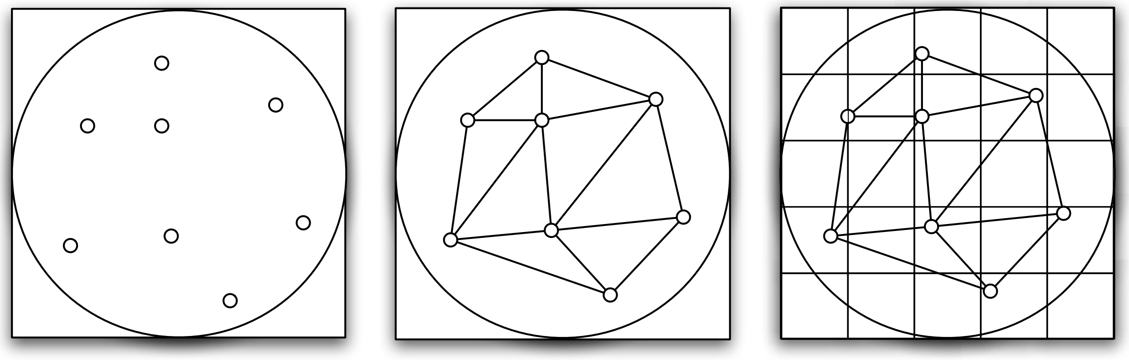


Figure 2.6: Resampling of views. Left: The camera positions are projected onto the local basis. Center: Delaunay triangulation of views. Right: Resampling to uniform grid. From Chen et al. (Chen et al., 2002).

the texture maps. Each image represents a point in the hemisphere. The 3D position of the camera is projected onto the vertex basis vectors and normalized. This point is projected down onto the plane and inserted into the Delaunay triangulation. The triangulation is then uniformly resampled at the view map resolution and stored. A diagram is shown in Figure 2.6.

Resampling can cause the amount of data to increase dramatically. In Chapter 3 I discuss methods to avoid this resampling by treating the SLF reconstruction as a scattered data approximation problem.

2.6 Online Methods

Most of the research in image-based modeling has focused on *batch processing* systems. These systems process the set of images over multiple passes, and consequently require that the entire set of images be available. For detailed capture of light fields, this requires significant storage (around 10^6 data samples (Hillesland et al., 2003)). In addition, incorporating additional images into these models requires recomputing the model from the beginning.

Recently researchers have developed *online methods* which provide direct feedback to the user. Schirmacher (Schirmacher et al., 1999) adaptively meshed the *uv* and *st* planes of a light field, and used an error metric along the triangle edges to determine the locations of new camera positions. Hillesland (Hillesland et al., 2003) updated a non-linear solution in an online fashion, but required multiple passes over the data. One of the inspirations for this research was Rusinkiewicz (Rusinkiewicz et al., 2002), who described a system to interactively capture geometry. The user was incorporated into the processing loop, which enabled them to view the sampling and to steer the solution to eliminate holes in the model.

Formulating surface light field construction as an *online processing* approach avoids these problems by incrementally constructing the model as the images become available. In the next chapter, I discuss a technique to convert the SLF batch capture process into an incremental process.

CHAPTER 3

Incremental Construction of Surface Light Fields

The goal of this research is to acquire the appearance of physical objects and render these objects at interactive rates. In contrast to analytical radiance models (Hanrahan and Krueger, 1993; Cook and Torrance, 1982; Oren and Nayar, 1994), the SLF approach has not been widely adopted. I believe that is partially because the capture process is time-consuming and complex.

In this chapter I discuss a system for interactively capturing, constructing, and rendering surface light fields by incrementally building a low rank approximation to the surface light field. Each image is incorporated into the lighting model as it is captured, providing the user with real-time feedback. This feedback enables the user to preview the lighting model and direct the image acquisition towards undersampled areas of the object. We also provide a novel data-driven quality heuristic to aid the user in identifying undersampled regions. Our system reduces the time necessary to capture the images and construct a surface light field from hours to minutes.

3.1 Introduction

Collecting the numerous images needed for the construction of surface light fields is a time-consuming and tedious process. Since the result can be viewed only after a lengthy post-process is complete, it can be difficult to determine when the light field is sufficiently sampled. It is not enough to uniformly sample the hemisphere, as this may miss high-frequency information such as highlights. Often, uncertainty about the sampling density leads users to capture many more images than necessary in order to guarantee adequate coverage. If undersampling artifacts are visible in the result, more images must be acquired and the entire factorization process must be repeated. This factorization process may take several hours for a complex SLF.

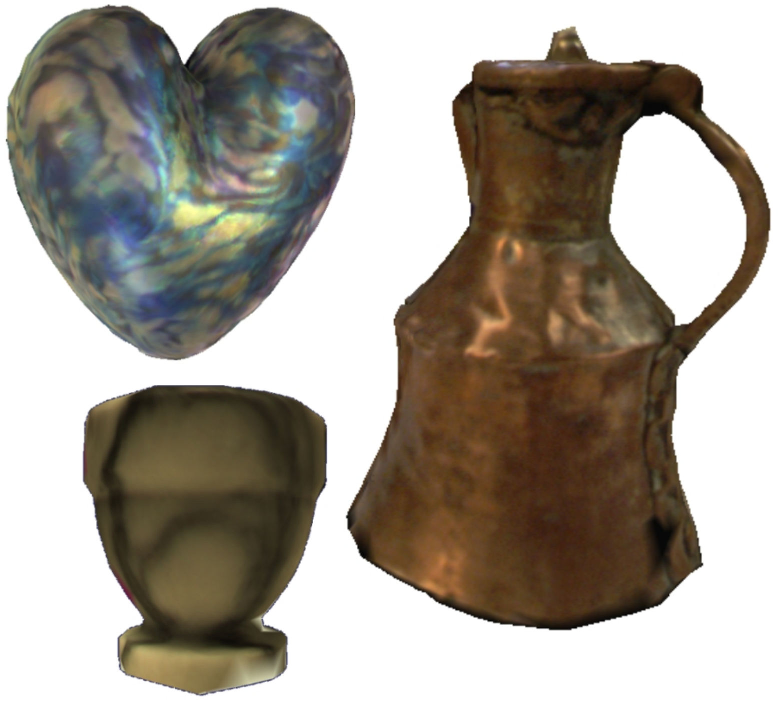


Figure 3.1: A heart figurine, a marble pestle, and a copper pitcher captured and rendered with our online system.

These data-acquisition problems can be traced to the lack of feedback during the image capture process, as humans are quite adept at recognizing undersampling errors. An incremental algorithm provides the user with feedback as the lighting model is being constructed, allowing the user to obtain the necessary quality and avoid taking many extra pictures. Examples of several models can be seen in Figure 3.1.

The goal of this work is a “casual capture” system which allows the user to interactively capture and view surface light fields. “Casual capture” refers to two elements of our approach; no specially-constructed capture rigs or tracking devices, and a “human-in-the-loop” interactive feedback system. We present a system for incrementally capturing and rendering a surface light field with fixed illumination conditions and known geometry. As each image is acquired, it is incorporated into the representation in a streaming fashion and displayed to the user. In this way, the user receives direct feedback about the capture process and can use this feedback to direct the image acquisition towards undersampled regions. Our online light-field capture process enables the user to examine the surface light field as it is being acquired, add images where necessary, and make the final determination as to whether the quality is sufficient. A sequence demonstrating this process is shown in Figure 3.2. We also introduce a novel data-driven quality heuristic to highlight these areas.

We believe this approach has several advantages over other SLF approaches:

- The SLF algorithm we present is fast and incremental. Since each image is processed as it is captured, the storage overhead is low, and new images can be incorporated at rates of more than one per second.

- The algorithm provides continuous feedback to the user. As a new image is incorporated into the surface light field, the rendering data structures are immediately updated and displayed, enabling the user to interactively evaluate the light field quality.

- We provide a data-driven quality heuristic to guide sampling. An error metric is interactively computed and displayed to aid in determining which parts of the 4D SLF space are undersampled.

- We provide a structured method for dealing with incomplete data. Due to occlusion, many surface patches are only partially visible. Rather than discarding these data, we use the current approximation to fill in these holes.

The core of our approach is the Online SVD (Brand, 2003), a fast memory-efficient algorithm for constructing an incremental low-rank Singular Value Decomposition. The next sections describe the mathematical background of the SVD, then proceed to describe the Online SVD. In Section 3.5, we present details of the implementation, followed by results and conclusions.

3.2 Mathematical Background

Singular Value Decomposition was used by Chen et al. (Chen et al., 2002) to build a compressed representation of the SLF which could be rendered at interactive rates. In this section I describe the mathematical background of the SVD.

3.2.1 Eigenvectors and Eigenvalues

Both the Singular Value Decomposition and Principal Component Analysis are rooted in the concept of eigenvectors and eigenvalues, which are the solutions to the equation

$$Ax = \lambda x$$

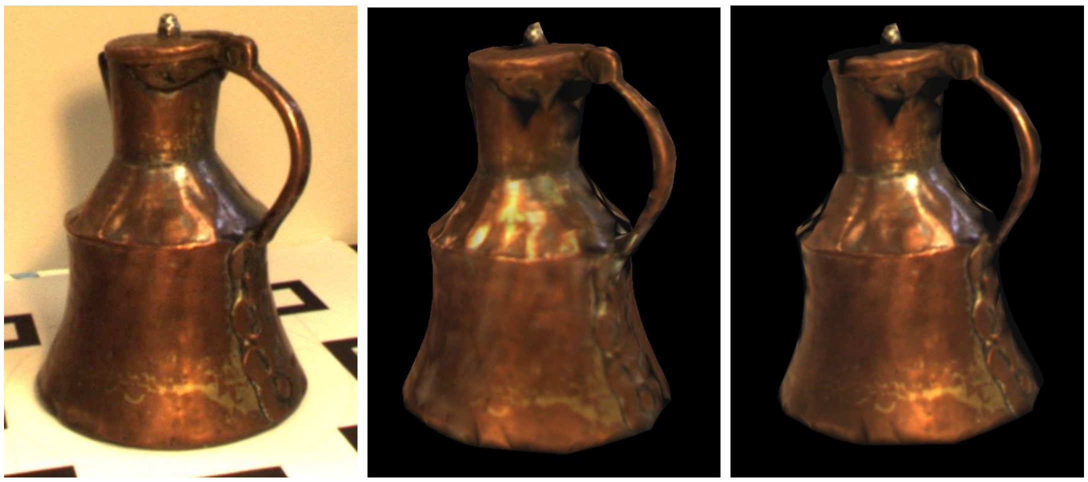


Figure 3.2: This sequence shows the image acquisition process. The left frame shows an image from the camera. In the middle is the reconstruction before this image is incorporated, with highlights interpolated from nearby views. The right image shows the reconstruction after it is incorporated, with the correct highlights.

where $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^{n \times 1}$, and $\lambda \in \mathbb{R}$. This equation says that for certain special vectors x (the *eigenvectors*), the general transformation matrix A only scales them by a factor (the *eigenvalues*) and does not rotate them. Every $n \times n$ square matrix has n (not necessarily distinct) eigenvectors. They can be found by rearranging the terms of the above equation:

$$Ax - \lambda x = 0$$

$$(A - \lambda I)x = 0$$

We don't consider the 0 vector to be an eigenvector, so the above equation is true when

$$det(A - \lambda I) = 0$$

This equation is called the *characteristic polynomial*, which is a set of n linear equations which can be solved for λ to determine the eigenvalues. The eigenvalues are substituted back into the matrix equation to get the eigenvectors. In practice, this is only done for small matrices due to the computational complexity. For larger matrices, an iterative algorithm based on the characteristic polynomial can be used.

3.2.2 Power Iteration Algorithm

The characteristic polynomial can be used to develop an algorithm for computing the eigenvectors and eigenvalues. This algorithm, which is known as *Power Iteration*, takes advantage of the property that vectors which are transformed by the matrix A will be scaled in the direction of the largest eigenvector. Initially, the vector is chosen to be some random non-zero vector. This vector is iteratively multiplied by the matrix, which causes it to gradually align with the largest eigenvector. This is repeated until the vector has converged. The eigenvector and eigenvalue are stored, and the eigenvector's contribution is subtracted from the matrix. The process is then repeated to compute the next largest eigenvector. Here is the pseudocode for determining an eigenvector:

```
$x_i = \text{rand}(n,1)$
$x_i = x_i / \|x_i\|$
**repeat**
    $x_{i-1} = x_i$
    $x_i = Ax_i$
    $x_i = x_i / \|x_i\|$
**until** $\|x_i - x_{i-1}\| > threshold$
$\lambda_i = \|x_i\|$
```

To compute the subsequent eigenvalues, the eigenvector is removed from A in a process called *Deflation*. The outer product of the eigenvector is subtracted from the matrix:

$A_{i+1} = A - \lambda_i x_i x_i^T$

The performance of Power Iteration is highly-dependent on the eigenvalues. The convergence ratio is controlled by the ratio of subsequent eigenvalues $\frac{\lambda_i}{\lambda_{i+1}}$. If this ratio is large, the algorithm will converge rapidly. Thus Power Iteration is usually used only when the first few eigenvalues need to be computed, since these are usually well-separated. Power Iteration was used by Chen et al. (Chen et al., 2002) to determine the eigenvectors of the SLF.

3.2.3 Principal Component Analysis

PCA is a widely used data compression technique in statistics and data modeling (Press et al., 1992). The PCA is computed by determining the eigenvectors and eigenvalues of the *covariance matrix*. The covariance of two random variables is their tendency to vary together. This is expressed as:

$$\text{cov}(X, Y) = E[E[X] - X] \cdot E[E[Y] - Y]$$

where $E[X]$ denotes the expected value of X. For sampled data this can be explicitly written out as:

$$\mathrm{cov}(X,Y) = \sum_{i=1}^{N} \frac{(x_i - \bar{x})(y_i - \bar{y})}{N}$$

with $\bar{x} =$ mean(X) and $\bar{y} =$ mean(Y). Note that $\mathrm{cov}(X,X) = \mathrm{var}(X)$, and for independent variables $\mathrm{cov}(X,Y) = 0$. The covariance matrix is a matrix A with elements $A_{i,j} = \mathrm{cov}(i,j)$. The covariance matrix is square and symmetric. For independent variables, the covariance matrix will be a diagonal matrix with the variances along the diagonal.

Calculating the covariance matrix from a dataset first requires centering the data by subtracting the mean of each sample vector. Considering the columns of the data matrix A as the sample vectors, we can write the elements of the covariance matrix C as:

$$c_{ij} = \frac{1}{N} \sum_{i=1}^{N} a_{ij} a_{ji}$$

written in matrix form:

$$C = \frac{1}{N} A A^T$$

Often the scale factor $1/N$ is distributed throughout the matrix and the covariance matrix is written simply as AA^T.

The eigenvectors of the covariance matrix are the axes of maximum variance. The PCA technique is effective because for many datasets the majority of the data variance can be captured by a small subset of the eigenvectors. An example from a SLF dataset is shown in Figure 3.3. Note that the magnitude of the eigenvalues falls off quickly, such that 70% of the value is contained in the first 15 eigenvectors. This implies that a good approximation of the full matrix can be computed using only a subset of the eigenvectors and eigenvalues. To approximate the matrix, the eigenvalues are truncated below some threshold. The data is then reprojected onto the remaining r eigenvectors to get a *rank-r* approximation. This can be shown to be optimal compression in the least-squares sense, i.e. it is the closest approximation to the full matrix that can be achieved using r terms.

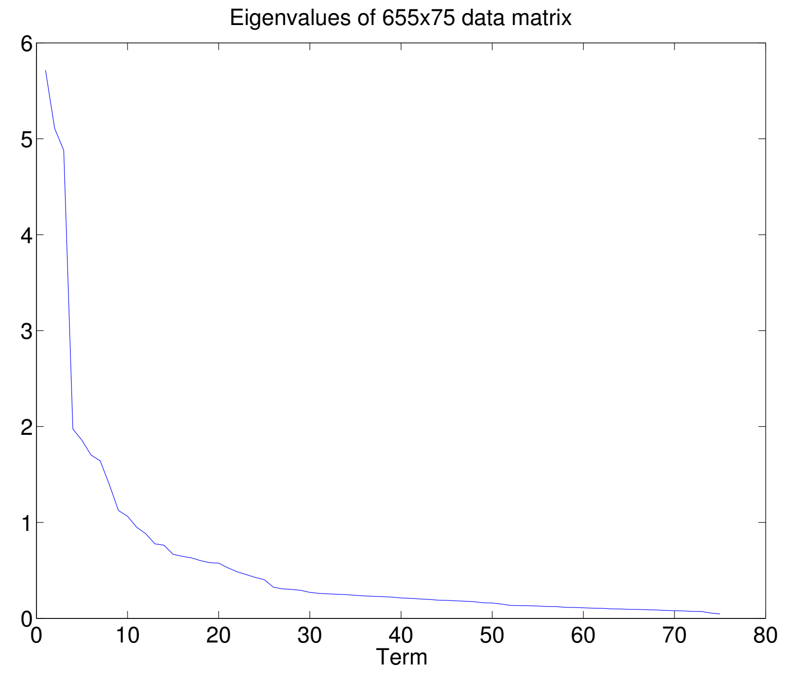


Figure 3.3: The eigenvalues of a 655x75 matrix from a SLF dataset. The mass of the eigenvalues are concentrated in the first dozen terms, which implies that a good approximation of the full matrix can be computed using only a subset of the eigenvectors and eigenvalues.

3.2.4 Singular Value Decomposition

The Singular Value Decomposition (SVD) is a technique for decomposing a matrix into a set of rotation matrices and a scale matrix. The form is:

$$A = USV^T$$

where $A \in \mathbb{R}^{m \times n}$ (with $m >= n$), $U \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$, and S is a diagonal matrix of size $\mathbb{R}^{n \times n}$. Both U and V are orthogonal.

The SVD is closely related to PCA and to eigenvalue computation. Recall from above the eigenvalue equation:

$$Ax = \lambda x$$

In order to compute the eigenvalues, A must be a square matrix. The SVD is less restrictive in that it can be performed on any $m \times n$ matrix. The singular values of a matrix A solve the equations

$$Au = \lambda v \text{ and } A^T v = \lambda u$$

The vectors u and v are known as the right- and left-singular vectors respectively. We can show the relation between SVD and eigenvalues through the following equations:

$$AA^T = (USV^T)(USV^T)^T = USV^TVSU^T = US^2U^T$$

$$A^TA = (USV^T)^T(USV^T) = VSU^TUSV^T = VS^2V^T$$

using the fact the U and V are orthogonal so $U^T = U^{-1}$. This shows that U and V can be calculated as the eigenvectors of AA^T and A^TA respectively. The square root of the eigenvalues are the singular values along the diagonal of the S matrix.

The advantage of the SVD is that there are a number of algorithms for computing the SVD. One method is to compute V and S by diagonalizing A^TA:

$$A^TA = VS^2V^T$$

and then to calculate U as:

$$U = AVS^{-1}$$

There are also a number of efficient algorithms which minimize error propagation such as the Golub–Reinsch SVD and the Lanzcos SVD (Golub and Loan, 1996). The

SVD is a powerful and widely-used compression technique, but it requires that the full set of data be available during processing. Since these data are usually extensively resampled to fit the columns and rows of the data matrices, this can be a significant storage cost. In the next section, we describe an algorithm which can incrementally compute the SVD.

3.3 Online SVD

The Online Singular Value Decomposition (Brand, 2003) is an incremental SVD algorithm (Hall et al., 2000; Chandrasekaren et al., 1997) that computes the principal eigenvectors of a matrix without storing the entire matrix in memory. The results are built up from a series of simple operations on the output eigenvectors, which are low-rank approximations to the full matrix. If the rank r is much smaller than the size of the matrices, this is a considerable saving, and reduces the computational complexity from quadratic to linear (Brand, 2003).

The Online SVD works as follows (see Figure 3.4). Consider a rank-r SVD

$$A = USV^T$$

where $U \in \mathbb{R}^{m\times r}$, $S \in \mathbb{R}^{r\times r}$, and $V \in \mathbb{R}^{n\times r}$. Each new column of samples c is projected onto the eigenspace:

$$j = U^T c$$

The amount that is orthogonal to the eigenspace is given by:

$$p = c - Uj$$

The norm of this vector, $\|p\|$, is a measure of how different the pixels in this new image are from our current approximation. If the pixels are similar (that is, $\|p\|$ is below a threshold) we can incorporate this new sample by simply rotating the existing eigenspaces.

$$U' = UR_U \qquad V' = VR_V$$

Otherwise, the current rank r of the approximation is insufficient, and we increase

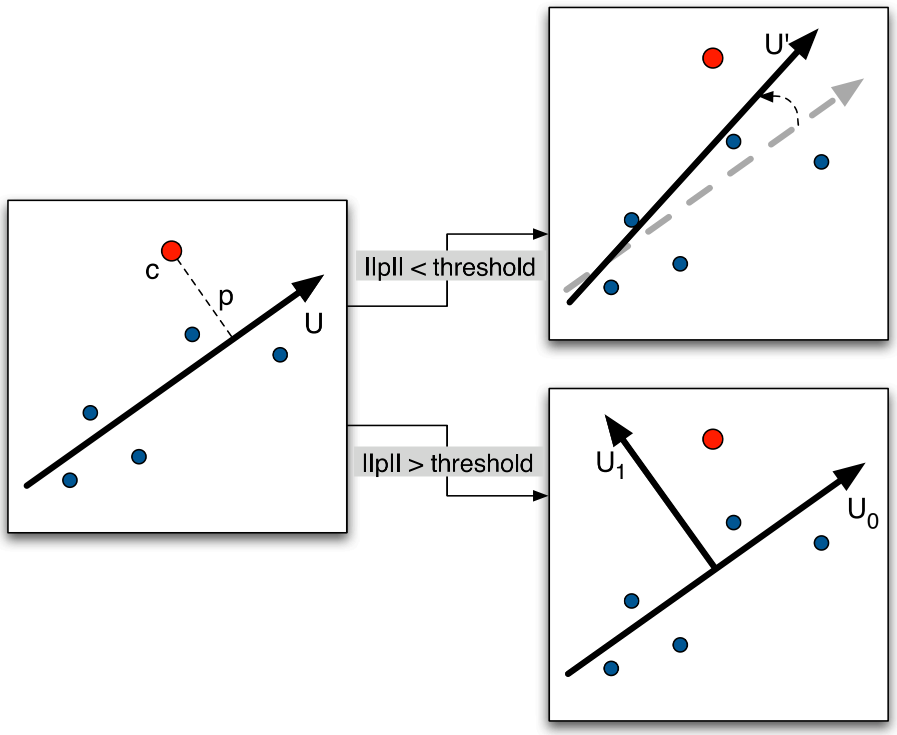


Figure 3.4: The Online SVD. A new point c is incorporated into the existing SVD. The orthogonal component p is computed by projecting onto U. If $\|p\|$ is below a threshold, then we can incorporate this new sample by simply rotating U. Otherwise, we must increase the rank of the approximation by adding another eigenvector that is orthogonal to the current set of eigenvectors.

the rank to $r+1$ and append the column j to our approximation.

$$U' = [U; j]\, R_U \qquad V' = V R_V$$

The rotations are computed by re-diagonalizing the $(r+1) \times (r+1)$ matrix

$$\begin{bmatrix} S & U^T c \\ 0 & \|p\| \end{bmatrix} \rightarrow [R_U, R_V] \tag{3.1}$$

These rotations can be computed in $O(r^2)$. Since only the output matrices U, S, and V are stored, this representation results in significant storage savings.

