

2013-04-02

Programmable Image-Based Light Capture for Previsualization

Clifford Lindsay
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/etd-dissertations>

Repository Citation

Lindsay, C. (2013). *Programmable Image-Based Light Capture for Previsualization*. Retrieved from <https://digitalcommons.wpi.edu/etd-dissertations/88>

This dissertation is brought to you for free and open access by [Digital WPI](#). It has been accepted for inclusion in Doctoral Dissertations (All Dissertations, All Years) by an authorized administrator of Digital WPI. For more information, please contact wpi-etd@wpi.edu.

Programmable Image-Based Light Capture for Previsualization

by

Clifford Lindsay

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the


Degree of Doctor of Philosophy

in


Computer Science

January 31st, 2013

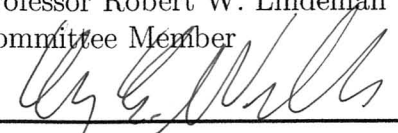
APPROVED:



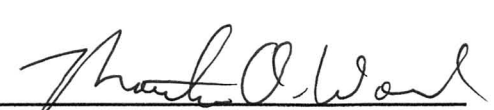
Professor Emmanuel O. Agu
Advisor



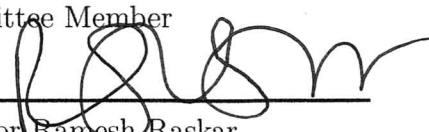
Professor Robert W. Lindeman
Committee Member



Professor Craig E. Wills
Head of Department



Professor Mathew O. Ward
Committee Member



Professor Ramesh Raskar
External Committee Member
MIT Media Lab
Massachusetts Institute of Technology

Abstract

Previsualization is a class of techniques for creating approximate previews of a movie sequence in order to visualize a scene prior to shooting it on the set. Often these techniques are used to convey the artistic direction of the story in terms of cinematic elements, such as camera movement, angle, lighting, dialogue, and character motion. Essentially, a movie director uses previsualization (previs) to convey movie visuals as he sees them in his "minds-eye". Traditional methods for previs include hand-drawn sketches, Storyboards, scaled models, and photographs, which are created by artists to convey how a scene or character might look or move. A recent trend has been to use 3D graphics applications such as video game engines to perform previs, which is called 3D previs. This type of previs is generally used prior to shooting a scene in order to choreograph camera or character movements. To visualize a scene while being recorded on-set, directors and cinematographers use a technique called On-set previs, which provides a real-time view with little to no processing. Other types of previs, such as Technical previs, emphasize accurately capturing scene properties but lack any interactive manipulation and are usually employed by visual effects crews and not for cinematographers or directors. This dissertation's focus is on creating a new method for interactive visualization that will automatically capture the on-set lighting and provide interactive manipulation of cinematic elements to facilitate the movie maker's artistic expression, validate cinematic choices, and provide guidance to production crews. Our method will overcome the drawbacks of the all previous previs methods by combining photorealistic rendering with accurately captured scene details, which is interactively displayed on a mobile capture and rendering platform.

This dissertation describes a new hardware and software previs framework that enables interactive visualization of on-set post-production elements. A three-tiered framework, which is the main contribution of this dissertation is; 1) a novel programmable camera architecture that provides programmability to low-level features and a visual programming interface, 2) new algorithms that analyzes and decomposes the scene photometrically, and 3) a previs interface that leverages the previous to perform interactive rendering and manipulation of the photometric and computer generated elements. For this dissertation we implemented a programmable camera with a novel visual programming interface. We developed the photometric theory and implementation of our novel relighting technique called Symmetric lighting, which can be used to relight a scene with multiple illuminants with respect to color, intensity and location on our programmable camera. We analyzed the performance of Symmetric lighting on synthetic and real scenes to evaluate the benefits and limitations with respect to the reflectance composition of the scene and the number and color of lights within the scene. We found that, since our method is based on a Lambertian reflectance assumption, our method works well under this assumption but that scenes with high amounts of specular reflections can have higher errors in terms of relighting accuracy and additional steps are required to mitigate this limitation. Also, scenes which contain lights whose colors are too similar can lead to degenerate cases in terms of relighting. Despite these limitations, an important contribution of our work is that Symmetric lighting can also be leveraged as a solution for performing multi-illuminant white balancing and light color estimation within a scene with multiple illuminants without limits on the color range or number of lights. We compared our method to other white balance methods and show that our method is superior when at least one of the light colors is known a priori.

Dedications

-I dedicate this dissertation to my son, Noah. Your imagination knows no bounds and that inspires me. My hope as a father is that I can inspire you too. May this example of hard work and persistence inspire you someday. And though every minute spent on this endeavor was time away from you, know that you were always in my thoughts. Without your love and understanding I could have never finished this journey.

-To my lovely wife Ana, whose daily sacrifice has made this dissertation a reality. You stood by me during this process which was just as painful for you as it was for me and for that I am truly grateful. Thank you for being my rescue, my shelter, and my home.

Acknowledgements

I would like to provide a special acknowledgment and thanks to my advisor Professor Emmanuel Agu. You believed in me when I found it hard to believe in myself. You have been a great mentor, guide, and friend. I would also like to thank you for the many arduous hours you have put into my academic work and helping me become a better writer, thinker, and person.

I would also like to acknowledge my other committee members for all their support, insightful conversations, and for directing my academic work. Professor Robert Lindeman thank you for listening to my crazy ideas with an open mind and not letting me get away with anything other than my best work. You have made me a better Computer Scientist by far. Professor Mathew Ward, thank you for the wonderful discussions, thoughtful insight, and dedication to my work. Not to mention how much I enjoyed our "walkie-talkies" together and for organizing my main proving grounds, ISRG. I would also like to thank Professor Ramesh Raskar of the MIT Media Lab for opening up his lab and group to me. Your kindness and graciousness was far greater than I expected. Also, I am extremely grateful to you for lending me your keen insight and command of the field of Computational Photography and Optics, which had a tremendous influence on this work. Partial funding for this work was provided by GAANN Fellowship.

Table of Contents

Abstract	iii
Dedications	v
Acknowledgements	vi
List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Introduction to Previsualization	1
1.1.1 Brief History of Previsualization	6
1.1.2 The Drawbacks with Current Previs Techniques	8
1.2 Motivation: Our Vision of Interactive Previsualization with Full Scene Capture	11
1.2.1 Summary of Dissertation Challenges	19
1.3 Thesis Statement	24
1.4 Dissertation Contributions	26
1.5 Outline of Dissertation	31
2 Related Work	32
2.1 Programmable Imaging	32
2.1.1 Programmable Camera Components	32
2.1.2 Scriptable Cameras	34
2.1.3 Smart Cameras & Other Camera-Computer Combinations	34
2.1.4 Programmable Cameras	36
3 PCam: A Programmable Camera Architecture	37
3.1 Overview	37
3.2 Tile-based Streaming Architecture	39
3.2.1 Overview	39
3.2.2 Stream Processor	40
3.2.3 Tile Based Data Structure	43
3.2.4 Supporting Architectural Features	46

3.2.4.1	Texturing	46
3.2.4.2	High-Quality Anti-Aliasing	47
3.3	Camera Shader Framework	48
3.3.1	Overview	48
3.3.2	Function Classifications	49
3.3.3	Example Shader	50
3.4	Implementation	52
3.4.1	PCam Version 1	53
3.4.1.1	Overview	53
3.4.1.2	UI Server Comparison	53
3.4.1.3	UI Frameworks API Review	55
3.4.1.4	Qt UI Implementation	57
3.4.1.5	Programming Languages & Standards	60
3.4.1.6	Software Packages & Other Frameworks	61
3.4.2	PCam Version 2	61
4	PCamUI: A Visual Programming User Interface for PCam	64
4.1	Overview	64
4.2	Background	68
4.3	Target Audience & Context	69
4.4	Filter-Based Abstraction for On-Camera Processing	71
4.4.1	Visual Filters and Camera Pipelines	71
4.5	Task Determination	73
4.5.1	Task 1: Pipeline Creation	74
4.5.2	Tasks: 2 & 3, Filter and Pipeline Editing	75
4.5.3	Layout & Interaction Scenarios	79
4.5.3.1	Layout	79
4.5.3.2	Interaction Scenarios	80
4.6	Conclusion & Future Work	82
5	Symmetric Lighting Capture and Relighting	84
5.1	Overview	84
5.2	Symmetric Lighting Theory	89
5.2.1	Image Formation	89
5.2.1.1	Camera Model	89
5.2.1.2	Physically-based Lighting Model	90
5.2.2	Light Contribution Estimation through Symmetry	91
5.2.3	Expanded Symmetric Lighting	98
5.2.3.1	Symmetric Lighting in N-Lights	99
5.2.3.2	Degenerate Cases and Computational Complexity	104
5.2.3.3	Error Estimation	107
5.2.3.4	Minimizing Epsilon for Non-diffuse Reflections	110
5.3	The Beta Map	113