3.3.1 Error

The batch SVD can iterate over the entire dataset multiple times, while the Online SVD uses only the current approximation and the incoming data vector. Consequently, the Online SVD tends to have more error than a batch SVD for the same rank. Figure 3.5 shows the error as a function of rank. If the rank is too low to approximate the dataset, it will bias the Online SVD by forcing it to select sub-optimal eigenvectors. Brand (Brand, 2003) suggests computing the Online SVD at twice the desired rank and truncating, which allows the eigenvectors more degrees of freedom to fit to the incoming data. This increases the size of the working data structures, but does not affect the size of the rendering data structures.

3.3.2 Missing Values

Acquired radiance data is often incomplete because of occlusion. Many systems are forced to discard surface patches with missing data, or fill in the holes with incorrect values, such as zeros or mean values. A better approach is to estimate the missing data using a process known as *imputation*. Imputation uses the current Online SVD estimate of the light field to fill in missing values (Brand, 2003). The known samples are projected onto the current eigenspace, and the unknown values are estimated by solving the under-determined system using Linear Least Squares (Golub and Loan, 1996). This fills in the missing values with the nearest plausible values using the Mahalonobis metric (a metric defined in the scaled eigenspace) (Brand, 2003).

Suppose that an incoming column of data is partitioned into two components, c_+ and c_o, which represent the known and unknown components of the incoming image. For

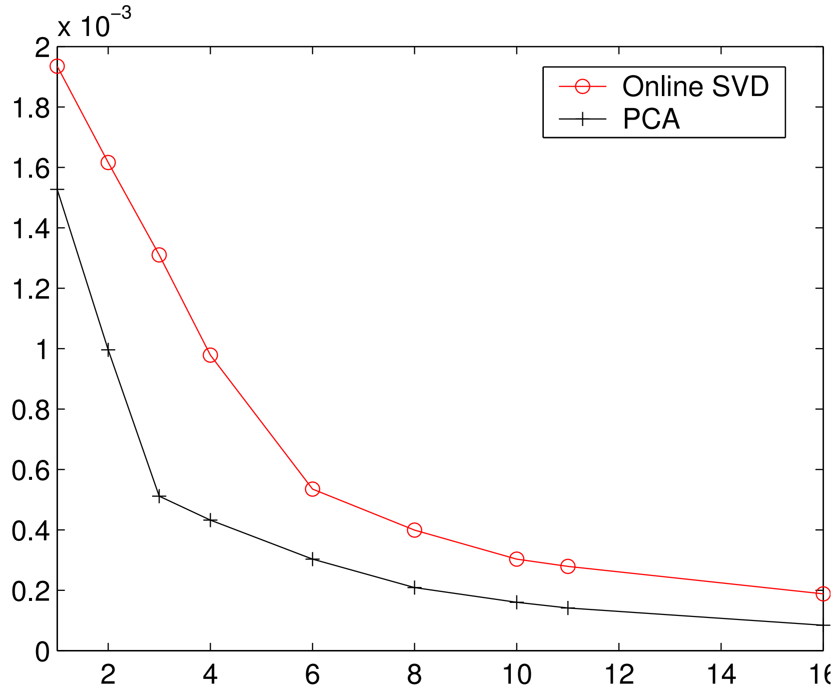


Figure 3.5: The Mean Squared Error of a reconstructed light field as a function of the rank. The majority of the light field is captured after 4-6 terms. For the same number of terms, the Online SVD has more error than the batch SVD.

example, c_o could represent the occluded region of a triangle. The current eigenspace is also partitioned into U_+ and U_o. We can set up two linear systems:

$$\begin{aligned} U_+Sb &= c_+ \\ U_oSb &= c_o \end{aligned}$$

These equations state that a vector b can be projected on the known subspace and the unknown subspace to get the known column vector and the unknown column vector. We can solve these equations using the method of Normal Equations (in (Golub and Loan, 1996) and discussed in Section 4.3.1). The first equation becomes:

$$b = (SU_+^T U_+ S)^{-1}(SU_+^T c_+)$$

This can be simplified using the fact that U is orthogonal:

$$b = (U_+S)^{-1}c_+$$

and substitute $U_oSk = c_o$

$$c_o = U_oS(U_+S)^{-1}c_+$$

This set of equations yields the full vector c which lies closest to the existing eigenspace in the Least Squares sense (for further details, consult (Brand, 2003)). The inverse is not a true matrix inverse, but is computed using the *pseudo-inverse* (discussed in Section 4.3.1). Substituting the above equations into Equation 3.1 yields:

$$\begin{bmatrix} S & U^Tc \\ 0 & \|k\| \end{bmatrix} = \begin{bmatrix} S & S(U_+S)^{-1}c_+ \\ 0 & \|c_+ - U_oS(U_+S)^{-1}c_+)\| \end{bmatrix}$$

Figure 3.6 shows the advantage of imputation for surface light fields. In practice, we only impute missing values when at least half of the surface patch is visible in an image. In addition, we can only impute values after 8-10 initial images have been processed, which allows the system to establish a reasonable approximation. A different approach to handle missing data is discussed in Chapter 4.

3.4 Data-Driven Quality Heuristic

The Online SVD enables the user to view the light field model as it is being captured. This visual feedback is helpful, but it can still be difficult to recognize undersampling errors during image acquisition. To aid the user in collecting high-quality radiance

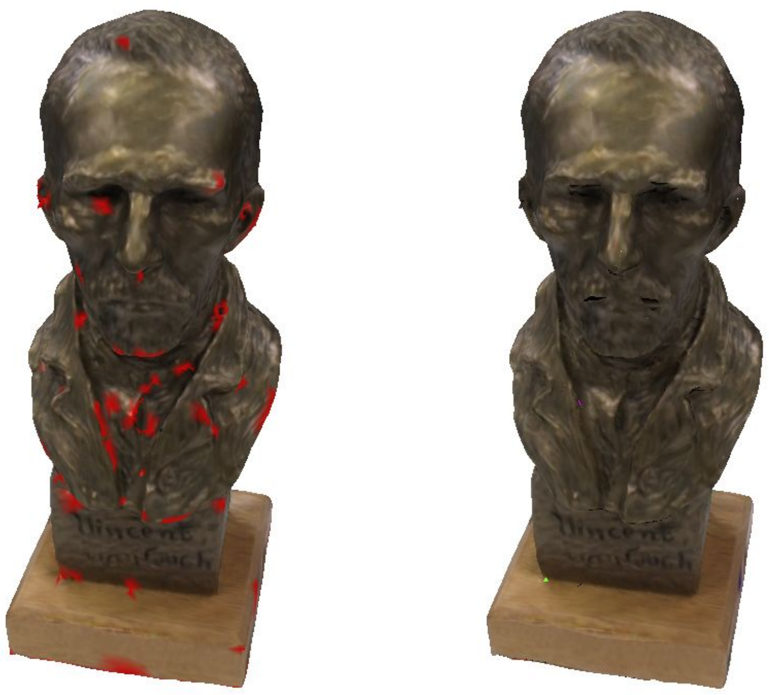


Figure 3.6: Left: The bust dataset with areas of missing data highlighted in red. Right: Imputation of the missing data fills in the areas with plausible values.

information, we developed a data-driven quality heuristic that uses the information obtained from the estimate to indicate whether more data are needed. This is displayed as the user views the model, and provides additional statistics about the reconstruction quality.

One possible way to do this is to assume a fixed BRDF and measure the error between this and the collected samples. This is the approach taken by Lensch (Lensch et al., 2003), who used an uncertainty minimization technique to guide image acquisition. We wanted to avoid this fixed BRDF assumption in order to capture a wide variety of reflectance properties.

There are two statistics that we want to provide feedback about: the variation over the surface, and the variation over the viewing direction. To this end, we developed a per-triangle scalar quality heuristic that is computed as the combination of two quality functions:

$$\psi(s,t,\theta,\phi) = \sqrt{\psi_p(s,t)^2 + \psi_h(\theta,\phi)^2}$$

The surface quality function ψ_p measures the quality of the surface approximation by tracking the projection error of the Online SVD. The projection error is computed as part of the updating step as the quantity $\|\mathbf{p}\|$. This value is a measure of how much

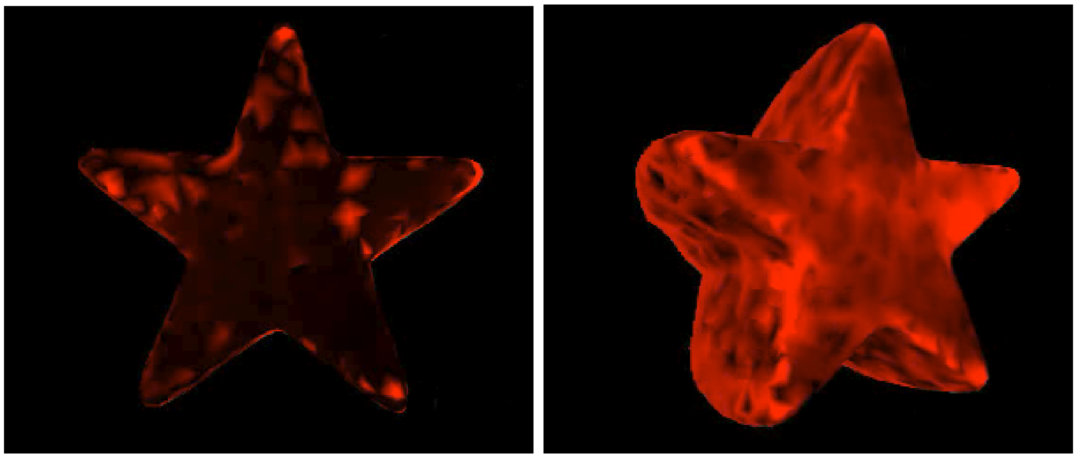


Figure 3.7: The data-driven quality heuristic $\psi(s,t,\theta,\phi)$ displayed on the star model. Left: The heuristic is dark in areas where images have already been captured, indicating that the area has been well-sampled. Right: In areas where more data is needed, the heuristic is bright to draw attention to the area.

the new image differs from our approximation. As the SVD refines, the projection error decreases. This scalar quantity is smoothed using an exponential fall-off filter.

The hemisphere quality function ψ_h is a measure of the sampling density of the hemisphere. The value is computed by using the areas of the triangles in the Delaunay triangulation of the hemisphere. As more images are captured, the areas of the triangles decrease. The Delaunay triangulation, which is used for interpolation, is described in more detail in Section 4.2.1.

The quality heuristic $\psi(s,t,\theta,\phi)$ is displayed as a scalar value at each triangle. We display this value in red in our system, since it is easily visible from a distance. An example in shown in Figure 3.7. As the user moves the tracked camera around the object, the object rotates on the screen and the brightness of the heuristic changes. As images are captured in a region, the heuristic darkens and the user moves to a different area. As the light field is being acquired, the user can either view the light field, the heuristic, or a split-screen view of both. In practice, we typically view the camera output and tracking information on one screen and switch between the heuristic and the light field on the other.

3.5 Implementation

The goal of this project is to build an interactive system that enables the rapid capture of exitant radiance under fixed illumination conditions. The radiance data is sampled

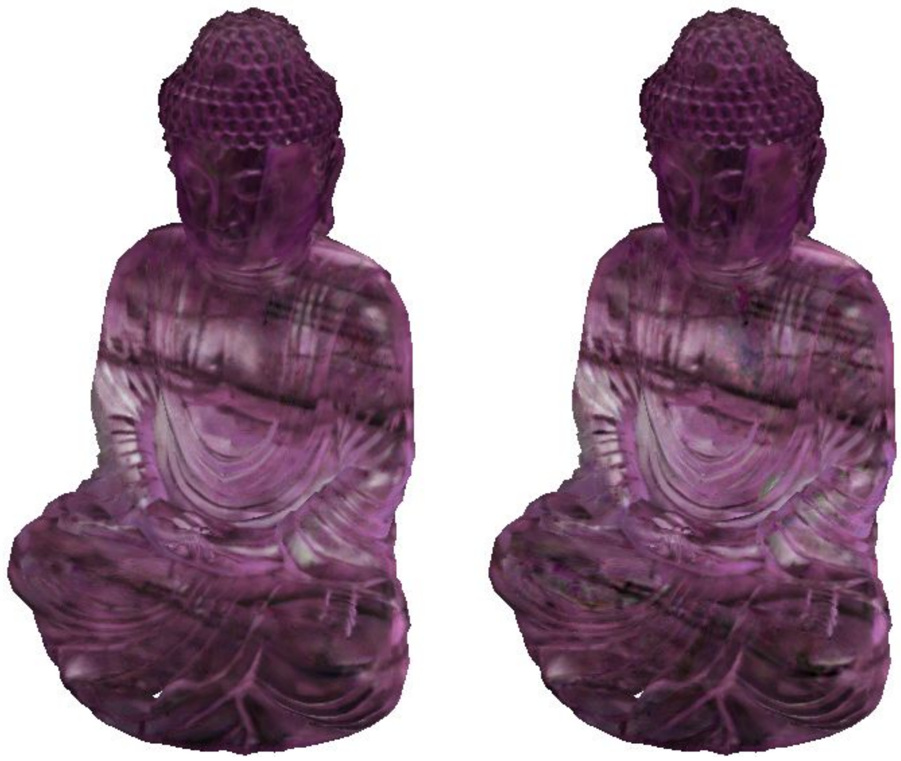


Figure 3.8: A side-by-side comparison of the Buddha model constructed with the OpenLF system (left) and our incremental approach (right). There are some slight differences noticeable in the chest area.

from the physical world and a compressed representation is generated such that it can be rendered at interactive rates on graphics hardware.

The Online SVD enables us to incrementally construct a low-rank approximation to a dataset. In this section we describe how to incorporate this tool into a SLF system. I will first give an overview of the entire SLF capture process, then describe some of the particular components of our system.

3.5.1 The Surface Light Field Construction Process

The user interacts with our system by moving a video camera around the object whose appearance is to be captured. The video camera is tracked in real-time, and the images are incorporated into the model and displayed on the screen. This enables the user to view the surface light field as it is being constructed and to correct for undersampling errors. As each new image is captured, it is resampled into a per-vertex column vector, which represents every pixel in a surface patch from one camera view. This image is incorporated into the Online SVD for each surface patch, and the rendering data structures are updated. The result can then be viewed on the screen, usually after only about 1 second. In addition, the user can view the data-driven quality heuristic as a scalar value for each surface patch, which provides additional statistics about the reconstruction quality.

This section discusses the entire SLF construction process, including the physical setup of the capture environment as well as the software tools that we used to build the systems. Many of the design choices were made for performance reasons to enable the user to view the result of each image acquisition as quickly as possible.

The capture process consists of

1. acquiring the geometry of the object,
2. acquiring a set of images of the object,
3. determining the 3D position and orientation of the camera with respect to the model,
4. determining the visibility of the surface patches,
5. sampling the values from the image,
6. incorporating the values into the representation, and
7. rendering novel images from this representation.

Steps 6 and 7 have been discussed previously in Sections 3.3 and 2.5. The other steps in this process are less complicated, and are described in this section. Since these steps are fairly well-understood in the computer vision community, we used several off-the-shelf components.

The geometry of the object is captured as a preprocess using a Faro$^{\mathrm{TM}}$digitizing arm. The 3D point samples (usually around 1000-3000 samples) are triangulated using the constrained 3D Delaunay mesh generator Triangle (Shewchuk, 1996). This triangulation is then loaded into Blender (Blender,) or MeshLab (MeshLab,) for cleanup and refinement. In addition to the 3D model points, the 3D locations of the fiducial markers are also acquired. This enables the model to be registered with the tracking system. An image of the scanning process along with the resulting mesh is shown in Figure 3.9.

The imaging device that we use is a Point Grey Research Flea$^{\mathrm{TM}}$video camera. This camera captures 1024×768 color images at 30 frames per second. The advantage of

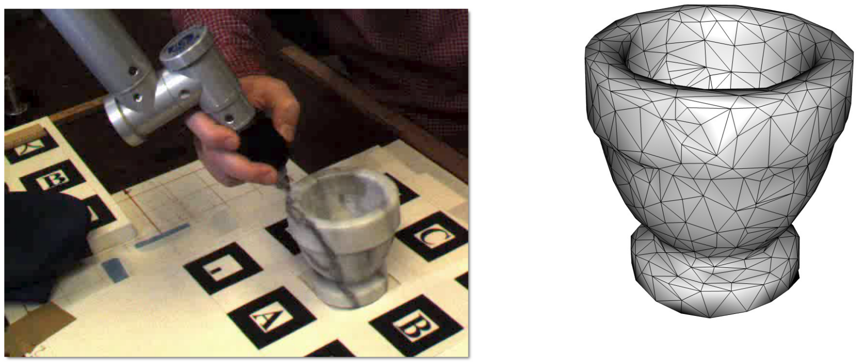


Figure 3.9: Acquiring the geometry of a marble pestle. Left: The object is scanned using a contact scanner. Right: The resulting mesh after triangulation and editing.

a video camera over a digital still camera is the higher speed of data transfer. The camera was calibrated with Bouguet's Camera Calibration Toolbox (Bouguet,), and images are rectified using Intel‘s Open Source Computer Vision Library (OpenCV,).

3.5.2 Pose Estimation

In order to project the image samples onto the geometry, the camera's position and orientation must be known. The 3D position and orientation of the camera is known as the *pose* of the camera, and the process of computing it from acquired images is known as *pose estimation*. To determine the pose of the camera with respect to the object, a stage was created with fiducials along the border. The 3D positions of the fiducials are located in the camera's coordinate system in real-time using the ARToolkit Library (Kato and Billinghurst, 1999). This library uses image segmentation, corner extraction, and matching techniques for tracking the fiducials. Knowing the 3D position of an imaged fiducial allows the pose of the camera to be computed. When multiple fiducials are present in an image, the camera pose can be refined by minimizing the differences between the pose estimates for each fiducial. A picture of our system showing the fiducials is in Figure 3.11.

We used several existing components of the OpenLF (OpenLF,) system, and modified several components to fit the new incremental approach. A diagram is shown in Figure 3.10. Like OpenLF, the visibility is needed to determine which surface patches to update. The OpenLF system uses a software algorithm to compute visibility, which we converted to graphics hardware for performance reasons. The mesh is first rendered from the point of view of the camera. The depth buffer is then read back to the CPU,

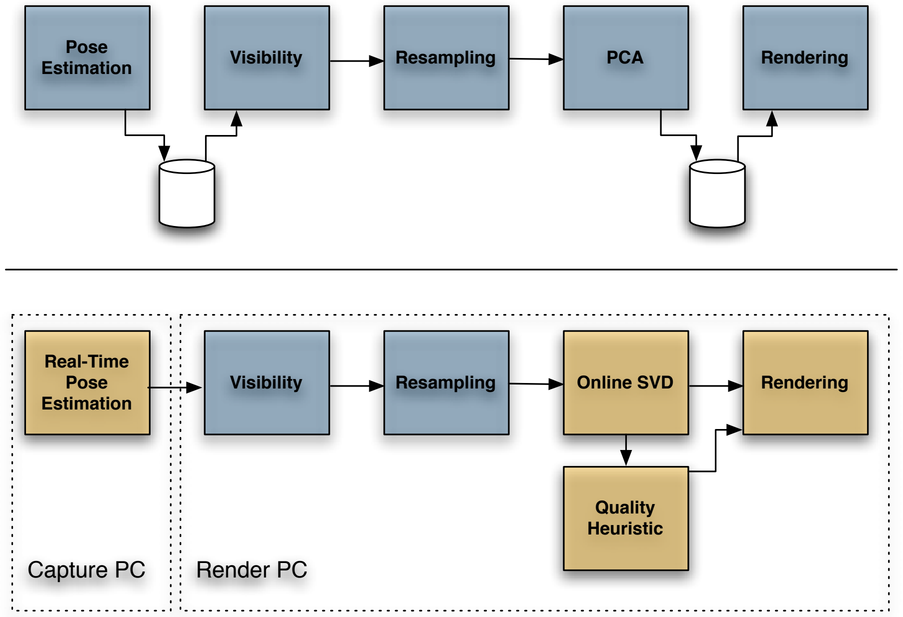


Figure 3.10: Top: The OpenLF system. Bottom: Our online system. The two systems share several components, including the visibility, resampling, and rendering. We introduce several new components to convert light field construction into an online process.

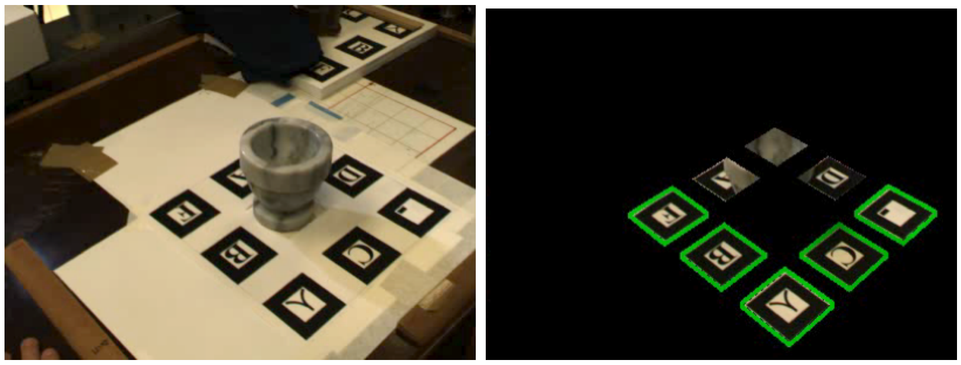


Figure 3.11: The pose estimation output. Left: The output from the video camera, showing the object and the tracking fiducials. Right: The tracking fiducials are segmented from the image and the pose estimation is computed.

where it is compared against the projected depth of each vertex. If the vertex passes the depth test, it samples a color from the projected position on the input image.

The view map construction was also modified to fit into the incremental framework and to use graphics hardware. The OpenLF system uses Delaunay triangulation to resample the view locations into the hemisphere, which we convert to an incremental Delaunay triangulation. The triangulation is initialized with four points at the corners of the view map. When a new image is captured, the camera location is projected onto the vertex basis vectors and normalized to get a point on the hemisphere. This point is projected down onto the plane and inserted into the Delaunay triangulation. The triangulation is then rendered using these points as the colors, and the graphics hardware resamples the colors across the hemisphere. The result is stored in the view map.

3.5.3 Computer Hardware

Our system uses two PCs, one for camera tracking and one for visibility, resampling, and rendering. The tracking PC, a 1.8GHz Intel Pentium 4, is connected to the camera via a IEEE-1394 interface. The images and pose estimates are sent over the network to the second PC, a 2.3GHz AMD Athlon64, where they are incorporated into the surface light field and rendered.

The systems presented in this thesis all use graphics hardware to accelerate the SLF reconstruction and rendering. GPUs are powerful and can handle large amounts of data, which enables the SLF to be rendered at interactive speeds. However, the programming model of GPUs imposes certain constraints on algorithm design. I will discuss these constraints in more detail in the next chapters.

The texture lookup on graphics hardware is more efficient when dealing with several large textures than it is with numerous small textures. The surface maps and view maps for all of the triangles are tiled into larger texture maps, as shown in Figure 3.12. We typically use surface maps of size 16x16 and view maps of size 32x32. All textures are stored in floating point format, which avoids quantization error and scale/bias artifacts. This requires more storage than the 8-bit textures used in OpenLF (OpenLF,), but the signed representation and high dynamic range are important as the principal components can be either negative or positive.