5.3.1	Shadow Detection Within The Beta Map	118
5.3.2	Gradient Domain Light Distribution	121
5.3.2.1	Method	124
5.3.2.2	Gradient Domain Operations	129
5.4	Conclusion	131
6	Previzualization and Evaluations	133
6.1	Overview	133
6.2	Scene Relighting	134
6.2.1	Three Point Relighting	137
6.2.2	High-Dynamic Range Relighting	140
6.2.3	Non-photorealistic Relighting	142
6.2.4	Light Editing	143
6.2.5	Previous work on Relighting	144
6.2.6	Relighting Implementation	150
6.2.7	Evaluations	159
6.2.7.1	Additional Results	166
6.2.7.2	Limitations	168
6.3	Light Color Estimation and Calibration	169
6.4	Multi-Illuminant White Balance	176
6.4.1	Overview	176
6.4.2	Related Work	178
6.4.2.1	Multiple Illuminants	180
6.4.3	Implementation	181
6.4.4	Results	183
6.4.5	Conclusion	185
6.5	User Studies	185
6.5.1	User Study #1: Relighting Evaluation	186
6.5.2	User Study #2: Previs Society Survey	191
6.5.3	User Study #3: PCam Expert Review User Study	194
7	Future Work	201
7.1	Expanded Scene Capture	201
7.2	Relighting With Complex Lighting Environments	203
7.3	Surface Reconstruction Using Geometry Maps	205
7.4	PCam Version 3.0	207
8	Conclusions	209
	Bibliography	211
	Appendix A	230
	Appendix B	232

Appendix C	237
Appendix D	244

List of Figures

1.1	A Timeline for how films are made	2
1.2	Storyboard and video frames of the "Bullet Time" sequence	4
1.3	Previs examples from " <i>Day After Tomorrow</i> " and " <i>The Matrix</i> " movies	7
1.4	The workflow of an idealized previs tool on a programmable camera	13
1.5	Our complete vision for an idealized previs interface	14
1.6	Real-time movie making timeline	16
1.7	A collage of example lighting concepts	17
1.8	An illustration outlining our previs components	25
3.1	Comparison between traditional and programmable camera pipelines	38
3.2	An illustration of an abstract stream processing model	40
3.3	A detailed view of the PCam architecture	42
3.4	Comparison of stream processing languages	48
3.5	An outline of the groups of API functions	49
3.6	Chroma keying stream processing example	51
3.7	Flow of data streams from one kernel to another	52
3.8	Early UI screen shots	56
3.9	Software organization of the Workbench camera interface	58
4.1	Current development process for programmable cameras using the API approach	65
4.2	A continuum that describes the target audience	70
4.3	A sample programmable camera pipeline with four filters	73
4.4	Screen shot of the pipeline creation window	75
4.5	Screen shot of the pipeline editing window	76
4.6	Flow diagram depicting the interaction and flow between the various windows of the UI described in this work.	77
4.7	Screenshots of widgets and widget interactions	78
4.8	An Org chart of the UI layout	80
4.9	Screen shot of the preveiw window	81
4.10	Interaction diagram	82
5.1	Several images of devices used to capture lighting	86
5.2	Depiction of the interaction between an illuminated point the sources	94
5.3	The general camera model with two lights	95

5.4	The relative contribution of a pixel in successive images from two lights	98
5.5	An illustration of the geometric symmetry in a 3 light capture system	101
5.6	7 n-vertex polygons in 2D that corresponds to lighting configurations	102
5.7	The degenerate form of a triangle	105
5.8	Error estimate due to specular reflection	109
5.9	Separation of diffuse and specular reflection for Symmetric Lighting .	112
5.10	A modified version of the Rendering Equation	114
5.11	Beta Map creation method	116
5.12	A simplified relighting example performed as texture map operations	116
5.13	Example shadow detection using threshold values and a β map . . .	119
5.14	Example shadow detection method	120
5.15	Large gradients corresponding to edges indicated by the red arrows. .	127
5.16	Results of the edge suppression technique	129
6.1	Images of a candy dish relit with our application	136
6.2	Removing lights and relighting	137
6.3	3 Point lighting example.	138
6.4	Two and three point lighting of the familiar sphere scene	139
6.5	Relighting of a scene with HDR lighting added	141
6.6	NPR image relighting	142
6.7	Modifying scene lighting by editing the β map	144
6.8	Diagram illustrating the relighting workflow	152
6.9	Image of Lego scene with the lighting setup described in this work .	158
6.10	Plot the error related to specularity	161
6.11	Visual comparison of the error associated with specular reflection .	162
6.12	Relationship of the error and ground truth images	164
6.13	Visual comparison of the error associated with our relighting application	165
6.14	Original image of a food scene compared to a render of the relit scene	166
6.15	Original image of a Lego scene compared to a render of the relit scene	166
6.16	A screen shot of the desktop application developed for relighting. .	167
6.17	CIE 1931 Chromaticity diagram with Planckian Illuminant Locus. .	170
6.18	Influence of Gamma correction on the lightness of an image	175
6.19	Image comparison of a scene with multiple light sources	177
6.20	Comparison of our method, ground truth, traditional white balance .	184
6.21	Demographic information and background information regarding the subjects in this user study.	189
6.22	Qualitative questions regarding the use and realism of the virtual relighting software.	190
7.1	Approximate representation of the geometry from the depth map and depth edge map.	202
7.2	The Geometry map creation process	206

List of Tables

- 3.1 A comparison of various features of four different UI frameworks we considered for implementing the camera interface. 54
- 5.1 Truth Table For Cross Projection Tensor Operations 128
- 6.1 Estimate of lighting color and the error using the average of the image pixel color rounded to the nearest integer. 173
- 6.2 Estimate of lighting color and the error using the mode of the color values instead of the average. 173

Chapter 1

Introduction

1.1 Introduction to Previsualization

Movie making is the process of storytelling through the recording of creative visual and auditory scenes. Often this endeavor involves matching the director's artistic vision for an initial story or idea to the physical configuration of the movie set, actor's portrayal of the story, camera options, sound, and editing. The movie making process is generally comprised of three main stages, *pre-production*, *production*, and *post-production*. In pre-production, all the preparations for the entire movie making process is performed, including but not limited to script writing, set building, hiring of actors and other personnel to work on the movies. During pre-production the director and cinematographers decide on which types of camera equipment and their various photographic options will be best for conveying the story. When all preparations have completed, the movie production stage is performed, where scenes are acted out and captured via video cameras. The process of recording the actor's performance is often called principle photography. The last stage of movie production, called post-production, is performed after the principle photography. Post-

production is where the various shots captured during production are edited, sound and music is integrated, and visual effects are added to complete the shots. As the movie making process comes to completion they are marketed and distributed for viewing. Each stage of this process are illustrated in Figure 1.1 along with some additional tasks often performed during each stage.

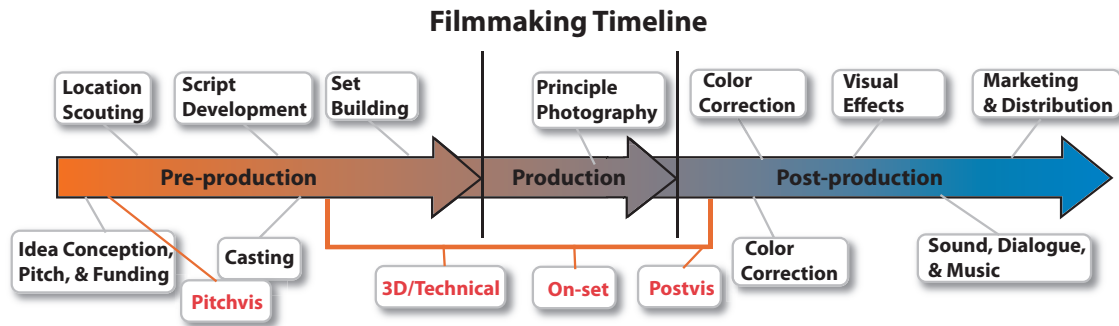


Figure 1.1: A general timeline for how films are made based on the description from [OZ10]. The process of making a film is generally divided into three stages; pre-production where all preparations for making the movie is done, production where the principle photography is captures, and post-production where additions to the movie is made. Previsualization is done as part of the Pre-production or done during production. This figure lists some of the major tasks that may be performed during each of the three movie making stage, including four different types of previs (Pitchvis, Technical, On-set, Postvis).