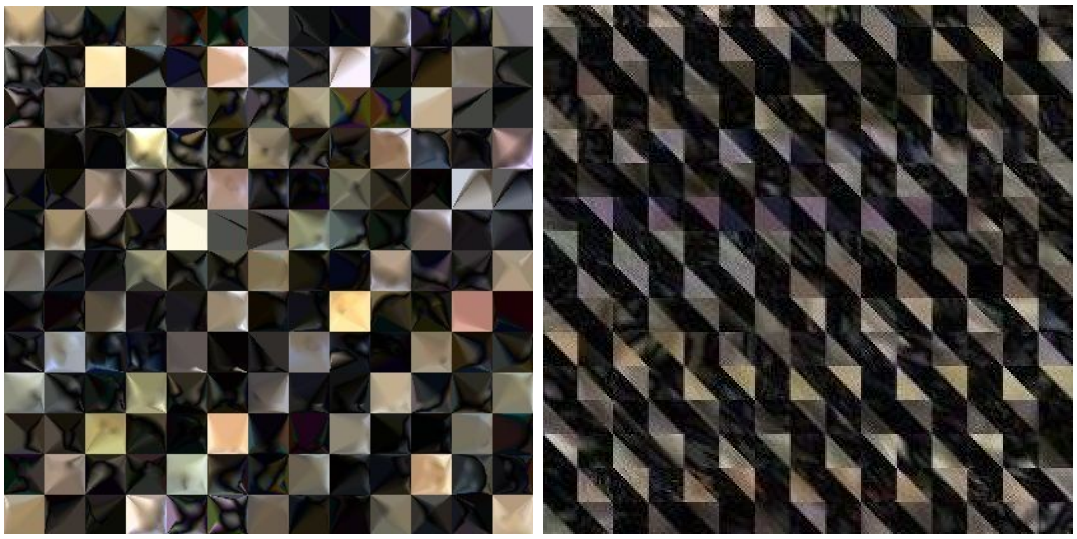


Figure 3.12: To improve the efficiency of the texture lookup on the graphics hardware, the numerous small textures are packed into larger textures. Left: The viewing functions $h(\theta, \phi)$ are parameterized over the hemisphere and stored in view maps. Right: The surface functions $g(u, v)$ are parameterized over the triangles and stored in surface maps.

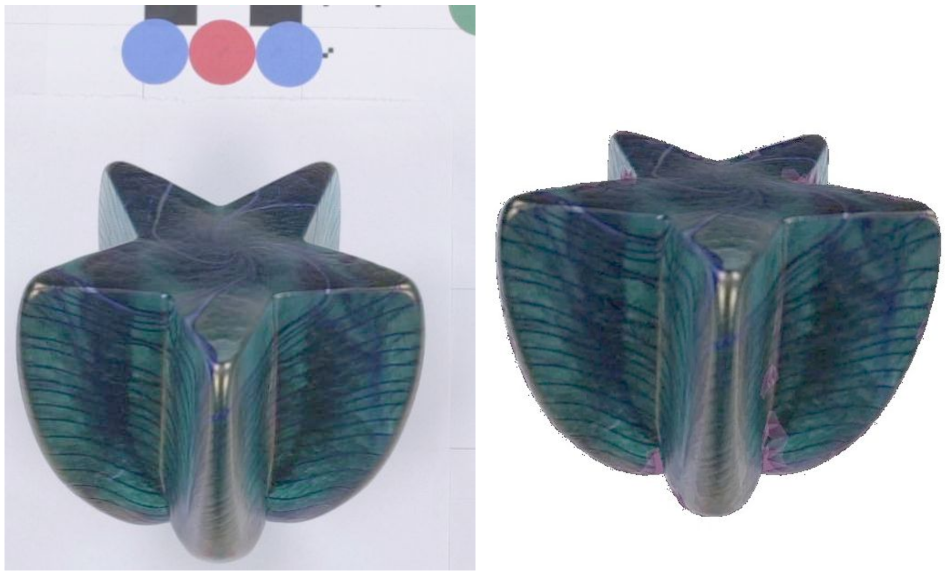


Figure 3.13: Left: A captured image of the star model. Right: The star model rendered in our system. Dataset courtesy of Intel Corporation.

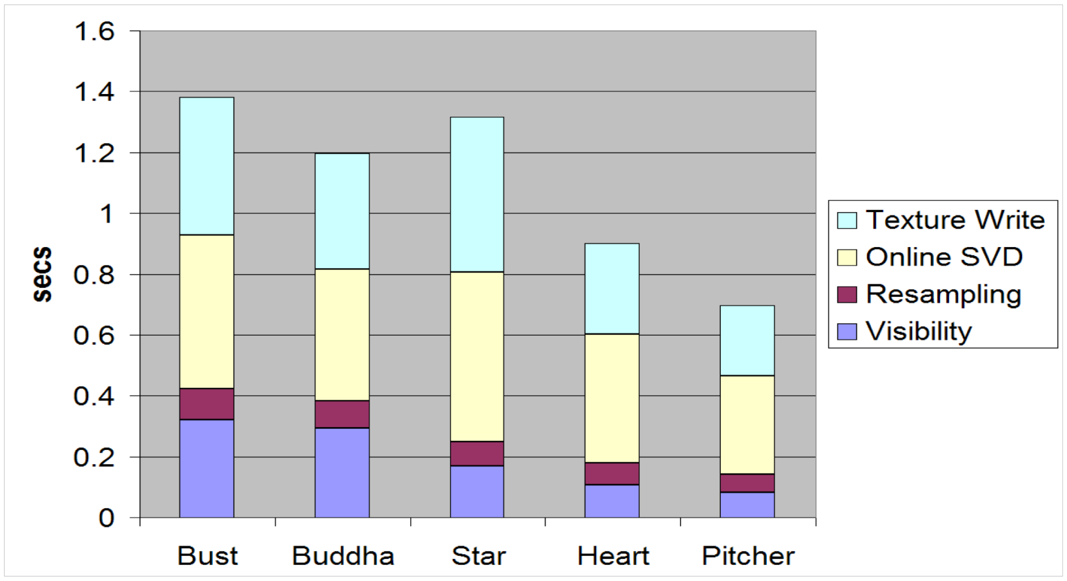


Figure 3.14: Timing results in seconds for several models. The bust has 7K triangles, the Buddha has 12K triangles, the star has 5K triangles, the heart has 2.7K triangles, and the pitcher has 3K triangles.

3.6 Results

One of the advantages of our method is the reduced storage cost, which translates to improved speed of model construction. In our tests, this resulted in an order of magnitude increase in speed over the OpenLF system. For the Buddha dataset shown in Figure 3.8, the processing time is reduced from 67 minutes in OpenLF to 7 minutes in our system. Timings for several datasets are presented in Figure 3.14.

As we noted in Sec. 3.3.1, the Online SVD can have more error than the PCA. We ran several experiments to compare the quality of the Online SVD to a batch PCA. Using a stored dataset, the images were fed one at a time through the Online SVD solver to mimic the conditions of an incremental construction. Since this was implemented within the OpenLF framework, we can directly compare the quality of the reconstruction. A side-by side-comparison of two images is shown in Figure 3.8.

During image acquisition, the data-driven heuristic provides visual feedback to the user to highlight undersampled areas. We have found this works well in practice, but we are interested in formally evaluating the usefulness of this heuristic. In lieu of a user study, we developed an automated experiment to measure the convergence of a surface light field using the heuristic. Using an archived dataset, we compared random image selection to image selection guided by the heuristic. To add a new image, the computer calculates the heuristic from the point-of-view of all of the remaining images,

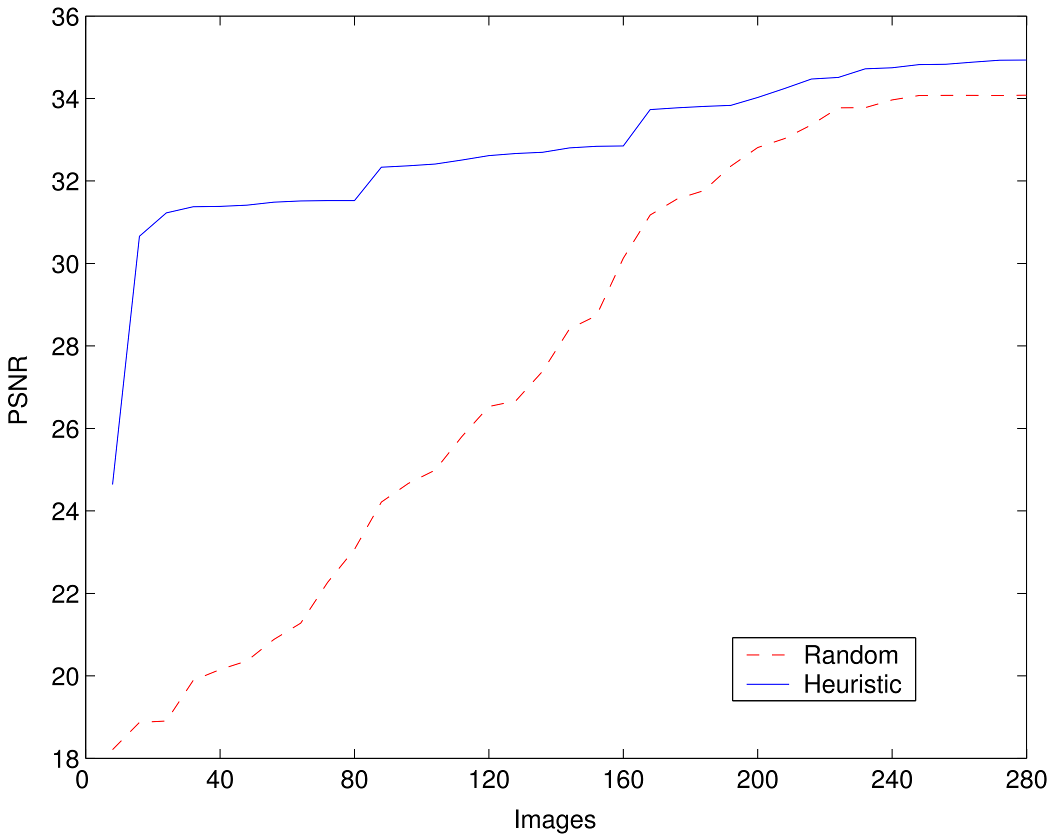


Figure 3.15: Measuring the convergence of surface light field construction using the data-driven heuristic. Compared to random selection, the use of the data-driven heuristic dramatically speeds convergence. The PSNR of the reconstruction was compared to a reference SVD implementation at 10 random viewpoints.

and selects the image with the highest error. The results of this experiment are shown in Figure 3.15. This experiment indicates that the heuristic can accurately predict which images will contribute the most content to the final result. Capturing images where the heuristic is large should likewise enable users to efficiently capture light fields. We are interested in conducting a user study to evaluate how well this heuristic information is communicated to the user.

One factor that can affect the quality of incremental approximations is dependence upon the ordering of data. In a different experiment, we measured the error over different permutations of a light field dataset. The results of this experiment are shown in Figure 3.16. For reference, we included the ground truth estimate from PCA and a sorted permutation of the samples. Note that the worst error is from the sorted permutation. Brand (Brand, 2003) points out that small, gradual rotations tend to amplify floating-point error, which causes the eigenvectors to lose orthogonality. He suggests

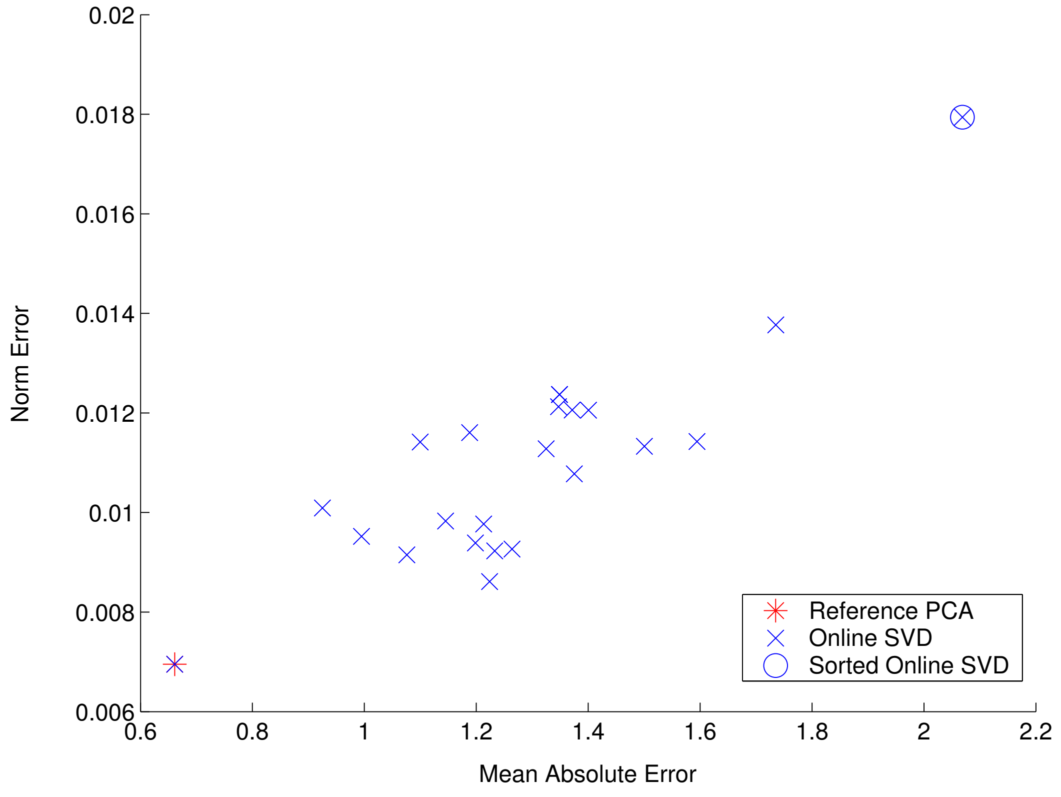


Figure 3.16: The total reconstruction error of the Online SVD from 20 random permutations of the original dataset. The reference PCA is shown in the lower left corner. A permutation of the SVD where the samples were sorted by norm is shown in the upper right corner. This indicates that the error is greatest when the samples are correlated.

periodically re-orthogonalizing the matrices using Gram-Schmidt orthonormalization. In a different paper (Brand, 2002), he provides an alternative matrix formulation that reduces this error. He also describes a method for "down-dating" the Online SVD to remove a previous column of data from the approximation. We are interested in exploring this techniques as it would allow the user to undo mistakes that occur during the image acquisition process.

3.7 Conclusions

In this chapter I discussed a method for incremental construction of surface light fields, which uses the Online SVD to build a surface light field approximation interactively during the acquisition stage. This online approach provides real-time feedback to the user, which enables the user to direct the image acquisition towards the undersampled areas of the model. This significantly reduces the acquisition time and helps build higher quality models. This approach can also decrease the processing time by an order of magnitude. I believe that incremental construction of surface light fields is a powerful tool for the capture and rendering of photorealistic models.

To assist the user in the capture process, I present a novel data-driven heuristic that provides an extra channel of information. I conducted several experiments to demonstrate that the data-driven quality heuristic, a tool for providing feedback to the user during sampling, can significantly increase the convergence of the reconstruction. Since the order in which the images are acquired affects the Online SVD solution, I also investigated the errors caused by permutations of the images.

I also discussed a method for imputing missing values from the data by using the current Online SVD approximation. This method worked reasonably well for our experiments, but it required extra computation and could result in artifacts. In the next chapter I discuss a different approach for dealing with the problem of missing data in SLF construction.

CHAPTER 4

Scattered Data Approximation

IBR techniques can represent the appearance of a wide variety of real physical surfaces with complex reflectance behavior. The challenges with IBR are handling the large amount of data, rendering the data efficiently, and previewing the model as it is being constructed. In this chapter, I describe the Incremental Weighted Least Squares approach to the representation and rendering of spatially-varying view-dependent anisotropic illumination. The appearance of the object is represented by a set of low-degree polynomials called WLS centers. During rendering, these polynomials are combined in a non-linear fashion to generate a full reconstruction of the exitant radiance. The rendering algorithm is fast, efficient, and implemented entirely on the GPU. The construction algorithm is incremental, which means that images are processed immediately as they are captured instead of in the traditional batch fashion. This human-in-the-loop process enables the user to preview the model as it is being constructed and to adapt to over-sampling and under-sampling of the surface appearance.

4.1 Introduction

As we saw in the previous chapter, the matrix factorization approach to SLFs has many advantages, including high compression rates and fast rendering. However, the matrix factorization step requires a complete and regularly sampled set of data, which necessitates an expensive resampling step. This is a problem when data is missing or sparse, which is often the case with SLF data.

A better approach is to treat SLF construction as a data approximation problem. The exitant radiance at each surface patch is represented as a function, and the input images are treated as samples from this function. Since there are no restrictions imposed on the geometry of the object or on the camera locations, the input samples are located at arbitrary positions. This is an example of a problem known as scattered data approximation (Wendland, 2005).

A common scattered data approximation technique is Radial Basis Functions (RBFs). Zickler et al. (Zickler et al., 2005) demonstrated the ability of RBFs to accurately reconstruct sparse reflectance data. Constructing this approximation requires a global technique, since every point in the reconstruction influences every other point. This is a disadvantage for an incremental algorithm, as every value must be recomputed when a new sample arrives. It is also difficult to render efficiently on graphics hardware, since many RBF algorithms rely on Fast Multipole Methods to reduce the size of the computation (Carr et al., 2001). We would like a method which has the scattered data representation ability of RBFs, but without the complex updating and reconstruction.

In this chapter I discuss Incremental WLS, a fast, efficient, and incremental algorithm for the representation and rendering of surface light fields. It is a non-linear polynomial approximation for multi-variate data, based on the idea of Least Squares polynomial approximation, which fits scattered data samples to a set of polynomial basis functions. WLS is similar to piecewise polynomial approximation and splines, except that the reconstruction is non-linear. WLS (Wendland, 2005) generalizes Least Squares by computing a set of approximations with associated weighting terms. These weighting terms can be either noise (in statistics) or distance (in graphics and computational geometry (Ohtake et al., 2003)). If we use distance, then WLS becomes a local approximation method. This local approximation is extended to a global approximation by computing the Partition of Unity (Shepard, 1968).

In this chapter I discuss several topics:

- I apply existing mathematical tools such as the Weighted Least Squares approximation technique by casting surface light fields as a scattered data approximation problem.

- I describe Incremental Weighted Least Squares, an incremental approach to surface light field construction which enables interactive previewing of the reconstruction.

- Using the Incremental WLS representation, I develop a real-time surface light field rendering algorithm, implemented on the GPU, which provides direct feedback about the quality of the surface light field.

The chapter proceeds as follows. I first describe several approaches to scattered data approximation, including piecewise polynomials and RBFs. In Section 4.3 I discuss Least Squares fitting and the extension to Weighted Least Squares. I then present

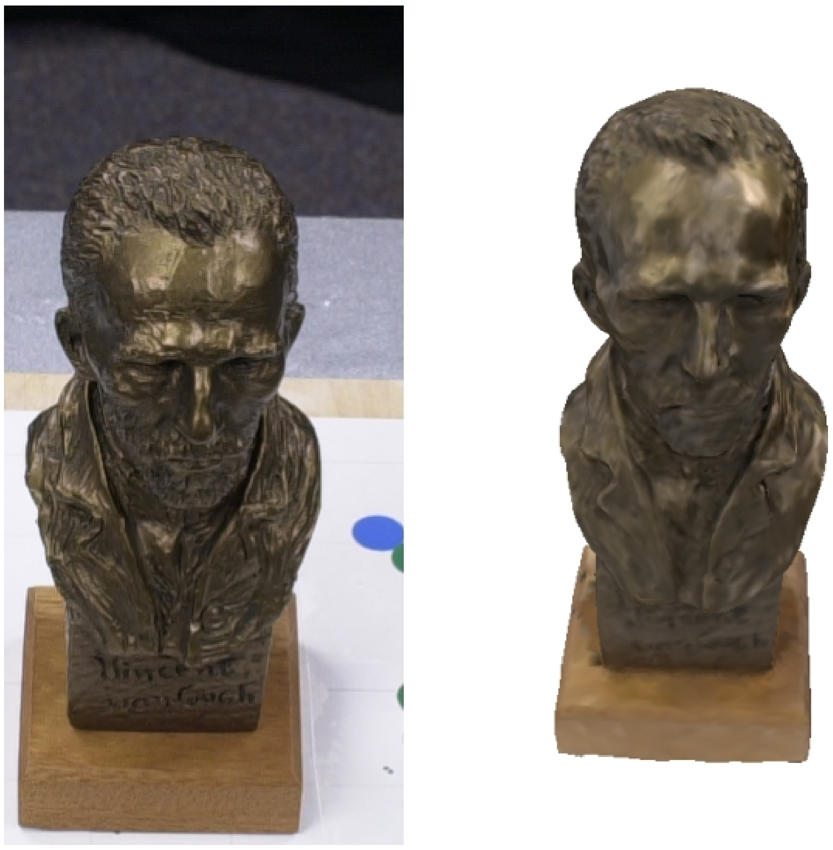


Figure 4.1: A side-by-side comparison of the WLS reconstruction with an input image that was not included in the training set.

Incremental WLS and describe how the WLS representation can be incrementally constructed and rendered. In Section 4.5 I discuss implementation details of the capture and rendering system, and then present results and conclusion.

4.2 Scattered Data Approximation

Scattered Data Approximations techniques take a set of samples in arbitrary locations and construct a function which is a reasonable fit to these samples. More formally, given a set of points $\mathcal{V} = \{\nu_0, \ldots, \nu_m\} \in \mathbb{R}^d$ and function values at these points $f_\nu \in \mathbb{R}$, we seek an approximant $f : \mathbb{R}^d \rightarrow \mathbb{R}$ which is as close as possible our data. In general, there are two types of approximants; *interpolating* and *approximating.* They are differentiated by their behavior at the sites ν. The residual at the sites is

$$\epsilon = \|f(\nu) - f_\nu\| \text{ for } \nu \in \mathcal{V}$$

For interpolating functions, $\epsilon = 0$ for all $\nu \in \mathcal{V}$. For approximating functions, the residual is minimized for the given set of basis functions.

4.2.1 Piecewise Polynomial

The most common scattered data approximation technique is to triangulate the data sites. This defines the regions where the polynomials lie as well as how they are joined at common vertices and edges. It is usually used for two-dimensional data, as higher dimensions are too difficult to triangulate. The quality of the spline approximation depends strongly on triangulation, so it is very important to get a good triangulation; long thin triangles will destroy the accuracy of the approximation.

The most commonly used triangulation is the *Delaunay triangulation* (Delaunay, 1934), which is the dual of the *Voronoi diagram.* The Voronoi diagram is a partitioning of the space into cells such that every point within the cell is closer to one site than any other site. This is written as:

$$T_y = \{x \in \mathbb{R}^2 \text{ s.t. } \|x - y\| = \min_{\nu} \|x - \nu\|, y \text{ and } \nu \in \mathcal{V}\}$$

The Delaunay triangulation is constructed from the Voronoi diagram by using the sites as vertices and connecting neighboring sites with edges.