At various points in the movie-making process, there are many cinematic or cinematographic options movie makers can use to augment the story telling process. For example, they often use focus, camera angles, lighting, perspective, and camera motion to accentuate parts of the story. These options must be weighed carefully by the movie makers, as subtle changes to these cinematic options can result in drastic differences to the look and feel of the movie. To help convey their artistic vision for the story, movie makers will typically plan out the sequence of cinematic elements prior to shooting the movie or a scene. This plan can then be utilized by the movie making crew to gauge the requirements for shooting each scene or as a means for the movie makers themselves to visualize and validate their cinematic choices to make

sure it fits the artistic vision. This process of planning and visualizing a sequence of cinematic elements is often referred to as *Previsualization* (previs) and is used extensively during the pre-production and production stages of movie making. The term previs has is often used to describe any form of planning or conceptualizing a shot for a movie such as storyboarding, pre-rendering, and pre-photography and can include a variety of medium such as images, sketches, and renderings. While there are several advantages to using previs as an aid for filmmaking, its primary function is that it allows filmmakers to vary the different staging and art direction options - such as lighting, camera placement and movement, stage direction and editing - before principle photography without having to incur the costs of actual production [Fer98]. By understanding the effects of the various options available to the filmmakers prior to shooting, previs can help to minimize reshoots, save time, money, and facilitate the creative process [OZ10].

In the context of movie making, previs is often used as a collaborative tool for disseminating how a particular is to be translated into film. From the Visual Effects Society (VES) Handbook of Visual Effects, previs is defined as "a collaborative process that generates preliminarily versions of shots or sequences" [OZ10]. Figure 1.2 illustrates how the directors of the movie "The Matrix" used previs as a collaborative process. In this figure, the directors commissioned a series of hand-drawn images (top), called Storyboards, before filming a complicated action sequence. They did this to solidify their ideas and help convey to the cinematographers and film crew how they wanted the shots to be setup. As can be seen by the similarity of final video frames and the Storyboards in Figure 1.2, this type of previs resulted in an effective communication of the director's artistic vision for the sequence.

Although previs is not a requirement for making a movie, it is quite ubiquitous in the movie making industry. This is a testament to the benefits that previs has on

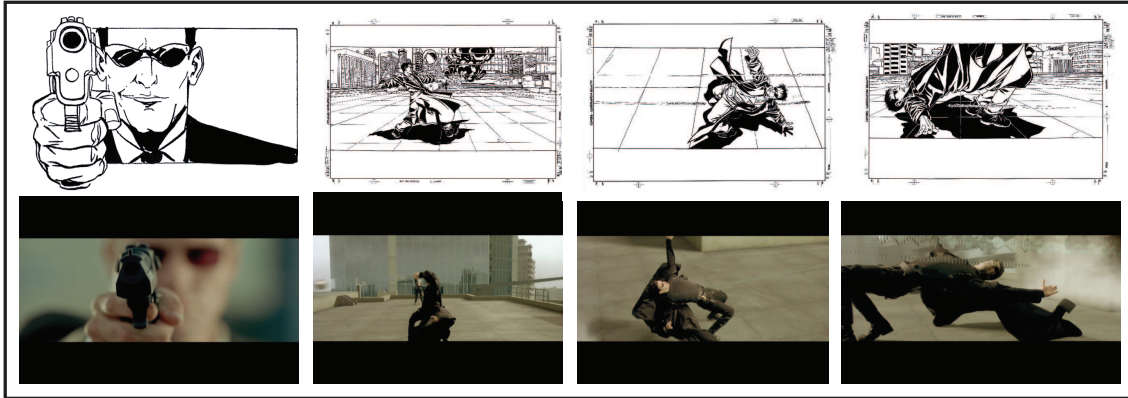


Figure 1.2: Storyboard and corresponding video frames of the "Bullet Time" sequence from the movie "*The Matrix*" (1999), Warner Bros Entertainment. This figure shows how the movie's creators previsualized a complex action sequence using the Storyboarding technique, which was then reproduced in the production of the movie [Bro99].

the movie making process, which can be an aid to everyone from directors to actors to producers and even studio executives. There are two main benefits of previs, saving time and therefore money and producing a higher quality movie [NIK11]. Fundamentally, previs is a tool for allowing creative control of artistic and technical exploration prior to committing to any one parameter before filming. This can benefit the movie in terms of quality, reduction of mistakes, and ultimately money saved.

Previs can be accomplished using a variety of mediums, which can be divided into two categories, traditional artistic mediums and digital mediums. The traditional medium methods include the use of Storyboards, Animatics, photography, or clay models. Historically, Storyboarding has been the preferred technique for previs [JOH07], which are hand drawn or painted scenes that are assembled in a sequence matching the timeline of a story. In the past few years, there has been a shift in previs techniques from using traditional mediums for movie visualization to the use of digital technologies such as video game engines and Digital Content Creation tools.

Digital tools are used to model, pre-render, or provide motion planning for complex scenes (i.e., scenes with a significant amount of stunts or complex visual effects). An example of digital previs is the use of 3D graphics applications for previewing a scene, which is referred to as 3D previs and is now the predominant way to preview or plan an action sequence or a shot with complicated camera movement [JOH07]. This shift from hand drawn to digital previs techniques has had such an influence on movie making that the Joint Subcommittee on Previsualization between VES, the Art Directors Guild, and the American Society of Cinematographers have proposed a redefinition of *previs* as using "3D animation tools and virtual environments for visualizing or previewing a shot or movie sequence" [OZ10].

This shift in previs methods is due to the advantage 3D applications have over traditional techniques. The advantage is that once a scene is digitally modeled (either approximately or precisely by means of Technical previs methods), visualizing changes to the scene can be done interactively and the effects of many potential changes can be previewed quickly. Storyboards on the other-hand are not interactive. If aspects of the Storyboard needs to be altered, part or all the Storyboards must be redrawn, which takes time. This quote from Alex McDowell¹ explains the benefits of 3D previs tools, "With 3D previs, what you end up with is a tool that ultimately gives everybody access to a much deeper level of information than has ever been accessible to production...in pre-production" [Des03].

There are several other types of digital previs methods that are generally used during the movie making process which are tailored to the specific context in which it is used. For example, *on-set previs* refers to the real-time integration of captured video with computer generated (CG) elements to aid directors. *Technical previs* is the term coined for the accurate, real-world measurement of set layout and camera

¹Alex McDowell is production set designer and Visual effects artist whose credits include Bee Movie, Watchmen, Fight Club, The Terminal, and Minority Report.

placement for seamless integration into virtual environments and is often utilized in *Postvis* (previs for verifying a shot with CG elements prior to editing) or for visual effects. *Pitchvis* (pitching a movie using previs) uses 3D rendering and interaction to illustrate a movie or shot idea in order to pitch the idea for funding a movie project.

1.1.1 Brief History of Previsualization

The first known Previsualization technique, Storyboarding, used by Disney Studios circa 1930 included hand-drawn panels which outlined the high-lights of a scene. The filming of Storyboards and editing them to a sound track soon followed and this process became known as the Leica Reel. In the 1970's, cost-effective video equipment became available and the Leica Reel morphed into a technique called Animatics. Animatics added restricted motion and camera angles to video sequences of Storyboards. Storyboards in Animatics were then replaced with hand-controlled models and figures used to plan complex sequences in the Star Wars Trilogy [Kat05a]. The introduction of computer generated imagery or more specifically 3D rendering originated in 1988 by Lynda Weinman for the movie Star Trek V: The Final Frontier. In this movie, the motion of the Starship Enterprise was visualized using primitive animation programmed on a computer in order to provide Previsualization of the scene. Later, video game technology was used to provide pre-planning camera movements for the movie the Abyss [Kat05b]. The use of 3D graphics applications or software for Previsualization has come to be known as Digital Previsualization or 3D Previsualization [OZ10].

More recently, Previsualization has developed into a full-fledge industry employing a wide variety of techniques and medium. On larger budget projects, the director works with artists in a visual effects department or with dedicated previs

service companies. Previs now routinely include music, sound effects and dialogue to closely emulate the look of fully produced and edited sequences. Because of the effort involved in generating rich previs sequences, they are usually employed for complex or difficult scenes that involve stunts and special effects. To this day, digital video, photography, hand-drawn art, clip art and 3D animation are all still being used either singly or in combination to preview sequences alongside some of the more complicated techniques just mentioned. Figure 1.3 shows some example previs shots on two recent movies, "Day After Tomorrow" and "The Matrix".