A piece-wise linear interpolation over a Delaunay triangulation is defined by linearly interpolating the function values from the vertices. This can be extended to a piece-wise quadratic function by using additional information such as the mid-points of the edges. In general, extending this to higher-order continuity requires subdividing the triangulation.

4.2.2 Shepard's Method

Shepard's Method (Shepard, 1968) is a non-polynomial global multivariate interpolation scheme for scattered data. The advantage of global methods is that they define a single function $f(x)$ over all of $\mathbb{R}^d$, instead of a (potentially large) set of local piece-wise polynomial functions. Shepard's Method has the form:

$$f(x) = \frac{\sum_{\nu \in \mathcal{V}} f_\nu \phi(\|x - \nu\|)}{\sum_{\nu \in \mathcal{V}} \phi(\|x - \nu\|)}$$

The weight functions $\phi(r)$ are chosen to be large at the sites and to fall off away from the points. A common weight function is $\phi(r) = r^{-\mu}$. These radially-symmetric functions decay as the distance from the site increases, where the μ parameter controls the rate of decay. One problem with these functions is that they are not compact, so their support can extend globally. Consequently, another popular choice for weight function is the bounded exponential decay function:

$$\phi(r) = \begin{cases} \frac{\exp[-t^2/(t^2-r)^2]}{\exp[r^2/h^2]-1} & \text{if } r \leq t \\ 0 & \text{otherwise} \end{cases}$$

The user parameters consist of the support radius t and the decay rate h.

Shepard's Method is based on the *Partition of Unity* (Shepard, 1968), which is a general technique to weight a set of basis functions such that they sum to 1. The form of the Partition of Unity is:

$$f(x) = \frac{\sum b(x)\phi(x)}{\sum \phi(x)}$$

The Partition of Unity is a non-linear combination of the basis functions, which we will talk about further in Section 4.3.2. The disadvantage of the Partition of Unity is that it does not preserve the differentiability or continuity of the basis functions. For example, a major disadvantage of Shepard's Method is that it has stationary points (vanishing gradients) at all data sites. This is a problem for modeling the appearance of an object, since the quality of the highlights are dependent on the first and second derivatives.

4.2.3 Radial Basis Functions

Another common scattered data approximation technique is *Radial Basis Functions*(RBFs), which consists of a finite linear combination of translated basis functions. These basis functions are radially-symmetric functions of the form $\phi(\|\cdot\|)$, where $\|\cdot\|$ is the Euclidean norm. The general form of f is:

$$f(x) = \sum_k \lambda_k \phi(\|x - c_k\|)$$

where $\lambda_k \in \mathbb{R}$ are the coefficients of the basis functions and $c_k \in \mathbb{R}^d$ are known as the *centers*. For interpolating functions, the centers are defined to be the original sites $\mathcal{V}$. For approximating functions, the centers are optimized to minimize the residual.

One advantage of RBFs is that they are independent of dimension. The basis functions $\phi(r)$ take Euclidean distance as input, so they can be trivially extended to arbitrary dimensions. This is in contrast to piece-wise polynomial schemes, which usually rely on the tensor product of the dimensions.

The basis functions are chosen to emphasize certain desirable qualities such as continuity, hole-filling, or differentiability. Commonly used basis functions include:

$\phi(r) = r$	bi-harmonic
$\phi(r) = r^2 log\ r$	thin-plate splines
$\phi(r) = \sqrt{r^2 + c^2}$	multiquadrics
$\phi(r) = e^{-\alpha r^2}$	Gaussian

Once the basis functions are chosen, the coefficients can be computed by solving the linear system:

$$A\lambda = f$$

where $A = \{\phi(\|x_i - x_j\|\}$, and x_i and x_j are the centers. A is known as the *interpolation matrix* and is square, symmetric, and non-singular (Micchelli, 1986). This is one of the primary advantages of RBFs; unlike high-degree spline approximation or polynomial interpolation, RBF interpolation is always uniquely solvable.

Note that the biharmonic, thin-plate splines, and the multiquadrics all increase as the distance from the point increases, while the Gaussian falls off to zero. This behavior affects the form of the interpolation matrix. For the biharmonic function, the matrix will be very dense with a zero diagonal. Gaussian basis functions, which have local support, generate a matrix which is banded and sparse. This local support also means that adding a point only affects a small set of interpolation coefficients. Adding a point into a global RBF requires recomputing the entire set of interpolation coefficients.

The combination of local support and matrix form would seem to imply that Gaussians are a much better choice for basis function. However, it order to effectively use a local function such as a Gaussian the sampling density must be fairly uniform, which is not the case with many datasets. Global functions such as thin-plate splines and the multiquadrics are very good at filling holes in the sampling domain. These occur, for example, when reconstructing surfaces from laser-scanned data. Thus global basis functions tend to be more widely used, and research is focused on fast methods for solving the interpolation matrix (Carr et al., 2001).

4.3 Least Squares

Least Squares methods are a set of linear approximation techniques for scattered data. As before, we are given a set of N scalar samples $f_\nu \in \mathbb{R}$ at points $\nu \in \mathbb{R}^d$, and we want a globally-defined function $f(x)$ that best approximates the f_ν samples. For the approximation problem, the goal is to generate this function $f(x)$ such that the distance between the scalar data values f_ν and the function evaluated at the data points $f(\nu_i)$ is as small as possible. This is written as:

$$\min\{\sum_i \|f(\nu_i) - f_\nu\|\}$$

(Note: this discussion follows Nealen (Nealen, 2004)). Typically, $f(x)$ is a polynomial of degree m in d spatial dimensions. Thus $f(x)$ can be written as

$$f(x) = Bc$$

where $c = [c_1 \dots c_k]^T$ is the unknown coefficient vector. The B matrix is the polynomial basis matrix, and is composed of a set of basis functions $b(\vec{x})$. These functions are chosen based on the properties of the data and the dimensionality. In our case, we use either the 2D quadratic basis set

$$b(x, y) = c_1 + c_2x + c_3y + c_4x^2 + c_5xy + c_6y^2$$

or the 2D linear basis set

$$b(x, y) = c_1 + c_2x + c_3y$$

The polynomial basis matrix B is creating by stacking all of the k polynomial basis equations:

$$B = \begin{bmatrix} b_1(x, y) \\ b_2(x, y) \\ \dots \\ b_k(x, y) \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ & \dots & \\ 1 & x_k & y_k \end{bmatrix}$$

The data points are also written as a vector $f = [f_1(x) \dots f_k(x)]^T$. The fitting problem can now be stated as finding the coefficient vector c that minimizes the error between the polynomial approximation Bc and the data values f:

$$\min_c\{||Bc - f\|\}$$

This can be solved using the Method of Normal Equations (Johnson et al., 1993):

$$\begin{aligned} Bc &= f \\ B^T Bc &= B^T f \\ c &= [B^T B]^{-1} B^T f \end{aligned}$$

Solving the Normal Equation requires inverting the matrix $B^T B$. Since this inversion must be performed for every Least Squares fit, it is worthwhile to look into how this can be performed efficiently.

4.3.1 Solving the Normal Equations

The matrix inversion is possible because $B^T B$ is square and non-singular (Johnson et al., 1993). The size of the matrix depends upon the dimensionality d of the data and the degree k of the polynomial basis. For small matrices, it can be inverted directly. For larger matrices, I will discuss several efficient algorithms, including QR Decomposition and SVD. These are often implemented in matrix inversion packages such as BLAS (Blackford et al., 2002).

4.3.1.1 QR Decomposition

One problem with the Normal Equations is that the matrix $B^T B$ can be ill-conditioned, and thus subject to numerical precision errors when inverting. This can be addressed by using a technique called QR Decomposition. This technique decomposes a matrix B into an upper-trapezoidal matrix R and an orthogonal matrix Q such that $B = QR$. By substituting equivalence into the above Normal Equations and using the fact that $Q^T = Q^{-1}$:

$$\begin{aligned} c &= [B^T B]^{-1} B^T f \\ &= [(QR)^T QR]^{-1} (QR)^T f \\ &= [R^T Q^T QR]^{-1} R^T Q^T f \\ &= R^{-1} (R^T)^{-1} R^T Q^T f \\ &= R^{-1} Q^T f \end{aligned}$$

This formulation is much less prone to numerical precision errors. The QR Decomposition can be computed using three different techniques; Gram-Schmidt orthogonalization, Householder transformations, or Givens rotations. Householder transformations are most commonly used, but there are advantages and disadvantages to each of these. For more information on these techniques, see (Johnson et al., 1993).

4.3.1.2 Singular Value Decomposition

Another technique is to invert the matrix using the SVD. Recall from Section 3.2.4 that the SVD decomposes a matrix $A = USV^T$. U and V are orthogonal and S is a diagonal matrix with the elements $\sigma_1, \sigma_2, \ldots, \sigma_n$ along the diagonal. This leads to a simple method for computing A^{-1}:

$$\begin{aligned} A^{-1} &= [USV^T]^{-1} \\ &= [V^T]^{-1} S^{-1} U^{-1} \\ &= V S^{-1} U^T \end{aligned}$$

where S^{-1} is a diagonal matrix with the elements $1/\sigma_1, 1/\sigma_2, \ldots, 1/\sigma_n$ along the diagonal. When A is a square matrix, this computes the inverse of the matrix. When A is not square a true matrix inverse does not exist, and this is known as a *pseudo-inverse*. The *pseudo-inverse* A^+ is a generalization of the inverse which satisfies the linear equation $x = A^+ b$.

4.3.2 Local Least Squares Methods

One of the limitations of Least Squares fitting is that the solution encompasses the entire domain. This global complexity makes it difficult to handle large data sets or data sets with local high frequencies. We prefer a method that considers samples that are nearby as more important than samples that are far away. This can be accomplished by adding a distance-weighting term $\phi(r)$ to the Least Squares minimization. We are now trying to minimize the function

$$\min\{\sum_i \phi(\|x - x_i\|)\|f(x_i) - f_i\|\}$$

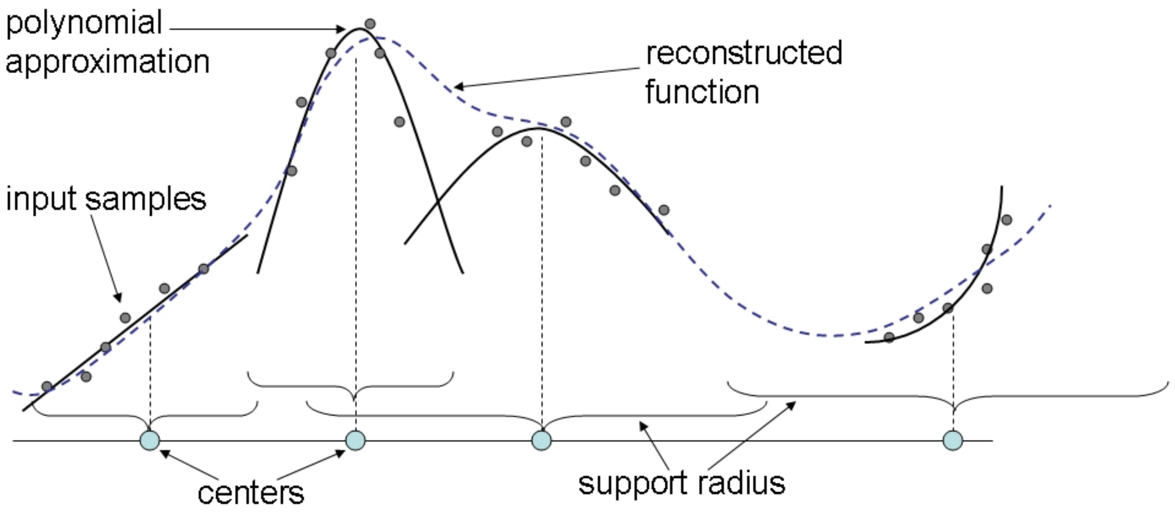


Figure 4.2: A diagram of the weighted least squares approach to function representation. Each center constructs a low-degree polynomial approximation based on samples in their neighborhood. These local approximations are then combined to form a global approximation.

Common choices for the distance-weighting basis functions $\phi(r)$ are the Wendland function (Wendland, 1995)

$$\phi(r) = (1 + \frac{4r}{h})(1 - \frac{r}{h})^4$$

and the interpolation function

$$\phi(r) = 1 + 2(r/h)^3 - 3(r/h)^2$$

which are both 1.0 at $d = 0$, and fall off to zero at the edges of the support radius h. These functions are illustrated in Figure 4.3.

It is interesting to note the connection with the Radial Basis Functions discussed in Section 4.2.3. In fact, we could use some of the other weighting functions that were discussed such as multiquadrics and thin-plate splines. However, these functions have infinite support and thus must be solved globally. The advantage of Weighted Least Squares is that instead of evaluating a single global approximation for all of the data samples, we create a set of local approximations. We then blend these local approximations together to get a global approximation. This provides us with local approximation behavior through the Least Squares fitting, which is useful because the data from the image capture can be noisy due to calibration errors.

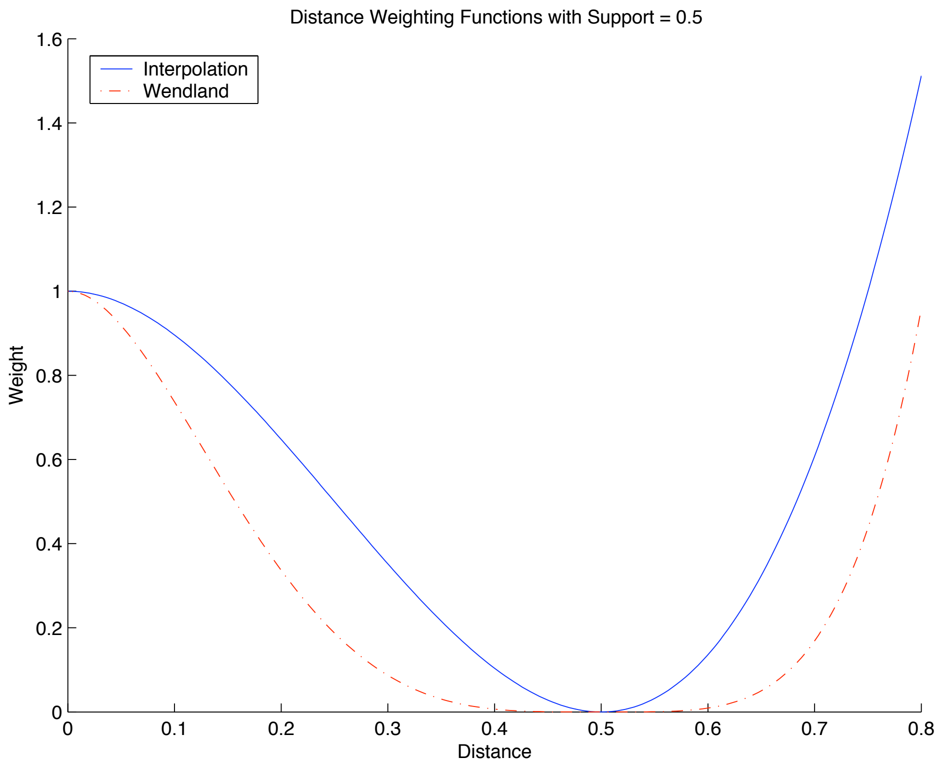


Figure 4.3: Two possible distance-weighting functions, with a variable support. Both functions are 1.0 at the center, and taper off to zero as the distance increases. While they are well-behaved within the support interval, they must be clamped outside this interval to avoid spurious weighting.

There are two methods for reconstructing a function using these distance weights. Moving Least Squares evaluates the Least Squares fit of the data points at every point in the domain. Weighted Least Squares evaluates the Least Squares fit at a small set of points called *centers*, and interpolates between these centers using the Partition of Unity. These centers are analogous to the concept of the centers of Radial Basis Functions. I will first discuss Moving Least Squares, then discuss Weighted Least Squares.

4.3.3 Moving Least Squares

Moving Least Squares (McLain, 1976; P. Lancaster, 1981) treats each point in the domain as a separate evaluation. This can be thought of as "moving" a Least Squares fit over the entire parameter domain to compute a new approximation everywhere. Moving Least Squares generates a continuously differentiable global function $f(x)$ if and only if the weighting function $\phi(r)$ is continuously differentiable (Levin, 1998). It consists of a set of local functions $f_x(x)$:

$$\min_{f_x}\{\sum_i \phi(\|x - x_i\|)\|f_x(x_i) - f_i\|\}$$

The functions $f_x(x)$ are constructed and evaluated at each point in the domain. The parameters of the basis functions can be adjusted to smooth the data, which makes it useful for reconstructing functions from noisy data (Alexa et al., 2003).

4.3.4 Weighted Least Squares

Since a neighborhood of points is required for evaluation, Moving Least Squares techniques require a fast query for the entire set of sample points, which would be difficult to implement on graphics hardware. Instead, we use Weighted Least Squares to compute a Least Squares fit at a small set of points, then interpolate using the Partition of Unity.

Each of the approximations is associated with a point $\bar{x}$, which is known as the center. At each of these centers, a low-degree polynomial approximation is computed using the distance-weighted samples x_i in the local neighborhood. This means that the coefficients are now a function of $\bar{x}$, and only defined locally around these centers. We can define a distance weighting vector $\Phi_{\bar{x}}$ with elements $\phi_i = \phi(\|\bar{x} - x_i\|)$.

The system is solved using Normal Equations:

$$c_{\bar{x}} = [\Phi_{\bar{x}} B^T B]^{-1} \Phi_{\bar{x}} B^T f$$

Expanding this out in equation form:

$$c(\bar{x}) = [\sum_i \phi(\|\bar{x} - x_i\|) b(x_i) b(x_i)^T]^{-1} \sum_i \phi(\|\bar{x} - x_i\|) b(x_i) f_i$$

We now have a set of local approximations at each center. During the reconstruction step, we need to combine these local approximations to form a global approximation. Since this global function is a weighted combination of the basis functions, it has the same continuity properties.

The first step in the Weighted Least Squares algorithm is to determine the m nearby local approximations that overlap this point. These functions are evaluated at the sample points and combined using a weight based on distance. However, the functions cannot just be added together, since the weights may not sum to 1.0. To get the proper weighting of the local approximations, the *Partition of Unity* (Shepard, 1968) is used to extend the local approximations to cover the entire domain. The Partition of Unity computes a new set of weights Θ_j by considering all of the m local approximations that overlap this point

$$\Theta_j(r) = \frac{\phi_j(r)}{\sum_{k=1}^{m} \phi_k(r)}$$

. The global approximation of this function is computed by summing the weighted local approximations:

$$f(x) = \sum_{j=1}^{m} \Theta_j(x) b(x)^T c(\bar{x}_j)$$

. This representation allows us to place the centers $\bar{x}$ at the most effective locations. This is important for incremental construction, which we discuss in the next section. We also discuss several strategies for center placement in Section 4.5.1.

Note that Weighted Least Squares is very similar to Shepard's Method, which we discussed in Section 4.2.2. Recall that Shepard's Method has the form:

$$f(x) = \frac{\sum f_\nu \phi(\|x - \nu\|)}{\sum \phi(\|x - \nu\|)}$$

There are several differences between the two methods. Weighted Least Squares uses a

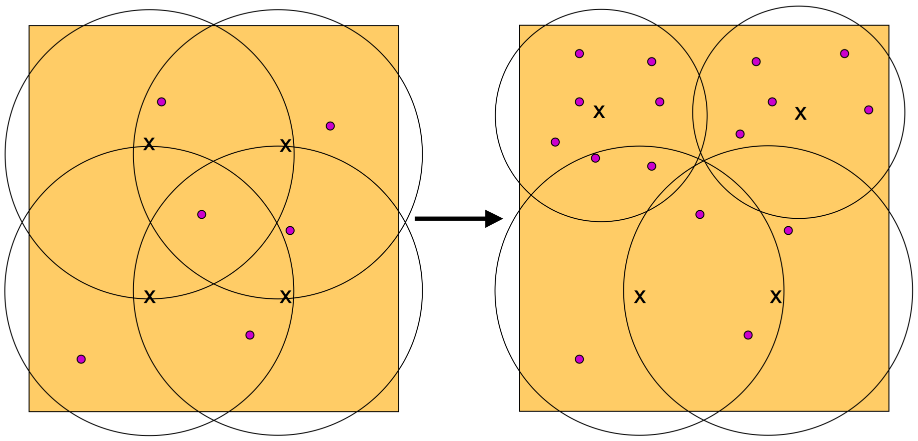


Figure 4.4: A diagram of the Adaptive construction of WLS. Initially, there are a fixed number of centers, each with a large domain. As new points are added to the data structure, they are classified according to which domains they fall within. When the number of points within a domain crosses a threshold, the domains are shrunk.

distance-weighted Least Squares approximation to the points, while Shepard's Method uses the distance-weighted points directly.

4.4 Incremental Weighted Least Squares

SLF representation can be treated as a batch process, by first collecting all of the images and then constructing and rendering the WLS representations. The advantage of batch processing is that all of the sample points are known at the time of construction, and the support radii and locations of the centers can be globally optimized.

The disadvantage to batch processing, as discussed in the previous chapter, is that it provides very little feedback to the user capturing the images. There is often no way to determine if the surface appearance is adequately sampled, and undersampled regions require recomputing the WLS representation. A better approach is to update the representation as it is being constructed, which allows the user to preview the model and adjust the sampling accordingly. In this section we describe two approaches to incrementally update the WLS approximation.

4.4.1 Adaptive Construction

The adaptive construction method starts with all of the centers having a large support radius. As new samples are generated, they are tested against the support radius of a

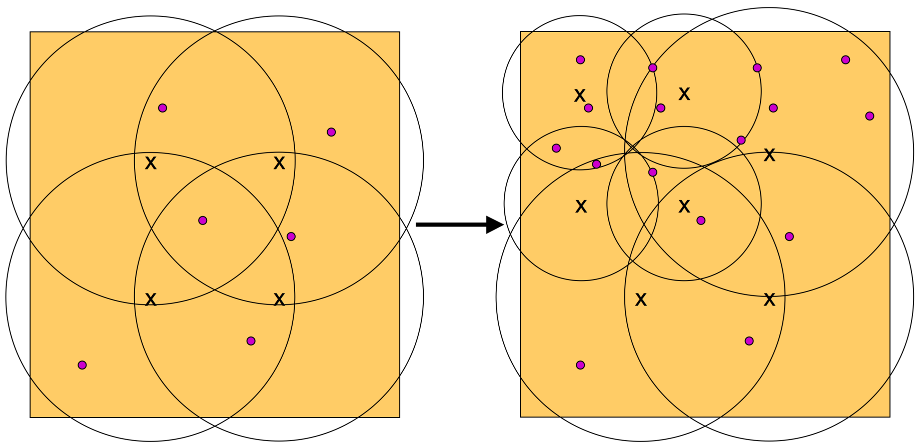


Figure 4.5: A diagram of the Hierarchical construction of WLS. Initially, there is a single center with a large domain. As points are added to the data structure, they are classified according to the domains they fall within. When the number of points within a domain crosses a threshold, the domain is subdivided into four smaller domains.

center, and added to that center's neighborhood list. Note that image samples can be members of more than one leaf node, since the support radii of the nodes can overlap. The WLS approximation is computed from the samples in the neighborhood list. As each center's list gets larger, the support radius is decreased, and samples are discarded if they no longer fall within the neighborhood. An example of this process is shown in Figure 4.4. In our implementation, this involves several user-defined parameters; we typically decrease the radius by 25% if the number of samples is more than 4 times the rank of the approximation.