For this project, we restricted our use of previs to the recreation of static movie sets for the purpose of visualizing on-set lighting changes. This dissertation reserves the incorporation into our previs framework the planning of action sequences, camera motion, complex visual effects, motion capture, sound, or acting as future work.



Figure 1.3: Previs examples: The left image is a preliminary render and final shot for the movie "Day After Tomorrow" [Boe07] and right image is a storyboard and final shot for the movie "The Matrix" [Bro99]

1.1.2 The Drawbacks with Current Previs Techniques

Despite the popularity of using digital techniques for previs, Storyboarding is still used by almost all movie makers in conjunction with digital methods, from the amateur to the high-budget movie studios [JOH07]. The continued use of Storyboards can be attributed to its low cost, low complexity, and the fact that the content is limited only by the artist’s imagination. This notion highlights several drawbacks with current previs techniques, which are cost, complexity, and the limitations of each individual technique. In this section we describe some of the limitations of the previously defined methods, 3D and on-set previs.

3D and interactive previs is generally only used by movie studios with large budgets who can afford to hire visual effects companies that specialize in 3D previs to handle the complexity of setup and operation of the specialized 3Dprevis software and hardware [JOH07]. Several companies exist who specialize in 3D previs, such as Pixel Liberation Front (PLF), Industrial Light and Magic (ILM), or Persistence of Vision (POV). Also, 3D previs is generally used for planning a shot or sequence of shots *prior to actually shooting* and as such generally 3D previs uses virtual cameras, sets, and actors and does not use live camera feeds therefore is not a viable tool for viewing shots. This specialized service, software, and hardware needed for measuring and recreating the scene digitally can add significant cost and time to the movie’s production. Although specific monetary figures are not available for movie production budgets, recent posting on the Previsualization website forum [Poh10] has industry insiders speculating anywhere from five to seven figures in cost for all types of previs combined. George Lucas has publicly stated that for Star Wars: Episode III, ”digital previs easily trimmed ten million dollars off the film’s budget” [Sza95]. Similar statements have been made by visual effects veteran and creative director at Pixel Liberation Front Ron Frankel, where he said ”The cost

for using previz on a film is measured in the tens of thousands of dollars. The savings associated with previz on a film are measured in the hundreds of thousands of dollars,” [Doy02]. The cost and complexity of 3D previz generally precludes the use of intricate previz techniques - techniques that use computer generated imagery, analyze scene data, and use complicated 3D scanning technologies - in low-budget movies, independent films, and especially novice or amateur projects.

Since 3D previz tools are used for planning a shot or sequences of shots, they are not generally used to preview recorded video or film footage. This is because the 3D scene representation is only an approximation to the real scene and does not use video as the primary input. On the other hand, *on-set previz* is primarily used to view the video of the shots as they happen, usually through a device called a ”video assist”. This type of previz has a much lower complexity and cost than 3D previz because it requires only a separate screen through which video is fed for viewing and minimal processing. Because of the simplicity of this device and it’s low cost, it is generally used on every movie set in some fashion. More recently, advances in video assist hardware systems for *on-set previz* now provide additional processing capabilities with limited complexity processing of video feed for tasks such as real-time blue/green-screen replacement or color correction.

Systems related to the video assist hardware, which are high-end augmented camera systems for real-time viewing of video feeds, have also become more popular in recent years. These systems allow cinematic cameras to be connected directly to large computer servers in order to provide additional processing capabilities. This type of previz system, which utilize large mobile trailers for storing computer systems, have limited mobility of the user or camera but provides significantly more processing capabilities to perform simultaneous tasks such as background replacement, motion capture rendering, color correction, and rendering of computer gen-

erated objects. Although these are dedicated specialized techniques, directors and cinematographers are requesting more of these features, which indicates the growing desire of directors and other movie makers to acquire the capability to view cinematic elements in real-time and interactively instead of waiting for them to be applied by a visual effects crew during post-production. This type of previs can be used to verify a shot and the cinematic choices the directors used, which is sometimes referred to as post-vis. Often times the limiting factor for post-vis, is that the director may have to wait several hours or days to view the shot due to the effort required to applied all the processing. In general the downside of *on-set previs* technique is the reduced processing and rendering capabilities compared to 3D previs, mobility when the camera feeds are attached to computer systems, or time when the shot has to be processed off-set.