4.4.2 Hierarchical Construction

The adaptive approach has the disadvantage that a single sample can cause the recomputation of numerous WLS coefficients, particularly in the initial phases when the support radii are large. The hierarchical approach avoids this computation by subdividing the domain as a quadtree. Initially, the representation consists of a single WLS center with a large support radius. When a new image sample arrives, the hierarchy is traversed until a leaf node is reached. The sample is deposited at the leaf node and the WLS approximation is recalculated. If the number of samples in a leaf node is larger than a pre-determining threshold, the leaf node is split into four children. Each child decreases its support radius and recomputes its WLS coefficients. An example of this process is shown in Figure 4.5. There are several user-defined parameters for this ap-

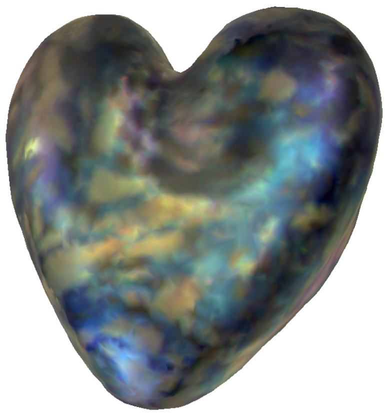


Figure 4.6: The heart object, rendered in our system.

proach; we have had good results splitting the nodes if the number of samples exceeds 4 times the rank of the approximation, and decreasing the area of the neighborhood by half (decreasing the radius by $1/\sqrt{2}$).

4.4.3 Rendering

WLS conforms well to the stream-processing model of modern graphics hardware. Each surface patch is independent and is calculated using a series of simple mathematical operations (polynomial reconstruction and Partition of Unity). More importantly, the local support of the WLS centers means that reconstruction only requires a few texture lookups in a small neighborhood.

After the centers and the WLS coefficients have been computed (using either the adaptive or the hierarchical technique), they are stored in a texture map for each surface patch. The coefficients are laid out in a grid pattern for fast access by the texture hardware. The adaptive centers are typically arranged in a grid pattern, but the hierarchical pattern must be fully expanded before it is saved to texture. This is done by propagating downward any leaf nodes that are not fully expanded.

During rendering, the viewpoint is projected onto the UV basis of the surface patch and used to index into the coefficient texture. The samples from the neighborhood around this element comprise the set of overlapping WLS approximations. Texture lookups are used to collect the neighboring centers and their coefficients. These polynomial coefficients are evaluated and weighted by their distance. The weights are computed using the Partition of Unity, which generates the final color for this surface patch.

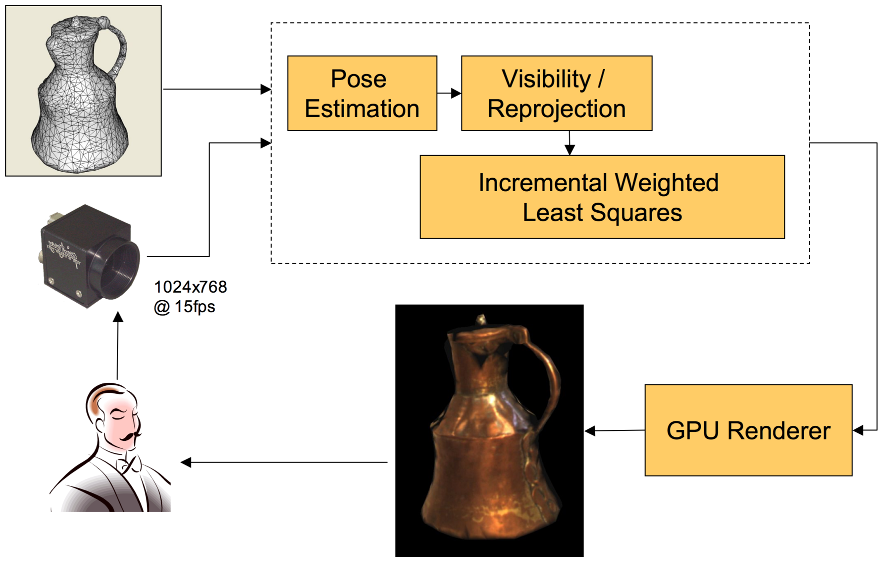


Figure 4.7: A diagram of the SLF capture and rendering system. Images are captured using a handheld video camera, and passed to the system. The pose of the camera is estimated using fiducials in the environment. Using the mesh information, visibility is computed and the surface locations are back-projected into the image. Each of these samples are incorporated into the Incremental Weighted Least Squares approximation, and sent to the card for rendering. The user can use this direct feedback to decide where to move the video camera to capture more images.

Once the color at each patch has been determined, we need a method to interpolate the colors across the model to smoothly blend between surface patches. One approach is to simply interpolate the colors directly. However this approach is incorrect, as it interpolates the values after the Partition of Unity normalization step. This generates artifacts similar to those encountered when linearly interpolating normal vectors across a triangle. The correct approach is to perform the normalization after the interpolation. For our system, we can accomplish this by interpolating the polynomial weights and colors independently, and using a fragment program to compute the Partition of Unity at every pixel.

4.5 Implementation

I have constructed a prototype system to demonstrate the ability of Incremental Weighted Least Squares to capture and render a wide variety of complex surface reflectance properties. The data structure and camera capture are managed on the CPU, while function reconstruction and rendering are handled by the GPU. This decouples the construction and rendering sub-systems and allows for free-form viewing while images are being incorporated. A diagram of the system in shown in Figure 4.7. The camera tracking, pose estimation, visibility, and image sampling steps are performed using the process described in Chapter 3.

Each surface patch is associated with a texture which stores the coefficients for one term of the polynomial basis. For all of the examples in this chapter we use a 3-term polynomial basis. We found that higher-order polynomial bases were susceptible to over-fitting (Geman et al., 1992). Over-fitting occurs when there are too few input samples, and the function tries to fit minor deviations in the data rather than the overall shape. The consequence is that reconstruction is very accurate at sample positions, but oscillates wildly around the edges. Using a lower-degree polynomial avoids this problem, at the cost of needing more centers to fit high-frequency data.

The positions of the surface patches, which are determined a priori, are represented as either vertices or texels. For most of the models we use vertices, and the renderer uses a vertex texture fetch to associate surface patches with vertices. As previously discussed, the graphics hardware can deal more efficiently with several large textures than with numerous small textures. As in the other systems, the coefficient textures are packed into a larger texture map.

4.5.1 Results

We have implemented this system on the hardware platform described in Chapter 3. A graph of timing results from several different models is shown in Figure 4.8, and the parameters used for these timings are shown in Table 4.9. The rendering algorithm is compact, fast, and efficient and can render all of the models in this paper at over 200 frames per second. An image generated with our system is shown in Figure 4.6, and a side-by-side comparison with a image that was withheld from the training data is shown in Figure 4.1.

The hierarchical construction method is much faster than the adaptive construction method due to the fact that the adaptive construction method can potentially cause the recomputation of a number of coefficients. For the 4K-vertex heart model, the adaptive construction generated about 5.1 Least Squares fitting computations per image, while the hierarchical construction only generated about 1.7. As this is the most time-consuming aspect of the process, reducing the number of Least Squares fits is important to achieve good performance. This performance gain enables higher resolution

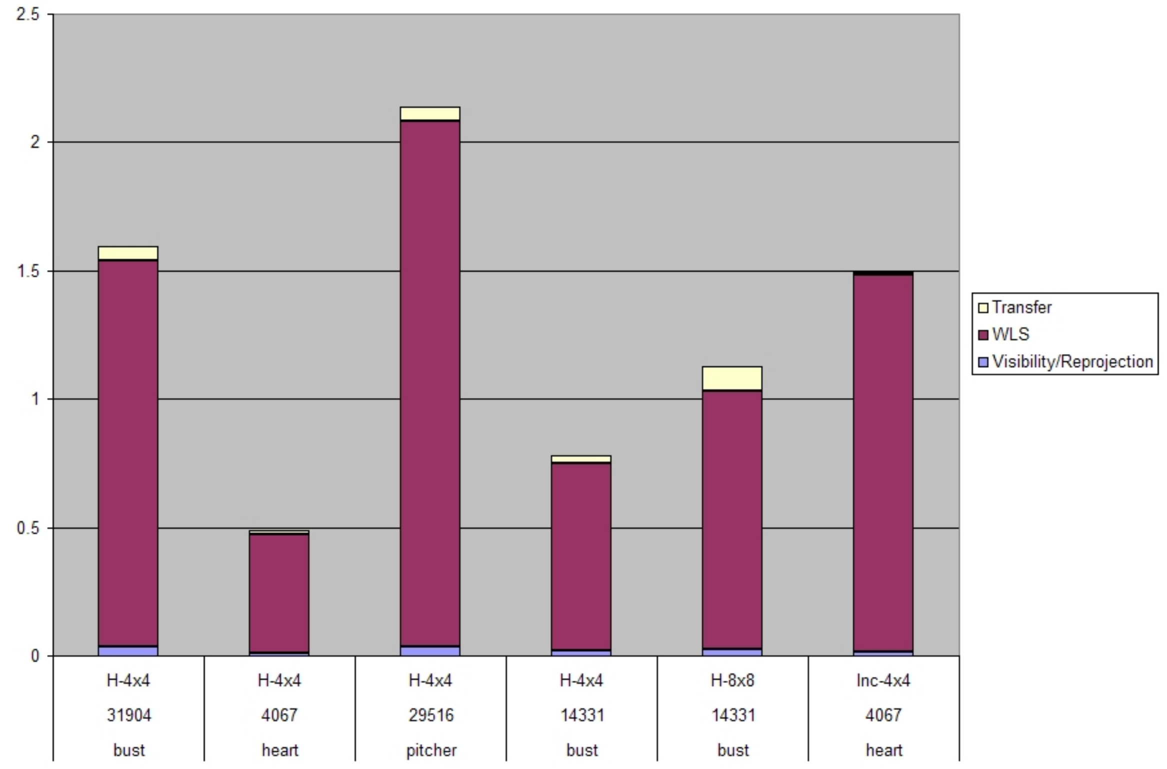


Figure 4.8: Timing results (in seconds per image) for the Incremental WLS construction. We measured three quantities; the time to compute the visibility and reproject the vertices into the image, the Least Squares fitting times, and the time to transfer the computed results to the graphics card for rendering. The dominant term is the Least Squares fitting.

Model	# Vertices	Construction	# Centers
Bust A	31K	Hierarchical	16
Heart A	4K	Hierarchical	16
Pitcher A	29K	Hierarchical	16
Bust B	14K	Hierarchical	16
Bust C	14K	Hierarchical	64
Heart B	4K	Adaptive	16

Figure 4.9: A description of the models and construction methods used for timing data.

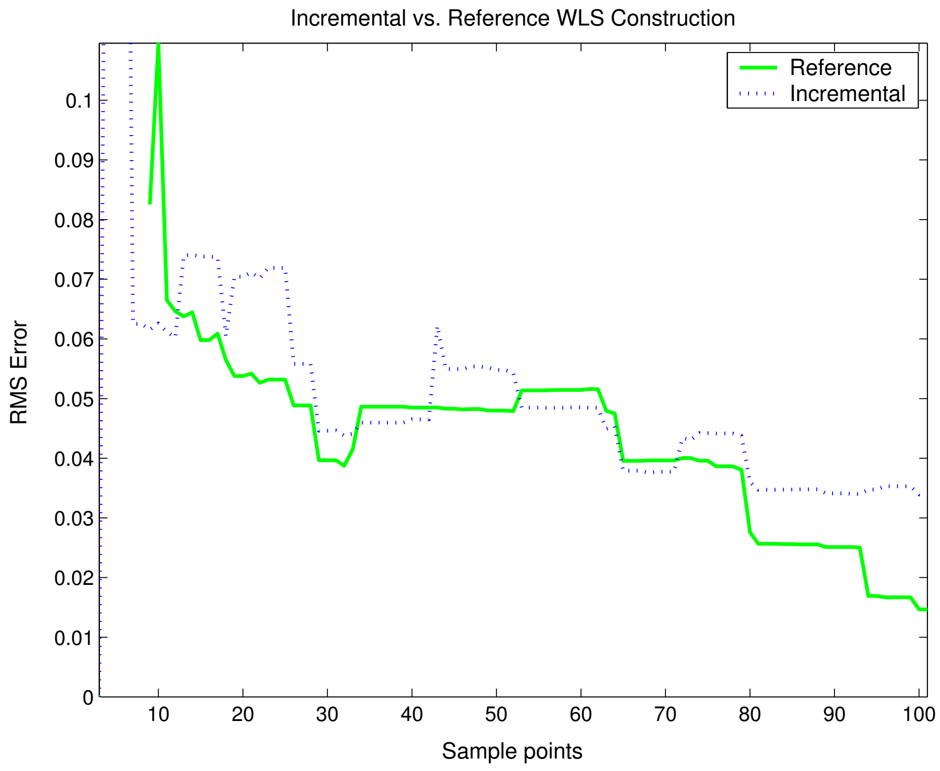


Figure 4.10: The reconstruction error of the hierarchical construction versus a batch construction for a single surface patch of the bust model. Each method used only the input samples available, and the error was measured against the full set of samples. The hierarchical algorithm is initially superior to the batch algorithm, and continues to be similar in error behavior while also being much faster to compute.

reconstruction; note that the bust model with 64 centers is only 1.4 times slower than the 16 center version, even though it has 4 times as many coefficients. However, the increased number of coefficients is reflected in the data transfer time, which is close to 4 times longer.

One potential issue with the hierarchical construction is that it could introduce error. To quantify this error, we conducted an experiment to compare the quality of the reconstruction with a reference batch process which has global knowledge. The results are shown in Figure 4.10.

We have tried several center placement strategies; a uniform grid over projected hemisphere directions, a uniform disk using Shirley's concentric mapping (Shirley and Chiu, 1997), and jittered versions of each. A comparison is shown in Table 4.1. For most of the models in this paper we use the grid method due to its ease of graphics hardware implementation.

Layout	Mean RMS Error	StD RMS Error
Uniform Grid	0.0519	0.0086
Jittered Grid	0.0608	0.0148
Uniform Disk	0.0497	0.0045
Jittered Disk	0.0498	0.0036

Table 4.1: RMS Error values from reconstructing a WLS approximation while varying the center layout. The error was computed with a training set of 64 images and an evaluation set of 74 images. The disk is a slight improvement in terms of error compared to the grid, and it has a large benefit in terms of reducing the variance of the error.

4.6 Conclusion

We have introduced Incremental WLS, a fast, efficient, and incremental algorithm for the representation and rendering of surface light fields. Incremental WLS can be used to render high quality images of surfaces with complex reflectance properties. The incremental construction is useful for visualizing the representation as it is being captured, which can guide the user to collect more images in undersampled regions of the model and minimize redundant capture of sufficiently sampled regions. The rendering algorithm, which is implemented on the GPU for real-time performance, provides immediate feedback to the user.

4.6.1 Incremental WLS and Online SVD

Before I describe the third component, I would compare the Online SVD and the Incremental Weighted Least Squares techniques. Both of these techniques incrementally construct a compressed representation of the scattered data. They differ not only in representation, but also in the way that they partition the SLF. The Incremental WLS partitions the SLF as a set of points which represent a 2D hemisphere. In contrast, the factorization approach of the Online SVD partitions the SLF as a set of surface patches which represent a 4D function of the surface and the hemisphere. The factorization approach shares the variance across surface and hemisphere, which enables it to adapt to the complexity of the hemisphere or surface.

The Incremental WLS does not share the reflectance between nearby surface points, and as a consequence does not achieve the same compression rate as the factorization approach. For our experiments, the size of a Incremental WLS SLF with comparable quality is about an order of magnitude larger. This ratio varies with SVD rank, number of centers, incremental construction method, and geometric complexity. Due to the increased size, the time to incorporate a new image in an Incremental WLS SLF is about 20%-40% slower than the Online SVD. This can be seen in Figures 3.14 and 4.8. However, the Incremental WLS has the capability to represent a much higher lighting frequency, since the WLS is a continuous representation. Also, the Incremental WLS does not have the same problems with missing data that the factorization approaches have. For both techniques, the rendering speed is realtime (30-60fps or more).

To increase the compression rate of the Incremental WLS, it would be possible to partition to SLF differently to allow nearby points to share data by collecting WLS coefficients from itself as well as its neighbors. This would involve adjusting the distance weighting to reflect the geometric distance along the surface of the model. We could also use an approach similar to Zickler (Zickler et al., 2005) to share reflectance values across a surface.

CHAPTER 5

Capturing a Surface Light Field Under Virtual Illumination

SLFs can be used to render the complex reflectance properties of a physical object. However, they can only represent the fixed lighting conditions of the environment where the model was captured. This is problematic for synthetic environments such as games or virtual environments in which a specific illumination environment is desired. These illumination environments could come from light probes captured in real locations, or from synthetic lighting environments created by artists. Some examples of lighting environments are shown in Figure 5.1.

The lighting environments must be physically duplicated in the lab at the time of capture which can be difficult. One approach is to use lights or projectors that are physically situated to mimic the virtual lighting positions and colors. This approach can only represent low-resolution lighting, and is often time-consuming and inaccurate. Another approach is to collect the full 6D BTF, which enables the object to be rendered under arbitrary lighting environments. However, this requires a significant increase in the amount of data that is acquired, most of which is unnecessary since the lighting

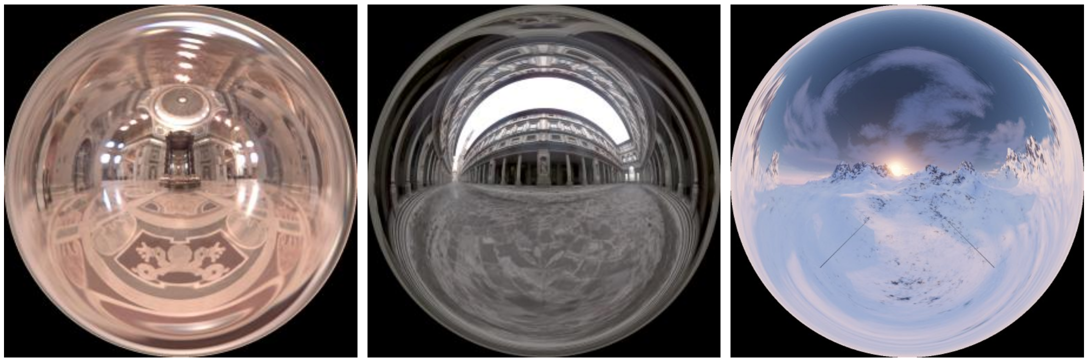


Figure 5.1: Sample light probes. Left and middle: Real light probes captured from St. Peter's Cathedral and Uffizi Gallery, courtesy of Paul Debevec (from *debevec.org*). Right: Synthetic light probe created by the artist Crinity.

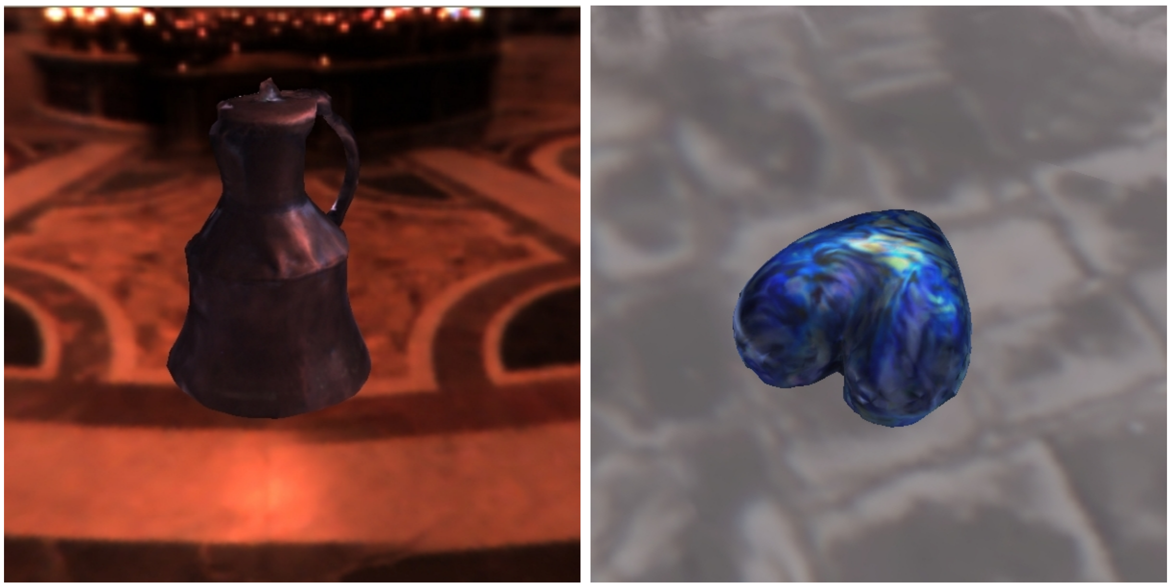


Figure 5.2: Left: pitcher model in St. Peter's light probe. Right: heart model in Uffizi light probe.

condition is already known.

In this chapter I describe a third approach: a method for capturing a surface light field using the virtual illumination from an environment map. We use a simple setup consisting of a projector, a camera, a pan-tilt unit, and tracking fiducials to recreate the desired lighting environment. To decrease noise and improve the quality of the capture under low- and high-dynamic range environment maps, we use an extended version of the multiplexed illumination algorithm (Schechner et al., 2003). This results in a high-dynamic range SLF which accurately represents the interaction of the virtual illumination with the real object. Two examples are shown in 5.2.

5.1 Introduction

In this chapter I describe a method to capture a SLF under virtual lighting, requiring only very modest equipment and infrastructure. To mimic the light falling onto the object from the environment, light is projected onto a large screen and reflected onto the object. This setup simulates a large controllable light source which illuminates the object. If the projector were to be pointed directly at the object, it would only simulate a small portion of the lighting environment from the perspective of the object (i.e. those rays coming directly from the projector).

In order to recreate a virtual lighting environment, the light must illuminate the object from all directions. This would require moving the screen and the camera around the object as we acquire images. However, instead of moving the camera as in previous chapters, we move the object itself using a programmable pan/tilt unit. Moving the object simplifies the setup since the screen and projector can be fixed in place. It also avoids some of the problems with the camera (and user) occluding the light. The illumination that is projected onto the screen is a portion of the lighting environment, and changes as the object rotates and tilts. In this way, the projected light changes to maintain the fixed relation between the object and the lighting environment. A diagram of our system is shown in Figure 5.3, and a photograph of the laboratory setup in shown in Figure 5.4.

However, a single projector and screen do not represent the entire hemisphere of incident light, so multiple camera positions are used to simulate the incident light arriving from all directions. The lighting environment can be arbitrarily complex, and can be either fully synthetic or captured from a real scene as a light probe (Debevec, 1998). Examples of light probes that we used in our experiments are shown in Figure 5.1. Since these light probes often have dramatic contrast between the darkest and brightest areas, we employ high-dynamic range lighting and imaging.

One of the major problems that must be addressed to make such a system work is the mapping between the physical setup and the virtual lighting environment. This is established by determining the correspondence between the position of projected pixels and the position of the object relative to the camera. We describe two methods for establishing this correspondence; a simple geometric calibration method for simple planar screens, and a more complicated method for screens with arbitrary geometry. The geometric calibration is only part of the total calibration; in order to correctly simulate the color of the lighting environment, it is also important to calibrate the color of both the camera and the projector.

To deal with the reduced illumination from reflecting the light off of the screen, we using multiplexed illumination (Schechner et al., 2003). This technique was developed in the computer vision community to reduce the noise in acquired images. Since the lighting environments are high-dynamic range images, I present a method to extend multiplexed illumination to high-dynamic range images.

In the next section I discuss how the different environments are registered both geometrically and photometrically. Section 5.3 describes how we use multiple cameras to capture the entire set of incident light. Section 5.4 describes multiplexed illumination and the extension to high-dynamic range. This is followed by results for objects under low and high dynamic range illumination.

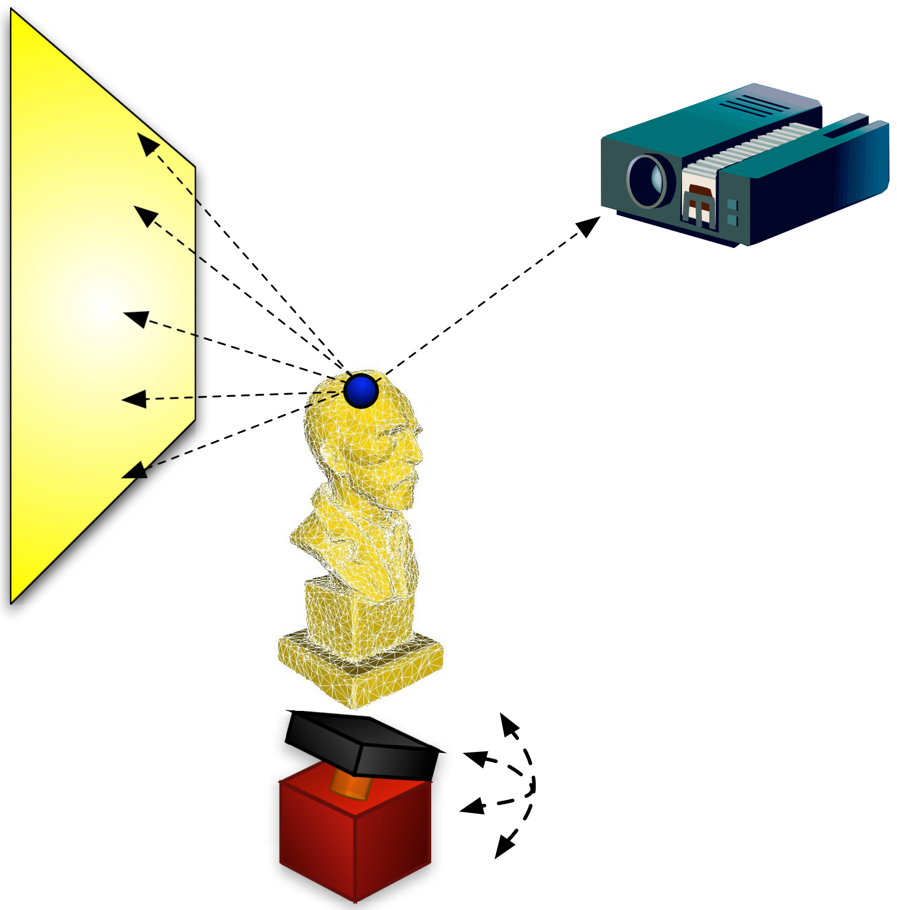


Figure 5.3: A diagram of the capture system. Light is projected onto a large screen, which is then reflected onto the object. The object is mounted on a pan/tilt device.

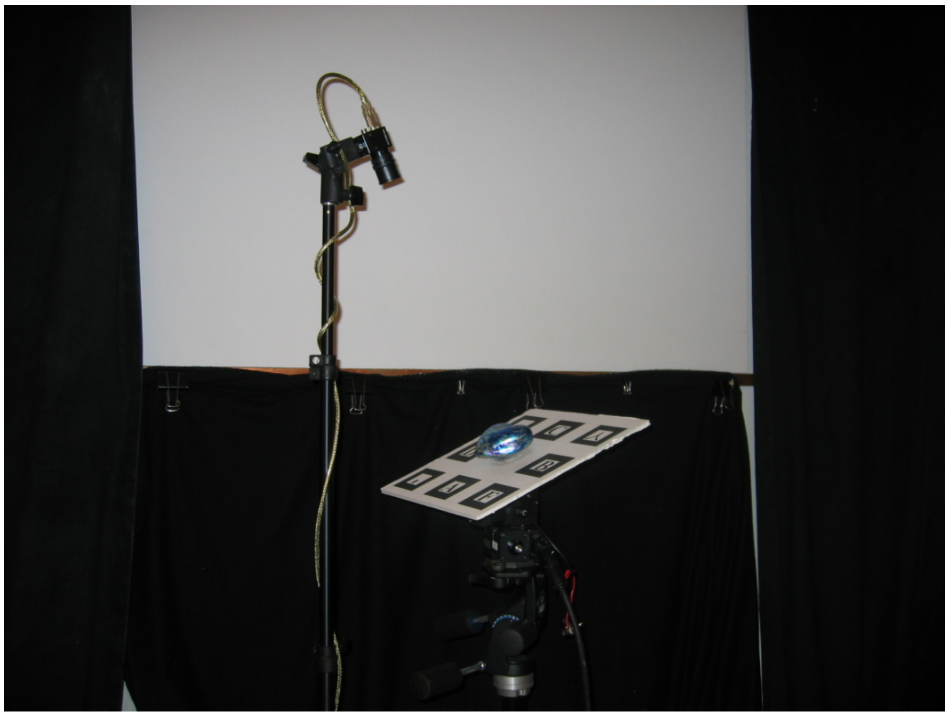


Figure 5.4: A picture of the Surface Light Field capture system. The light is projected onto the screen and reflects down onto the object. The object is mounted on a tracking board, which is mounted on a pan/tilt device. The fiducial markers are used to estimate the object's position and orientation relative to the screen. The black curtains on the walls minimize light scattering in the lab.

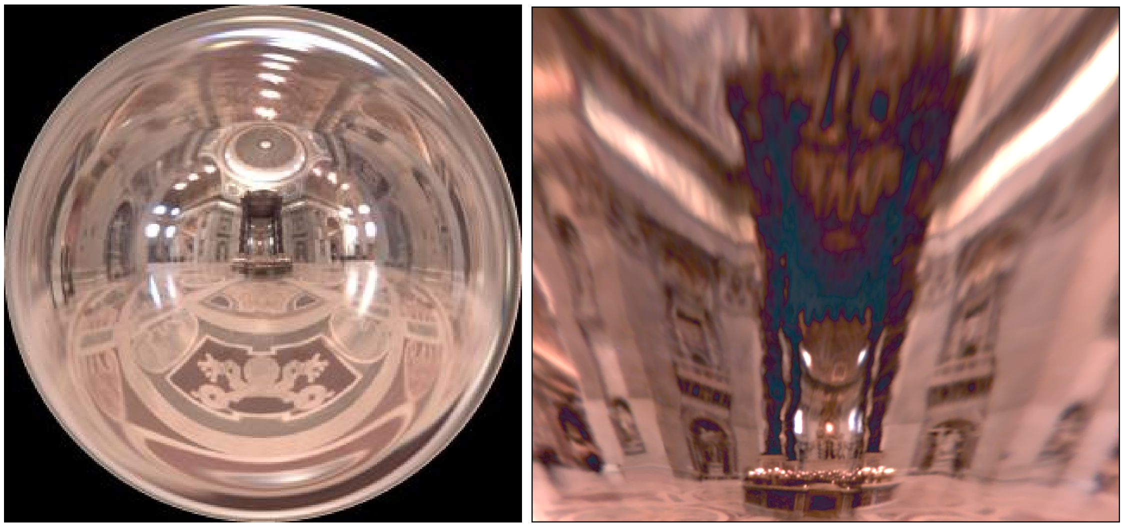


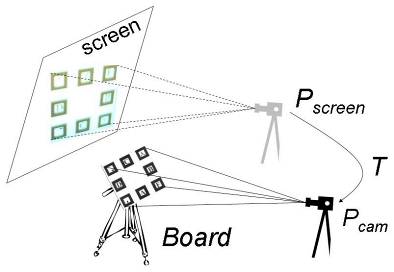
Figure 5.5: Mapping the virtual illumination environment to our physical setup. Left: The virtual lighting environment. Right: The image that is projected onto the screen. Note that the image is stretched upward and outward to represent the rays of light from the virtual illumination environment.

5.2 System Calibration

One of the important components of our system is the calibration, since it registers the virtual lighting environment to the real physical object. This calibration involves four components: the camera, the tracking fiducials, the physical object, and the projected image on the screen. The camera is fixed in relation to the screen, and the fiducials are fixed in relation to the object. Thus by calibrating the geometric relationship between the screen and the physical object, we can accurately characterize the entire system. In this section I describe how this relationship is determined in our system.

We have developed two techniques for geometrically calibrating the setup; a single image calibration, and a reflective object calibration. The single image calibration is simple and fast, but it can only handle screens which are planar parallelograms. The reflective object calibration is more robust and relaxes many of these constraints, but requires more images. Both of these systems leverage the fiducial tracking system ARToolkit (Kato and Billinghurst, 1999) that was used in the previous chapters.

The result of this calibration is a mapping between pixels on the screen and points on the surface of the object. This mapping is used to determine how to display the lighting environment on the screen, as shown in Figure 5.5.



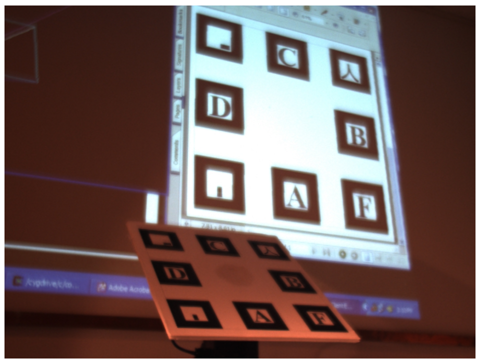


Figure 5.6: Calibration method for planar screens. Left: A diagram of the physical setup for geometric calibration of the screen. Right: The calibration image, which shows both the projected fiducial board and the physical fiducial board. This is used to establish a correspondence between the two coordinate systems.

5.2.1 Single Image Calibration

This calibration method uses ARToolkit to compute the correspondence between the screen and the object. We display fiducials of known projected size onto the screen. The camera is placed in such a way that it can see the calibration pattern on the screen simultaneously with the fiducials on the base plate. This is shown in Figure 5.6. We estimate the camera position $P_{screen} = [R^T_{screen} | -R^T_{screen} C_{screen}]$ in the screen coordinate system. We also compute the position of the camera $P_{cam} = [R^T_{cam} | - R^T_{cam} C_{cam}]$ with respect to the board coordinate system.

This results in two estimated camera positions: one in the screen coordinate system and one in the tracking board coordinate system. Since both camera positions were computed by employing the same image, we can compute a transformation T

$$T = \begin{bmatrix} R^T_{sb} & -R^T_{sb} C_{sb} \\ 0_{1\times 3} & 1 \end{bmatrix}, \tag{5.1}$$

which transforms the camera P_{screen} from the screen coordinate system into the board coordinate system. This results in a rotation R_{sb} and translation C_{sb} of the screen in the board coordinate system (see Figure 5.4).

Since the projected fiducial board does not represent the entire screen, we need to determine the exact extent of the screen in the board coordinate system. Given the size of the displayed fiducials and their size in pixels on the screen we can compute the homography H_{ps} that maps the image onto physical screen coordinates. Note that this only works for screens which are planar parallelograms. This homography can be applied afterwards to the image corners, which describes the screen coordinates in the board coordinate system. These corners of the screen are used to compute the direction vectors into the light probe.

These calibration transformations are computed once during initialization and stored. During runtime the object is rotated, and the tracking fiducials are used to update the camera position and the screen position by applying the inverse change to the camera and screen. Hence we are always able to determine the screen position to display the appropriate solid angle of the light probe.

5.2.2 Reflective Object Calibration

The previous method works well for screens which are planar parallelograms. However, it can be useful to consider screens which are not planar. For example, consider projecting images into the corner of a room. Since the screen covers a larger solid angle above the object, we can reduce the number of cameras required to capture the full hemisphere of incident light. These considerations led us to look for a more general calibration method.

This calibration procedure uses a reflective calibration object and can deal with arbitrary screen geometries. The goal of the procedure is to determine the light rays corresponding to each projector pixel. These rays are used to index into the environment map to provide a "window" into the synthetic environment. This relationship is influenced by the angle between object and screen, geometry of the screen, and nonlinearities of the projection. This method also addresses several concerns over the previous method from Section 5.2.1:

- The tracking frame of the object may not be orthogonal to the screen.
- The screen may not be planar, and may have multiple surfaces (such as corners).
- The projector may not be orthogonal to the screen, and may have keystone effects.

These factors imply that physically measuring the system is difficult and often inaccurate. We propose a calibration procedure that takes advantage of a particular property of light probes; they are independent of translation. Thus the calibration procedure needs to only compute the rays emanating from the object, and does not need to compute the translation information. This simplifies the calibration procedure to one of determining the relation between a pixel on the screen and a ray in object space.

Pixel	-2%	-1%	+1%	+2%
(34, 131)	0.4064°	0.2011°	0.1971°	0.3902°
(151, 63)	2.0260°	0.9988°	0.9719°	1.9179°
(-18, 156)	0.0919°	0.0455°	0.0446°	0.0883°
(-72, 154)	0.7546°	0.3733°	0.3655°	0.7235°

Table 5.1: Angular error vs. relative radius error

We place a mirrored sphere with known radius and position in approximately the same location as the object. This mirrored sphere reflects the illuminated points on the screen back to the camera. Using the known projector pixel and the reflection of this point into the camera, the ray associated with each projector pixel can be computed. This is shown in Figure 5.7.

The process works as follows. A pixel is illuminated on the screen, and an image is captured of the light reflected from the sphere. The position of the pixel in the image is segmented using an adaptive threshold. The ray through the camera plane into the scene can be computed from the pose of the tracking board. This ray is then traced into the scene and intersected with the sphere. At the intersection point, the normal is computed and a reflected ray is generated. This reflected ray is the ray in 3D space which corresponds to the projector pixel.

This establishes a correspondence between projector pixels and rays in the scene. For rendering, these rays are rotated according to the delta rotation of the tracking board, and used to index into the environment map. The resulting images for the projector are shown in Figure 5.7.

5.2.2.1 Error

The primary source of error in this calibration procedure is from the physical measurement of the sphere. We measured the error in the reflected rays as a function of the error in radius of the sphere for several projector pixels. These results are shown in Table 5.1. Pixels near the edge of the sphere are more susceptible to error, since the normals change more quickly. In general, there is about 1° of angular error for 1% radial error. Other sources of error which we did not measure are camera pose estimation and segmentation problems.

Currently, we use a 9x9 sampling of the screen (100 calibration points) aligned on a regular grid. For screens which are mostly planar (including our corner screen), this is sufficient. However, for more complex screens a denser sampling would be needed.

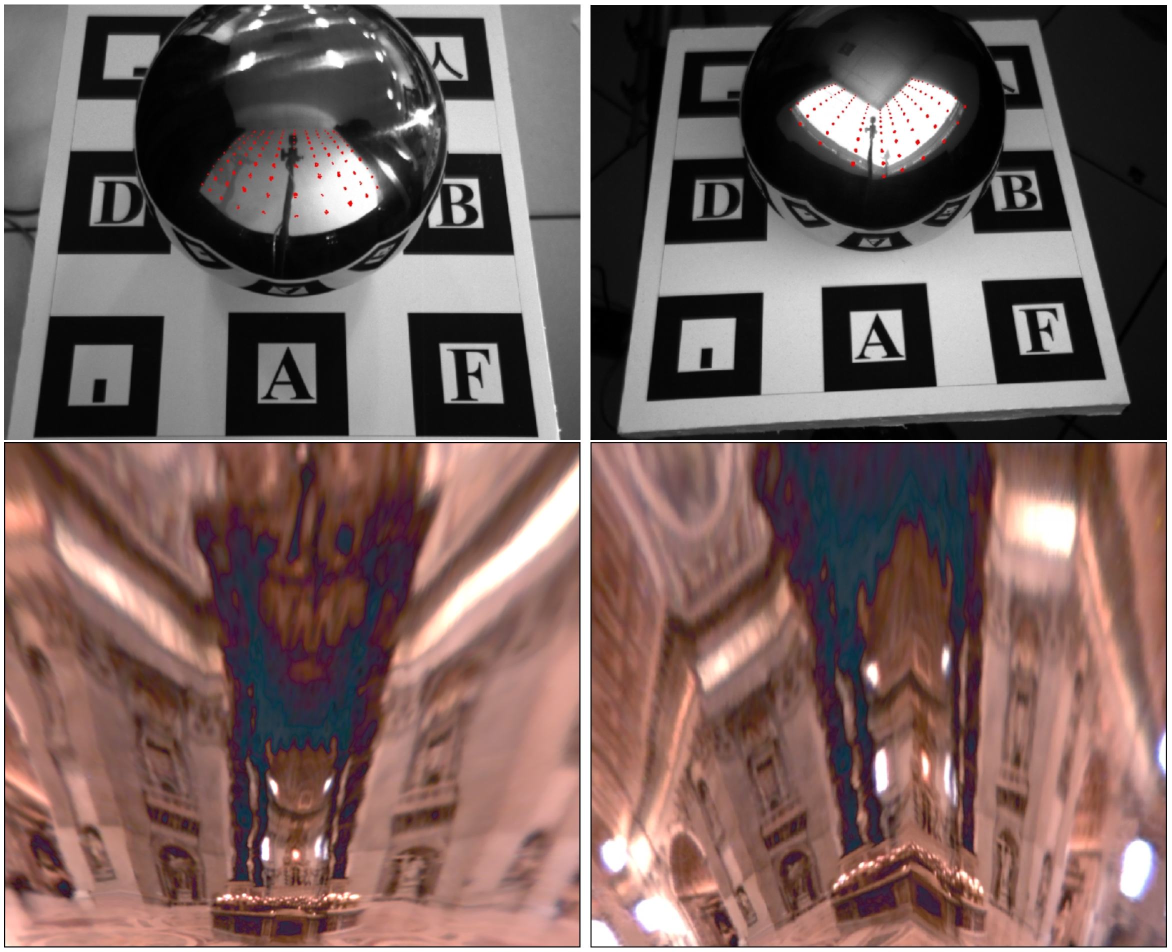


Figure 5.7: The reflective object calibration process, shown for a planar screen on the left and a corner screen on the right. Top: A reflective calibration object is placed in the same location as the object. Pixels are projected onto the screen and segmented from the images. Bottom: The reflected rays are used to index into the environment map, producing an image which is correct from the point-of-view of the object.

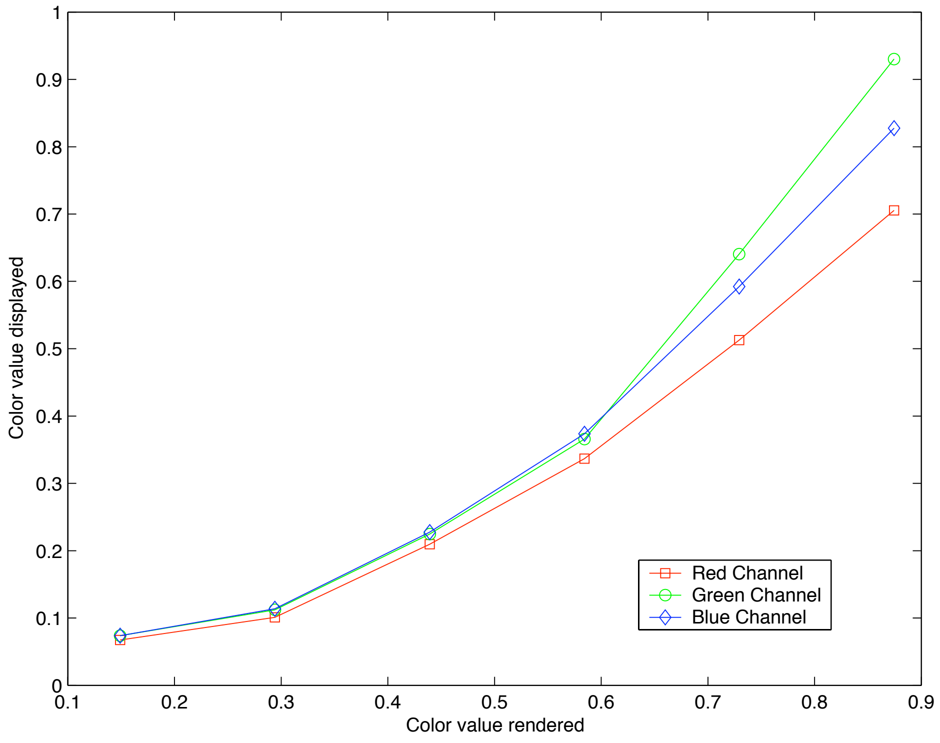


Figure 5.8: The relationship between the RGB color values that are sent to the projector and the color values that are actually displayed on the projector. We fit a polynomial to these curves and inverted it to correct for the error.

5.2.3 Projector Color Calibration

One of the sources of error in our setup is the color from the projector. It is not correct to assume that the light coming from the projector is linearly correlated with the values that are sent, even with the gamma set to 1.0 and the controls adjusted. This is important because both the HDR exposures and the Multiplexed Illumination assume that the light can be linearly combined. Without correcting for this non-linearity, the color of the resulting images would be incorrect.

We attacked this problem by applying a color correction to the image sent to the projector. To do this, we first need to determine the color response of the projector. We calibrated the color response of the camera using the Camera Color Calibration Toolkit (Ilie and Welch, 2005). We established the reference colors by placing the Macbeth color checker in the direct light of the projector. Once the camera was calibrated for this light level, we then displayed a series of pure and mixed colors and captured images. By averaging over a large area of these images, we obtained the measurements shown in Figure 5.8.

Once the projector values were measured, we fit a curve of the form $c = \alpha_1 x^2 + \alpha_2 x + \alpha_3$ to each RGB channel separately. A quadratic curve was chosen because it is simple to fit and can be inverted in graphics hardware by selecting the positive root of the quadratic equation. We first attempted to fit an exponential curve to the data, but were unable to obtain good results due to a knee in the data at the lower values.

An interesting area of research might be applying the methods of the Camera Color Calibration Toolkit to the projector. Using a camera that had been calibrated, the projector could display a Macbeth color checker, and use the same non-linear optimization over the set of available projector parameters.

5.3 Multiple Cameras

In order to properly capture the appearance of the object under the synthetic lighting conditions, the projector-camera system must be able to display the entire hemisphere of incoming light. However, due to physical limitations a single camera can only capture a section of the hemisphere, as shown in Figure 5.9. In this figure the tessellated points from the calibration procedure are rendered in relation to the object. Any object captured under this physical setup would only be lit by a portion of the environment.

5.3.1 Coverage

To capture the full environment, we need to add more cameras, which are situated so that they can capture a broader portion of the hemisphere. In Figure 5.9 four cameras have been placed around the object to capture more light. In essence, each camera is capturing the light which has bounced off the object in a different direction.

The diagrams in Figure 5.9 were using the vectors from the calibration procedure. The calibration procedure generates a set of vectors in the tracker's coordinate system. These vectors were rotated into each camera's coordinate system, then rotated back into the common board coordinate system using the initial camera Cam_0.

$$v' = Cam_0^T * Cam_i * v$$

5.3.2 Integrating Multiple Cameras

Once a set of images have been captured for each camera position, they need to be combined to generate a single surface light field. As long as the "windows" do not

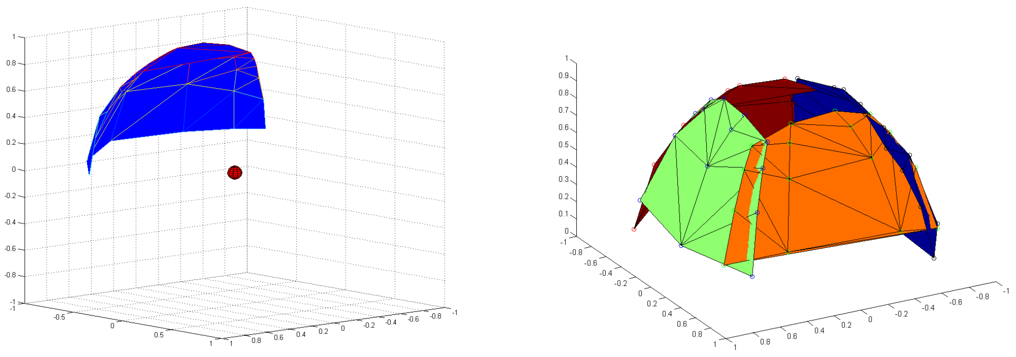


Figure 5.9: How much of the environment does the screen cover? Left: A single camera position covers only a portion of the environment, resulting in an incorrect SLF. Right: Four cameras cover the entire hemisphere of light above an object, leaving only a small hole in the top.

overlap, the light is independent and can be simply summed to get the final result. However, since each of the cameras are located in different positions, it is unlikely that their projected positions will line up. If that were the case, then we could simply add the images together to get a composite image.

Instead, we combine the images in the resampling stage, which is later in the pipeline. As described in Chapter 3, after the visibility has been computed the camera positions are projected onto the coordinate system of each vertex and the points are resampled using a Delaunay triangulation. The values are resampled onto a fixed grid, which is the same for every camera. At this stage we can add the values from the different cameras together without incurring any more projection error than the resampling already creates.

For these experiments, we used only one camera and captured the images sequentially. It would obviously be much faster to use a set of cameras and capture in parallel, although it would raise the cost of the system.

5.3.3 Overlap

The data from cameras which capture different portions of incoming light can be summed to get a surface light field which incorporates the entire hemisphere of incoming light. However, if these portions overlap then some of the light will be counted twice. This overlap can be seen in Figure 5.9. There are several methods to avoid overlap. One method is to manually position the cameras so that none of the windows overlap. This can be quite difficult to do, particularly for large numbers of cameras.

A better method is to compute the section of the screen which is occluded by previous cameras, and not display in that area. This technique only works if the cameras are captured sequentially, which is the case in our setup. For each camera position, we need to compute the portion of the screen which is occluded by previous camera positions. We do this on a per-pixel basis by treating each pixel on the screen as a ray and testing against the rays from the calibration procedure.

In general, this operation would require testing whether a ray intersects a spherical polygon, which is a difficult task. We can make several simplifying assumptions to make this more efficient. The first simplification is to notice that the edges of the screen in the world are straight lines, even though they are curved when normalized. This means that we can define a plane which contains the edge of the screen and the center of the calibration sphere. For each screen pixel, we test whether the ray is within the frustum of a previous camera. This is a simple dot-product with each of the planes that define the frustum, which can be accomplished on the graphics hardware. If the pixel is inside a previous frustum, we do not display a color for this pixel to avoid counting regions twice.

The other simplifying assumptions have to do with the relative size of the objects. As previously mentioned, we assume that the calibration object is approximately the same size as the object we are capturing. In this way the rays which emanate from the calibration object are approximately the same as the rays which would emanate from the object. The new assumption we are adding is that the calibration object is very small relative to the screen, so that it can be treated as a point. For our experiments, this is true; the area of the screen is about 12,000cm^2 and the area of the sphere in the plane of the screen is about 70cm^2, over two orders of magnitude smaller.

5.4 Multiplexed Illumination and High-Dynamic Range

Since the range of illumination present in the world is much larger than can be reproduced by displays or captured by cameras, we need to use High-Dynamic Range (HDR) imaging techniques (Larson et al., 1997). Research in this area was pioneered by Debevec (Debevec, 1998) in a paper that described how to linearize the response of cameras and combine multiple exposures into a single HDR image. Debevec also describes a technique for illuminating synthetic objects under HDR illumination (Debevec, 1998). A recent book serves as an excellent reference to the body of work surrounding HDR imaging (Reinhard et al., 2005).

To accurately model an object under a virtual illumination environment, we need to be able to handle the high-dynamic range of light. This requires mapping the high-dynamic range values from the light probe onto the low dynamic range of the projector. A common technique is to split the high-dynamic range light probe into multiple exposure levels, which can then be re-combined to form the full dynamic range. We use the technique by Cohen et al. (Cohen et al., 2001) and split the energy into discrete levels, each of which fits within the 8-bit range of the projector. We use the function

$$Color_E = \frac{1}{10} \left\lfloor \frac{Color}{10^E} \right\rfloor \tag{5.2}$$

with the exposure level $E \in [-2...3]$. Any color values outside the range $[0...1]$ are ignored. This mapping enables us to reconstruct the full dynamic range of the light probe as a sum of scalar multiples of the exposure levels.

Once this mapping is established, we could capture a set of images using the different exposure levels, then scale the images to get the high-dynamic range result. The problem with this direct approach is that most of the light energy is concentrated at a few pixels on the screen. Hence very little light is falling on our object for shorter exposure times. The problem is that these short exposure images are the images that must be scaled by a large number to convert them to high dynamic range images. Unfortunately, this also scales the CCD noise by the same amount. To minimize this noise we extend the multiplexed illumination (Schechner et al., 2003) to high-dynamic range images. In this section I describe multiplexed illumination and our extension to high-dynamic range lighting.

5.4.1 Multiplexed Illumination

Capturing images under dim lighting is difficult due to the presence of camera CCD noise. This noise can significantly degrade the quality of the image due to the low signal-to-noise ratio. Schechner et al. (Schechner et al., 2003) introduced a technique to significantly reduce the noise in the captured images with multiple low intensity light sources. The main idea of multiplexed illumination is to reduce the additive independent noise of the camera's CCD array in low-light situations by collecting multiple measurements per image. Using n light sources, we can increase the signal-to-noise ratio by up to $\frac{\sqrt{n}}{2}$ with the same number of images.

Before the general case is described, we first consider an example of three light sources. An acquired image lit by light source l will be called $a(x, y)$, and the image irradiance is $i(x, y)$. Now suppose that we acquire a set of 3 images of the same scene, each lit by only one of the light sources. We can write this in matrix form as:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix}$$

We have used a 1 where the light is on, and 0 where the light is off. In this setup, it is trivial to recover the images as lit by one of the light sources, as the images were acquired under a single light source. However, each of these images are lit by only 1/3 of the available light sources. This is a problem in low-light situations, where the noise from the CCD cameras corrupts the images. For many of the light probes that we use, the signal-noise ratio of the captured images is poor. We can improve this ratio by increasing the signal and lowering the noise. Increasing the signal requires incorporating more light sources into each measurement, while decreasing the noise requires incorporating more measurements into each image. Multiplexed illumination is a technique to accomplish both at the same time. If each of the images is captured under 2 of the 3 lights, we can write the system:

$$\begin{bmatrix} a_{1,2} \\ a_{2,3} \\ a_{1,3} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix}$$

Each of the observed images $a(x, y)$ consist of the light from the multiple lights multiplexed together. We want to *demultiplex* these images by reconstructing the images $i(x, y)$ which contain the light from only a single light source. Suppose the we want to reconstruct i_1, the scene lit by light 1. If we add the observed image $a_{1,2}$ to $a_{1,3}$, we get twice i_1 as well as i_2 and i_3. Since $a_{2,3}$ consists of i_2 and i_3, we can subtract this two get twice i_1. Dividing by two gives us the contribution of i_1, as reconstructed from the multiplexed images. Writing this down for each of the 3 lights gives us:

$$\begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_{1,2} \\ a_{2,3} \\ a_{1,3} \end{bmatrix}$$

The advantage of this new method is the reduced noise. To see this, assume that each of the images have independent additive noise σ. The noise of the demultiplexed image will be a combination of the noise from the three component images. To get the total noise, we add the variances of the individual noisy images together (the square of the standard deviation):

$$(\pm\frac{1}{2}\sigma_1)^2 + (\pm\frac{1}{2}\sigma_1)^2 + (\pm\frac{1}{2}\sigma_1)^2 = \frac{3}{4}\sigma^2$$

The noise has been reduced to $\sqrt{\frac{3}{4}}\sigma$, an improvement of about 14% without increasing the number of images that we have to acquire.

5.4.2 General Multiplexed Illumination

Now let's consider the general case of n lights and n images. The light are additive quantities and are linearly related by superposition

$$\underbrace{[a_{\zeta_0}(x,y),\ldots,a_{\zeta_n}(x,y)]^T}_{a(x,y)} = W\,\underbrace{[i_0(x,y),\ldots,i_m(x,y)]^T}_{i(x,y)}$$

where $a_{\zeta_k}(x,y)$ is the light observed at pixel (x,y) under the set of lights ζ_k and $i_l(x,y)$ is the energy contributed by light source l at pixel (x,y). The multiplexing matrix W for the light sources $l = 1,\ldots,m$ describes which light sources illuminate the scene. An element $W_{i,j}$ is one if the light j is illuminated in image i, and zero if the light was "off". The sets ζ_k consists of all of the lights in row k which are "on".

In order to recover the images $i_l(x,y)$ as lit under a single light source l, we demultiplex the observed images $a(x,y)$ by inverting the matrix W:

$$i(x,y) = W^{-1}a(x,y)$$

The next question is to determine an effective multiplexing matrix. There are several properties that an optimal multiplexing matrix should have: it should minimize the MSE, and it should be easy to invert. The authors (Schechner et al., 2003) suggest using a special matrix form called *Hadamard matrices* which were developed in the X-ray astronomy community (Harwit and Sloane, 1979). Hadamard matrices have 0's or 1's as elements of the matrix, and have the property that $\frac{n+1}{2}$ of the elements are 1 along any row (an algorithm for constructing these matrices is given in (Harwit and Sloane, 1979)). This means that for each image, a little over half of the lights are on.

From this, we can calculate the benefit of using such Hadamard matrices for multiplexing the illumination. Let the light level from one light be L with a standard deviation of σ due to additive independent noise. Thus the SNR of an image under a single light source is:

$$SNR_{single} = \frac{L}{\sigma}$$

Using multiplexing with Hadamard matrices, the number of lights that are used is $\frac{n+1}{2}$ so the signal is increased to $L\frac{n+1}{2}$. The noise level is a combination of the independent additive noise from n images, which means that the variance of the noise is $\sigma^2 n$ and the standard deviation is $\sigma\sqrt{n}$. The SNR of the multiplexed images is:

$$SNR_{multi} = \frac{L(n+1)}{2\sigma\sqrt{n}}$$

This means that the ratio of the multiplexed SNR to the single image SNR is:

$$\frac{SNR_{multi}}{SNR_{single}} = \frac{\sqrt{n} + \frac{1}{\sqrt{n}}}{2} \approx \frac{\sqrt{n}}{2}$$

The larger the number of images, the more useful this technique becomes. For 5 images, the increase is SNR is only 1.34x, but for 50 images, the increase is 3.61x. It's important to note that multiplexed illumination does not require taking any more images than single light illumination. The only added computation is the post-process demultiplexing step, which is minimal when using Hadamard matrices.

5.4.3 Multiplexed illumination for HDR images

In this section I discuss how to apply multiplexed illumination to HDR images. The approach from (Schechner et al., 2003) can only be applied to one exposure level at a time since the relationship across exposure levels is non-linear (Equation 5.2).

Our approach tiles the projector screen into multiple regions $i_0, \ldots, i_m$ where each region displays a different exposure level (see Figure 5.10). As previously mentioned, much of the high-intensity lighting is concentrated in a small number of pixels. These short-exposure regions are dark in comparison to the long-exposure regions. By splitting the screen into multiple regions, we can balance the amount of light that falls on the object. This allows us to use a single exposure setting across all of the images, which simplifies the acquisition process.

Each region on the screen is a light source l displaying a part of the light probe with exposure level E_r. Demultiplexing these images requires that we account for light sources that have different exposure levels. However, this means that the coefficients

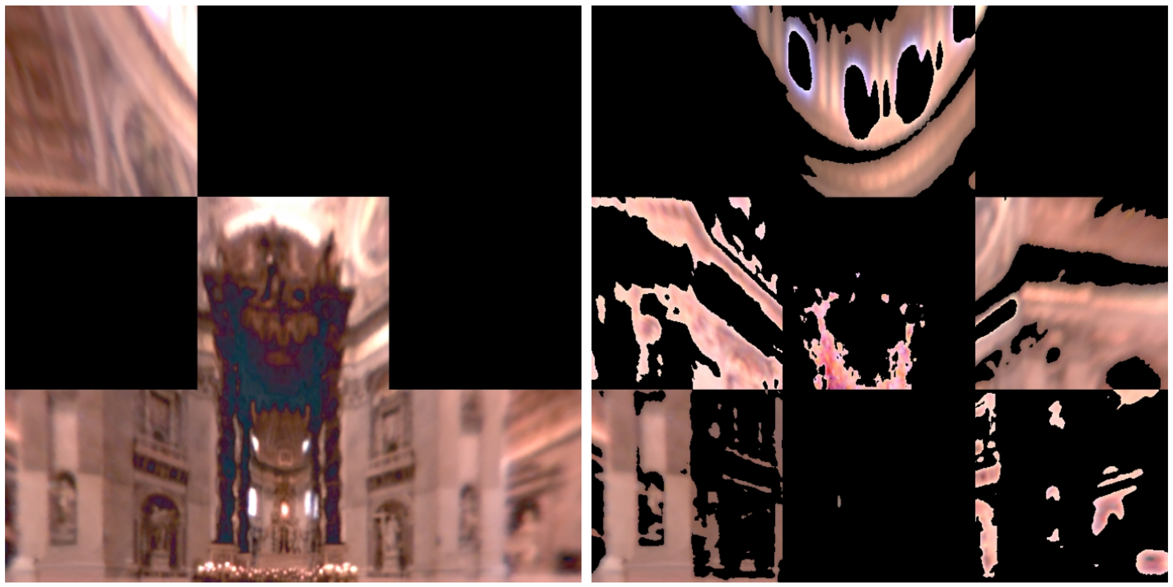


Figure 5.10: Multiplexing the St. Peter's Cathedral light probe (light probe courtesy of debevec.org). Left: Low-dynamic range multiplexing. The screen is divided into 9 regions, where each region is either "on" or "off". Approximately half of the regions are "on" at a time. Right: High-dynamic range multiplexing. Each region represents a different exposure level.

$\tilde{W}_{k,l}$ of the weight matrix $\tilde{W}$ for multiplexed high-dynamic range illumination are no longer binary. They now model the exposure level of the region $\tilde{W}_{k,l} = 10^{-E_t}$ with $t = 0, \ldots, h-1$ enumerating the different exposures E_t

$$a(x,y) = \tilde{W} \underbrace{\left[i_{0,E_0}(x,y), \ldots, i_{m,E_0}(x,y), \ldots, i_{m,E_{h-1}}(x,y)\right]^T}_{i_h(x,y)}, \tag{5.3}$$

where $i_{l,E_t}(x,y)$ with $t = 0, \ldots, h-1$ is pixel (x,y) in the image of light source l with exposure level E_t and $a_k(x,y)$ is the k-th image. Hence the light of each light source at all exposure levels E_t at pixel (x,y) can be computed by

$$i_h(x,y) = \tilde{W}^{-1} a(x,y). \tag{5.4}$$

Equation (5.4) is an extension to handle multiple dynamic ranges of the light sources. A sample image used for high dynamic range illumination is shown in Figure 5.10.

To compute the contribution of each light source from the captured images, the weight matrix $\tilde{W}$ has to be inverted. The efficient inversion from (Schechner et al., 2003) can not be directly applied to the weight matrix $\tilde{W}$ for the HDR illumination since these values are no longer binary. Additionally the high range of entries $\tilde{W}_{k.l}$ of the exposure levels for high dynamic range images leads to a high condition number of $\tilde{W}$. This poses numerical problems for a direct inversion of $\tilde{W}$.

To invert this matrix, we note that the illuminations of the light sources within each exposure level are additive quantities, but they are not additive across exposure levels. Hence the matrix $\tilde{W}$ consists of blocks corresponding to the different exposure levels. Each of the blocks can be written as

$$\tilde{W}_{tm:(t+1)m-1,th:(t+1)h-1} = 10^{-E_t} W, \tag{5.5}$$

where $\tilde{W}_{tm:(t+1)m-1,th:(t+1)h-1}$ is the sub matrix of $\tilde{W}$ from row tm to $(t+1)m-1$ and columns th to $(t+1)h-1$. According to (5.5) the inversion scheme from (Schechner et al., 2003) can be applied to each of the sub matrices $\tilde{W}_{tm:(t+1)m-1,th:(t+1)h-1}$

$$\tilde{W}^{-1}_{tm:(t+1)m-1,th:(t+1)h-1} = 10^{-E_t} W^{-1}. \tag{5.6}$$

This re-ordering of the matrix results in a new weight matrix $\tilde{W}$ for high dynamic range illumination that can be inverted as efficiently as before. To simplify the demultiplexing computation, the same sequence of exposure levels is displayed for each position. This means that the weight matrix $\tilde{W}$ only has to be inverted once. Furthermore the computation can be done online after the capture of image a_k by multiplying the k-th column of $\tilde{W}^{-1}$ with a_k. This means that we can stream the images a_k through memory and never have to store them.

5.5 Implementation and Results

We use a 1024x768 resolution projector pointed at a 4x3 foot white screen as light source. The printed fiducial board is placed atop a pan-tilt unit from Directed Perception$^{\text{TM}}$. Our surface light field capture rig is shown in Figure 5.4. The rest of the hardware setup, including camera and computers, is the same as discussed in the previous chapters.

One of the advantages of our system is cost; we estimate that a total of \$3000 is enough to buy the required equipment, although in fact most labs already have this gear. Furthermore, our camera-based calibration does not require a rig for precise positioning. We used off-the-shelf tripods to hold the camera and the pan-tilt motor.

The system was built using the software components described in Chapter 3, such as camera tracking, visibility, resampling. Since these components were developed for standard 8-bit images, the extension to high-dynamic range floating-point images

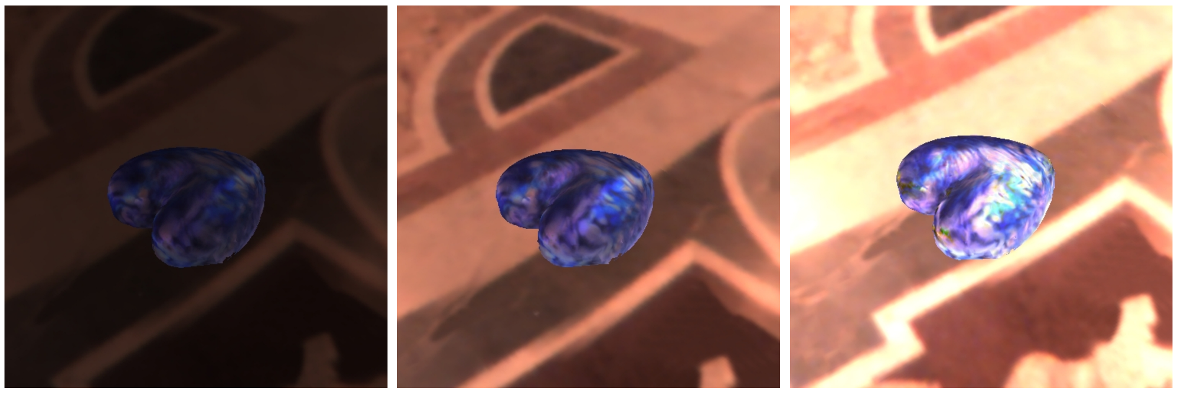


Figure 5.11: Different HDR levels of the heart model illuminated with the St. Peter's Cathedral light probe.

requires extensive software modifications.

5.5.1 Results

We captured two objects, a pitcher model and a heart model on our capture rig under a number of different illumination environments, as shown in Figures 5.2 and 5.11. Capturing a low-dynamic range surface light field from 80 viewpoints took about 40 minutes, and OpenLF processing took another 20 minutes. The high-dynamic range capture multiplies the capture time by the number of exposure levels (we used five exposure levels in our experiments). For the heart model shown in Figure 5.11, the capture took about 4 hours. The light field processing time is only slightly changed, as the demultiplexing recombines these exposure levels into a single high-dynamic range image. There is a small added computational burden as the HDR images are stored as 32-bit floating-point rather than 8-bit fixed-point images.

As discussed in Chapter 3, the OpenLF system has problems with missing data. In the virtual illumination system, this problem is exacerbated by the multiple camera positions. A triangle which is not visible in one camera view may be visible in another, which results in a data mismatch which must be corrected in a post-process. Using the Incremental WLS system would avoid this problem.

5.5.2 Error

An example of this error is shown in Figure 5.12. This image shows the camera blocking part of the screen, thereby reducing the amount of light that falls on the object. In fact, this is a common problem in Image-Based Modeling since the acquisition devices often

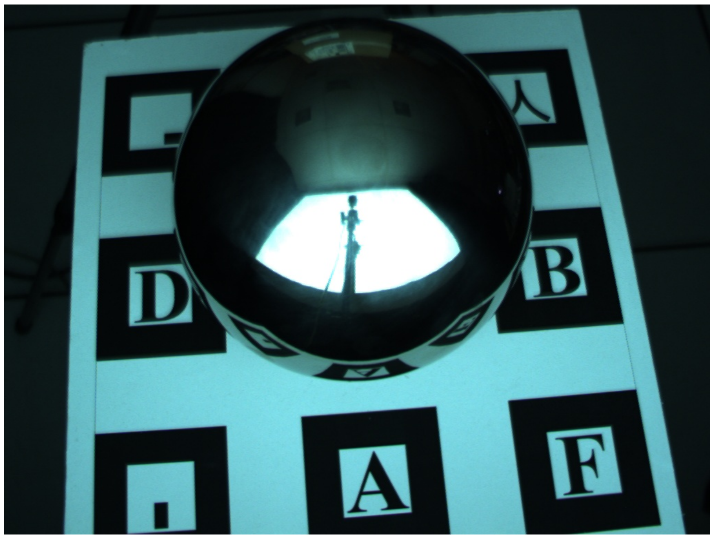


Figure 5.12: An example image from the capture system which shows the camera blocking part of the screen.

interfere with the lighting (Debevec et al., 2000). To quantify this error, we measured the area of the image that is covered by the camera, and compared it to the area that the screen covers. In the 1024×768 image shown, over 3.1×10^5 pixels are screen pixels, and 1.6×10^4 are camera pixels. This means that the camera covers about 5.2% of the light from the screen.

The image shown is the worst-case for camera position. When the camera is positioned to the sides or back, it does not block the light from the projector and the screen. For our system, it might be possible to mount the camera behind the screen so that it peeks through a small hole. This would minimize the light interference.

5.6 Conclusion

In this chapter, I described a system which captures the surface light field of an object under virtual illumination from a light probe. It consists of a projector which shines light onto a screen and reflects onto the object. This configuration enables us to capture a larger portion of the hemisphere than shining the projector directly on the object. To model the physical world, we used high-dynamic range light probes and mapped the values onto the 8-bit levels of the projector. To avoid amplifying noise, we extended the multiplexed illumination algorithm to high-dynamic range imagery.

CHAPTER 6

Summary and Conclusion

6.1 Conclusion

SLFs are a powerful technique for representing the appearance of objects. However, they have not been widely adopted in movies or games. In Chapter 1 I discuss several reasons why this may be true. I also highlighted three particular limitations to discuss in further detail. They are batch construction, extensive resampling, and the restriction of fixed lighting. I addressed these limitations by enabling incremental construction, scattered data approximation, and the ability to capture under synthetic lighting environments.

6.1.1 Incremental Construction

To address the batch construction limitation, I discuss a system for incrementally capturing, constructing, and rendering directionally-varying illumination by incrementally building a low rank linear approximation to the surface light field. Each image is incorporated into the lighting model as it is captured, providing the user with real-time feedback. This feedback enables the user to preview the lighting model and direct the image acquisition towards undersampled areas of the object. Incremental construction allows the surface light field to be viewed interactively using graphics hardware as the model is being constructed. We also provide a novel data-driven heuristic that highlights undersampled areas of the surface light field and directs the user towards effective camera views. This heuristic is used to aid the user in identifying undersampled regions. Our system is an order of magnitude faster than previous systems, and reduces the time necessary to capture the images and construct a surface light field from hours to minutes.

6.1.2 Scattered Data Approximation

The incremental construction technique is very powerful, but it requires extensive resampling of the data. One way to avoid this resampling is to use scattered data approximation techniques. I discuss the Incremental Weighted Least Squares approach to the representation and rendering of spatially and directionally varying illumination. Each surface patch consists of a set of WLS node centers, which are low-degree polynomial representations of the anisotropic exitant radiance. During rendering, the representations are combined in a non-linear fashion to generate a full reconstruction of the exitant radiance. The rendering algorithm is fast, efficient, and implemented entirely on the GPU. The construction algorithm is incremental, which means that images are processed as they arrive instead of in the traditional batch fashion. This human-in-the-loop process enables the user to preview the model as it is being constructed and to adapt to over-sampling and undersampling of the surface appearance. I also compared the Incremental Weighted Least Squares technique to the Online SVD technique to illustrate the difference in compression and rendering times.

6.1.3 Virtual Illumination

Another main limitation of SLFs is that they can only represent the fixed lighting conditions of the environment where the model was captured. If a specific lighting condition is desired, then there are two options: either use a combination of physical lights as an approximation, or capture a full 6D BTF and only use the portion that corresponds to the desired lighting. I describe a third option: a method for capturing a surface light field using the virtual illumination from an environment map. I use a simple setup consisting of a projector, a camera, and tracking fiducials to recreate the desired lighting environment. We determine the correspondence between surface points and rays in the virtual environment using two calibration methods; a fast method for planar screens, and a slower technique for screens with arbitrary geometry. To decrease noise and improve the quality of the capture under low- and high-dynamic range environment maps, I extend Multiplexed Illumination to handle High-Dynamic Range images. To demonstrate the effectiveness of this approach, I show results from objects captured under different lighting environments.

6.2 Limitations

We encountered a number of research issues when designing these systems, some of which we were able to address directly. However, there were a number of issues that we were not able to sufficiently address. Some of these issues are related to the physical constraints of the capture setup. For example, it is difficult in our setup to capture extreme angles due to the projection step. Due to small errors in the mesh or camera pose estimate, these points are often discarded. These points are crucial for representing the highlights that occur at extreme grazing angles.

We also have to deal with various problems relating to extraneous light. One problem is the scattering of light around the lab, which we try to minimize by hanging black curtains on the walls and ceiling. We also have to be careful to avoid certain configurations which can cause the projector to shine directly into the camera. The fiducial board that we use can interfere with capture process by both reflecting light and blocking certain extreme camera angles. A different design for fiducial tracking (or by tracking points directly on the object's surface) could alleviate some of these issues. The camera (and possibly the user) is also in the lighting environment, and can block light. This problem is common to many Image-Based Modeling techniques.

One of the major difficulties that we encountered during the course of this research is that the quality of the SLF is highly-dependent upon a number of calibrations. These include both the intrinsic and extrinsic camera calibration, the registration of geometry with tracking, and the registration of light probe with geometry. We would prefer that the geometry serve only as a coarse approximation of the object, and use the reflectance data to capture the appearance of the model. However, small errors in the mesh can dramatically affect the reconstruction quality, particularly around the silhouette edges. Compounding this problem is the fact that the tracking must be computed very quickly to handle the stream of images from the camera at 30 fps. This combination of the mis-registration of the geometry and the errors in tracking causes the SLF to blur, as is shown in Figure 6.1. This is due to the same point on the object being projected to different points in the SLF. Note that this occurs despite the ability of our system to represent these high-frequency characteristics. This blurring often obscures the very detail that we are using the SLF to represent.

Another limitation of the techniques presented here is that many of the parameters determined in advance. These parameters include geometric resolution, surface and hemisphere resolution (the number of rows and columns in the factorization matrices), placement of WLS centers, and SVD rank. For example, the SVD rank is fixed for all of

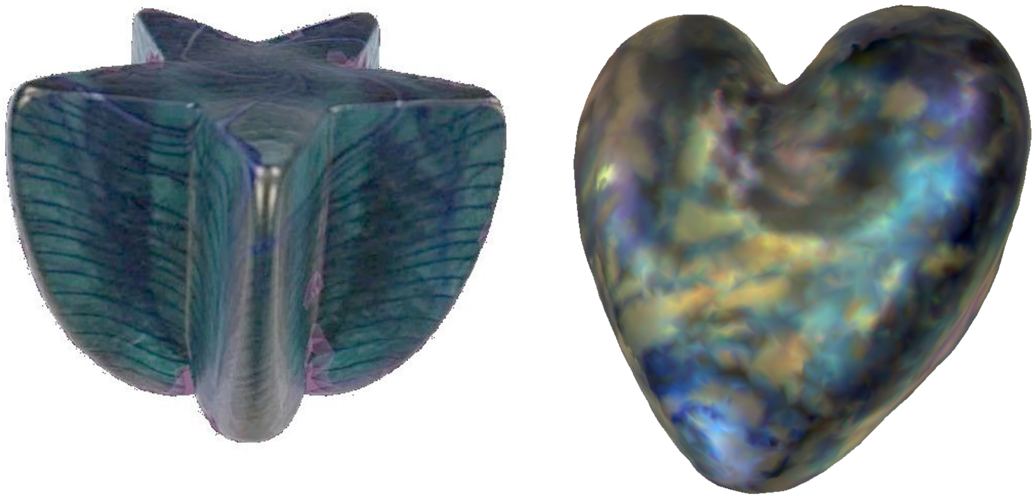


Figure 6.1: The effect of tracking quality on SLF reconstruction. Left: High-quality tracking and calibration from Chen et. al (Chen et al., 2002) enables the SLF to accurately represent surface properties. Right: Errors in real-time tracking and calibration cause the SLF of a similar object to blur out interesting surface properties.

the surface patches, regardless of the different surface complexities. This is problematic since we would like to represent a large class of surfaces, which may vary in frequency of lighting or geometry. Allowing the rank to vary per patch based on an error cutoff would address this problem, but it would require conditional rendering passes. It may also be possible to determine some parameters programmatically by collecting a small set of initial images which could be used to determine the fixed parameters.

6.3 Future Work

There are a number of directions for future work based on this research. In this section I describe several possibilities, starting with the incremental framework, followed by SLF ideas.

6.3.1 Incremental Construction

I believe that the incremental framework presented in Chapter 3 is a powerful idea for a wide-range of Image-Based Modeling acquisition methods. This idea was used by Rusinkiewicz et. al (Rusinkiewicz et al., 2002) to capture geometry in real-time and to provide feedback to the user about the quality of the sampling. This approach could be applied to a number of domains where feedback can affect the quality of the result, such as the capture of Bidirectional Texture Functions (BTFs). BTFs require several orders of magnitude more data to represent, which dramatically increases the capture time. Providing feedback to the user could have a significant impact on the capture time by guiding the user towards important areas of the BTF and avoiding redundant capture. Mathematically, the Weighted Least Squares approach easily generalizes to multiple dimensions by changing the polynomial basis.

I focused on the incremental construction for providing feedback to a user about the quality of sampling. However, it could also be used in a fully-automatic system to provide feedback to a robot gantry or a pan-tilt rig. Such a system could compare a captured image to the current SLF from that viewpoint. If the images are sufficiently different, the captured image would be incorporated into the SLF.

The timing results from Chapters 3 and 4 show that it takes about one to two seconds to incorporate a new image into the approximation. We are interested in decreasing this time by implementing the system on the GPU as in Hillesland (Hillesland et al., 2003). Our initial interest in Online SVD was due to streaming nature, which we thought would be a natural match to a GPU implementation. Implementing these algorithms on the GPU would significantly accelerate the capture process, since these are examples of "computationally-dense" algorithms (Owens et al., 2000). This acceleration would enable us to reduce the capture time and eliminate the redundant storage of data structures on both the CPU and GPU.

It would be interesting to apply the incremental approach to SRFs. Both SLFs and SRFs are the same dimensionality, with the SRFs varying the input illumination instead of the viewpoint. Since it can be more difficult to visualize the coverage of the input illumination directions, the continuous feedback and data-driven heuristics might be very beneficial.

More importantly, the idea of compressing and representing sparse scattered data is common across many scientific disciplines, as demonstrated by the wide-spread use of the SVD. We are interested in examining how an incremental framework could benefit these areas.

6.3.2 Surface Light Fields

An interesting future approach would be to evaluate and improve the data-driven quality heuristic that was presented in Chapter 3. In the current implementation, the heuristic is displayed as a scalar value, but it should be possible to use all of the color channels to convey more information about the sampling. We would also like to investigate other quality functions that take into account the incident light or the reflected light directions. In addition, we would like to formally show that the quality heuristic can decrease the number of images required.

The optimal capture process would acquire radiance data from every direction at every point on the surface. This is impossible, not only because of the sheer amount of data that this would require, but because points are often occluded from certain viewpoints. Some points may even be occluded from all viewpoints (such as points along the bottom of the model which are blocked by the fiducial board). Any ray which does not have data must interpolate or extrapolate values from nearby rays. If the data is so sparse that interpolation or extrapolation is not reasonable, it may be possible to *synthesize* data. In order for this to be valid, the surface must be assumed to consist of patches which have the same statistical properties, in general some sort of repeating pattern. This enables new surface patches to be synthesized by probabilistically combining data from existing patches. There are several well known algorithms for texture synthesis, such as (Ashikhmin, 2001; Efros and Leung, 1999). Tong et al. (Tong et al., 2002) applied these algorithms to BTFs.

One of the primary advantages of artistic techniques to represent appearance is that they allow the artists full creative control. Since the SLF is constructed from images acquired from the real world, it is limited to physical objects. Several authors have proposed techniques for editing light fields (Wood et al., 2000; Chen et al., 2005). One promising approach is to use a data-driven reflectance approach (Matusik et al., 2003). In this paper, the authors acquire a database of BRDFs of different materials, then construct a data-driven representation that allows artists to intuitively navigate through the high-dimensional space of materials. This allows them to create materials to suit needs. One could imagine interfacing with such a database to allow artists to modify a SLF. The artist could select a region of the SLF, along with some example BRDFs from the database. The system would then construct a parameterization that allows the user to "guide" the SLF region towards to the BRDF. This process could be repeated until the artist is satisfied. The major algorithmic challenge with such a system would constructing the parameterization to link the SLF to the BRDF database. This could be done using the SVD techniques from the paper (Matusik et al., 2003), although it may require a resampling step.

One of the primary limitations of the SLF is that it tightly coupled to the geometry. This forces us into a two-step capture process where the geometry must be captured before the SLF capture process can begin. It also causes problems when geometry is poor. An interesting area of future research would be to explore a more loosely-coupled SLF representation which would allow the user to capture the geometry and the SLF at same time. This would not only simplify the capture, but it would eliminate some of the registration errors from capturing the geometry in a different coordinate frame than the reflectance. A loosely-coupled representation could also enable the rendering of deformable geometry such as faces. One approach would be to use a higher-level representation which can be resampled to generate each new representation, which is similar to dynamic reparameterization (Isaksen et al., 2000).

BIBLIOGRAPHY

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. (2003). Computing and rendering point set surfaces. *IEEE Transactions on Computer Graphics and Visualization*, 8(4).

Ashikhmin, M. (2001). Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, New York, NY, USA. ACM Press.

Blackford, L. S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., and Whaley, R. C. (2002). An updated set of basic linear algebra subprograms (blas). *ACM Trans. Math. Softw.*, 28(2):135–151.

Blender. Blender: A 3d modeling toolkit. *blender.org*.

Bouguet, J.-Y. Matlab Camera Calibration Toolbox. *www.vision.caltech.edu/bouguetj*.

Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision*, volume 2350 of *Springer Lecture Notes on Computer Science*, Copenhagen.

Brand, M. (2003). Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining*.

Buehler, C., Bosse, M., McMillan, L., Gortler, S., and Cohen, M. (2001). Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432. ACM Press.

Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. (2001). Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH*.

Chandrasekaren, S., Manjunath, B., Wang, Y., Winkler, J., and Zhang, H. (1997). An eigenspace update algorithm for image analysis. In *Graphical Models and Image Processing*.

Chen, B., Ofek, E., Shum, H., and Levoy, M. (2005). Interactive deformation of light fields. In *Symposium on Interactive 3D Graphics and Games*.

Chen, W.-C., Bouguet, J.-Y., Chu, M., and Grzeszczuk, R. (2002). Light field mapping: Efficient representation and hardware rendering of surface light fields. In *SIGGRAPH*.

Cohen, J., Tchou, C., Hawkins, T., and Debevec, P. (2001). Real-time high-dynamic range texture mapping. In *Eurographics Rendering Workshop*.

Cook, R. L. and Torrance, K. E. (1982). A reflection model for computer graphics. In *SIGGRAPH*, pages 7–24.

Coombe, G., Frahm, J.-M., and Lastra, A. (2007). Capturing real objects under virtual illumination.

Coombe, G., Hantak, C., Lastra, A., and Grzeszczuk, R. (2005). Online construction of surface light fields. In *Eurographics Symposium on Rendering*.

Coombe, G. and Lastra, A. (2006). An incremental weighted least squares approach to surface light fields. In *Graphics Research and Applications (GRAPP)*.

Debevec, P., Hawkins, T., Tchou, C., Duiker, H.-P., Sarokin, W., and Sagar, M. (2000). Acquiring the reflectance field of a human face. In *SIGGRAPH*.

Debevec, P., Yu, Y., and Boshokov, G. (1998). Efficient view-dependent image-based rendering with projective texture-mapping. Technical report, Berkeley, CA, USA.

Debevec, P. E. (1998). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH*.

Debevec, P. E., Taylor, C. J., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics*, 30(Annual Conference Series):11–20.

Delaunay, B. (1934). Sur la sphere vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800.

Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *ICCV (2)*, pages 1033–1038.

Frahm, J.-M., Coombe, G., and Lastra, A. (2006). Calibration of a surface light field capture system. In *ProCams Workshop*.

Furukawa, R., Kawasaki, H., Ikeuchi, K., and Sakauchi, M. (2002). Appearance based object modeling using texture database: acquisition, compression and rendering. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 257–266, Pisa, Italy. Eurographics Association.

Gardner, A., Tchou, C., Hawkins, T., and Debevec, P. (2003). Linear light source reflectometry. In *SIGGRAPH*.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58.

Goesele, M., Lensch, H. P. A., Lang, J., Fuchs, C., and Seidel, H.-P. (2004). Disco: acquisition of translucent objects. In *SIGGRAPH*, pages 835–844, New York, NY, USA. ACM Press.

Golub, G. H. and Loan, C. F. V. (1996). *Matrix Computations.* Johns Hopkins University Press, 3rd edition.

Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The Lumigraph. In *SIGGRAPH*, pages 43–54.

Hall, P., Marshall, D., and Martin, R. (2000). Merging and splitting eigenspace models. In *IEEE Trans. on Pattern Analysis and Machine Intelligence.*

Hanrahan, P. and Krueger, W. (1993). Reflection from layered surfaces due to subsurface scattering. In *SIGGRAPH*, volume 27, pages 165–174.

Harwit, M. and Sloane, N. J. A. (1979). *Hadamard Transform Optics.* Academic Press, New York.

Hawkins, T., Cohen, J., and Debevec, P. (2001). A photometric approach to digitizing cultural artifacts. In *2nd International Symposium on Virtual Reality, Archaeology, and Cultural Heritage.*

Hillesland, K., Molinov, S., and Grzeszczuk, R. (2003). Nonlinear optimization framework for image-based modeling on programmable graphics hardware. In *SIGGRAPH.*

Ilie, A. and Welch, G. (2005). Ensuring color consistency across multiple cameras. In *IEEE Conference on Computer Vision*, volume 2, pages 1268–1275.

Isaksen, A., McMillan, L., and Gortler, S. J. (2000). Dynamically reparameterized light fields. In *SIGGRAPH*, pages 297–306, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Johnson, L. W., Riess, R. D., and Arnold, J. T. (1993). *Introduction to Linear Algebra 3rd Edition.* Addison-Wesley.

Kato, H. and Billinghurst, M. (1999). Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *2nd International Workshop on Augmented Reality.*

Koudelka, M., Magda, S., Belhumeur, P., and Kriegman, D. (2001). Image-based modeling and rendering of surfaces with arbitrary brdfs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 568–575.

Lafortune, E. P. F., Foo, S.-C., Torrance, K. E., and Greenberg, D. P. (1997). Non-linear approximation of reflectance functions. In *Proceedings of SIGGRAPH 97*, pages 117–126.

Larson, G., Rushmeier, H., and Piatko, C. (1997). "a visibility matching tone reproduction operator for high dynamic range scenes". In *IEEE Transactions on Visualization and Computer Graphics*, volume 3.

Lensch, H., Kautz, J., Goesele, M., Heidrich, W., and Seidel, H. (2001). Image-based reconstruction of spatially varying materials. In *Eurographics Rendering Workshop*, pages 103–114.

Lensch, H. P. A., Lang, J., Sa, A. M., and Seidel, H.-P. (2003). Planned sampling of spatially varying brdfs. In *Eurographics*.

Levin, D. (1998). The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1531.

Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *SIGGRAPH*.

Malzbender, T., Gelb, D., and Wolters, H. (2001). Polynomial texture maps. In *SIGGRAPH*.

Masselus, V., Peers, P., Dutre, P., and Willems, Y. D. (2003). Relighting with 4d incident light fields. In *SIGGRAPH*, pages 613–620.

Matusik, W., Loper, M., and Pfister, H. (2004). Progressively-refined reflectance functions from natural illumination. In *Eurographics Symposium on Rendering*.

Matusik, W., Pfister, H., Brand, M., and McMillian, L. (2003). A data-driven reflectance model. *SIGGRAPH*.

Matusik, W., Pfister, H., Ngan, A., Beardsley, P., Ziegler, R., and McMillan, L. (2002). Image-based 3d photography using opacity hulls. *SIGGRAPH*, pages 427–437.

McAllister, D., Lastra, A., and Heidrich, W. (2002). Efficient rendering of spatial bidirectional reflectance distribution functions. In *Workshop on Graphics Hardware*.

McCool, M. D., Ang, J., and Ahmad, A. (2001). Homomorphic factorization of brdfs for high-performance rendering. In *SIGGRAPH*, pages 171–178. ACM Press.

McLain, D. (1976). Two dimensional interpolation from random data. *The Computer Journal*, 19(2):178–181.

McMillan, L. and Bishop, G. (1995). Plenoptic modeling: An imge-based rednering system. In *SIGGRAPH*.

MeshLab. Meshlab: A mesh editing tool. *meshlab.org*.

Micchelli, C. (1986). Interpolation of scattered data: distance matrices and conditionally positive definite functions. In *Constructive Approximation*, volume 1, pages 11–22.

Miller, G. S., Rubin, S. M., and Ponceleon, D. (1998). Lazy decompression of surface light fields for precomputed global illumination. In *Eurographics Workshop on Rendering*.

Mueller, G., Meseth, J., Sattler, M., Sarlette, R., and Klein, R. (2004). Acquisition, synthesis and rendering of bidirectional texture functions. In Schlick, C. and Purgathofer, W., editors, *Eurographics State of the Art Reports*, pages 69–94.

Nealen, A. (2004). An as-short-as-possible introduction to least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. Technical report.

Nicodemus, F., Richmond, J., Hsia, J., Ginsberg, I., and Limperis, T. (1977). Geometric considerations and nomenclature for reflectance. *Monograph 160.*

Nishino, K., Sato, Y., and Ikeuchi, K. (2001). Eigen-texture method: Appearance compression and synthesis based on a 3d model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1257–1265.

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. (2003). Multi-level partition of unity implicits. In *SIGGRAPH*, pages 463–470.

OpenCV. OpenCV: The Open Computer Vision Library. *sourceforge.net/projects/opencvlibrary.*

OpenLF. Openlf: The open lightfield library. *sourceforge.net/projects/openlf/.*

Oren, M. and Nayar, S. K. (1994). Generalization of Lambert's reflectance model. *Computer Graphics*, 28:239–246.

Owens, J. D., Dally, W. J., Kapasi, U. J., Rixner, S., Mattson, P., and Mowery, B. (2000). Polygon rendering on a stream architecture. In *Workshop on Graphics Hardware*, pages 23–32.

P. Lancaster, K. S. (1981). Surfaces generated by moving least squares methods. *Mathematics of Computation*, 37:141–158.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1992). *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press.

Reinhard, E., Ward, G., Pattanaik, S., and Debevec, P. (2005). *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting.* Morgan Kaufmann.

Rusinkiewicz, S., Hall-Holt, O., and Levoy, M. (2002). Real-time 3d model acquisition. In *SIGGRAPH.*

Schechner, Y. Y., Nayar, S. K., and Belhumeur, P. (2003). A theory of multiplexed illumination. In *International Conference on Computer Vision (ICCV).*

Schirmacher, H., Heidrich, W., and Seidel, H.-P. (1999). Adaptive acquisition of lumigraphs from synthetic scenes. In *Computer Graphics Forum*, volume 18(3), pages 151–160.

Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA. ACM Press.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Lin, M. C. and Manocha, D., editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag. From the First ACM Workshop on Applied Computational Geometry.

Shirley, P. and Chiu, K. (1997). A low distortion map between disk and square. *Journal of Graphics Tools*, 2(3):45–52.

Tong, X., Zhang, J., Liu, L., Wang, X., Guo, B., and Shum, H.-Y. (2002). Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH*, pages 665–672, New York, NY, USA. ACM Press.

Wendland, H. (1995). Piecewise polynomial,positive definite and compactly supported radial basis functions of minimal degree. *Advances in Computational Mathematics*, 4:389–396.

Wendland, H. (2005). *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press.

Wood, D., Azuma, D., Aldinger, W., Curless, B., Duchamp, T., Salesin, D., and Stuetzle, W. (2000). Surface light fields for 3d photography. In *SIGGRAPH*.

Zickler, T., Enrique, S., Ramamoorthi, R., and Belhumeur, P. (2005). Reflectance sharing: Image-based rendering from a sparse set of images. In *Eurographics Symposium on Rendering*.