As previously mentioned, *on-set previs* usually provides very limited processing of the scene, unlike another type of previs called Technical previs, which can provide a more detailed representation of a scene by using capture techniques, can be used as additional input for specialized rendering software run on server farms. For example, Technical previs can include the use of Lidar scanners or hand measurements to record depth and approximate geometry of the scene [FZ09], which is then used to create visual effects in post-production. Individual objects can be scanned and photographed to provide shape, material, and reflection data for use in 3D previs or post-production visual effects [OZ10]. Additionally, relighting can be performed to capture actors and objects under different lighting conditions. This is usually done during the post-production phase of the movie making timeline (as seen in Figure 1.1) and is utilized mainly in visual effects.

Generally, all techniques for scene capture are done in an offline fashion and utilize methods that are not suitable for real-time processing or use complicated and

expensive capture equipment. An example of such an offline technique is the state of the art relighting developed by Einarsson and Debevec [ECJ⁺06], which requires a specially designed performance stage with 30-50 2400fps high-speed cameras and thousands of controllable lights to capture actors and props in a lighting agnostic environment to facilitate relighting in post-production. The disadvantage of such a system is that the set or actors need to be moved to this facility in order to perform relighting, which sometimes is not feasible. Currently there is no method to perform relighting of a set without the use of expensive and separate relighting facilities.

1.2 Motivation: Our Vision of Interactive Previsualization with Full Scene Capture

Our grand vision for the perfect previs tool is one that provides all the capabilities to completely manipulate and render physical scenes in a virtual manner. Our motivation for creating an idealized previs tool is to allow unbridled creative freedom for movie makers to plan, experiment, and validate all the artistic choices that must be made during the filmmaking process. We believe this would lead to better movies created in shorter time and be expressed in a fashion closer to the vision of their creators. Because our tool would emphasize the capability to edit all aspects of the scene virtually, this tool would also change the workflow of the movie making process by incorporating more capture techniques into the first stages of the movie making process (pre-production and production). Inspiration for our previs tool comes from similar tools designed for creating, editing, and rendering completely virtual scenes created with Digital Content Creation tools (DCC tools), such as Maya and 3D Studio Max. The difference being that our previs tool would create and operate on virtual representations of real scenes from captured physical data. The rest of

this section describes the details of our vision for an idealized previs tool and some of changes to the current movie making workflow that would result from using this tool.

In the previous section we described 3D, Technical, and On-set previs methods and some of their drawbacks and limitations. These drawbacks can hinder the creative exploration of options for changing scene elements. Despite these drawbacks, each method provides a vital function to the previs process as a whole, such as 3D rendering and manipulation capabilities from 3D previs, precise capture and measurement of Technical previs, and video input and processing from on-set previs. These previs methods have little overlap in functionality because they were designed for different tasks, integrating their core functions would complement each method and alleviate the drawbacks discussed in the previous section. Therefore, we believe the ideal previs system would be the union of the core features from 3D, Technical, and on-set previs.

A new workflow for our idealized previs system would capture all scene data, convert the data to an editable and renderable format, and finally provide a software interface to allow the user to change and render the scene interactively. To accomplish this, a platform that integrates all captured scene data into a single interface analogous to those used in DCC tools as seen in Figure 1.4. Then using this interface, the user can be free to create and modify the scene according to their artistic vision. Figure 1.5 shows an example scenario where an object in the scene has been selected by the user via the interface (red dotted line). The interface then provides all the options for editing the real object virtually such as textures, reflectance, geometry, and lighting changes and seeing the changes in real-time. Additionally, nothing precludes this interface from rendering purely virtual objects as well; since the scene has been fully captured then all virtual objects can be rendered

with real lighting and vice-versa. In essence, what we are describing is no longer just a tool for previs but a tool that could be used visualizing post-production tasks as well. These tasks could include but are not limited to color correction, visual effects, relighting, editing, and thus producing as close to the finished rendering of the scene as possible during the production phase of movie making. This would free any restrictions on any creative or artistic aspect of the movie making process for directors as well as reduce the time spent on post-production. Additionally, this could lead to more rapid movie making or even *real-time movie making*, where the movie is essentially finished when the shooting is done without the need for a lengthy post-production process. For real-time movie making with this idealized interface to become a reality, it would be necessary to integrate most aspects of post-production into the production stage of movie making.

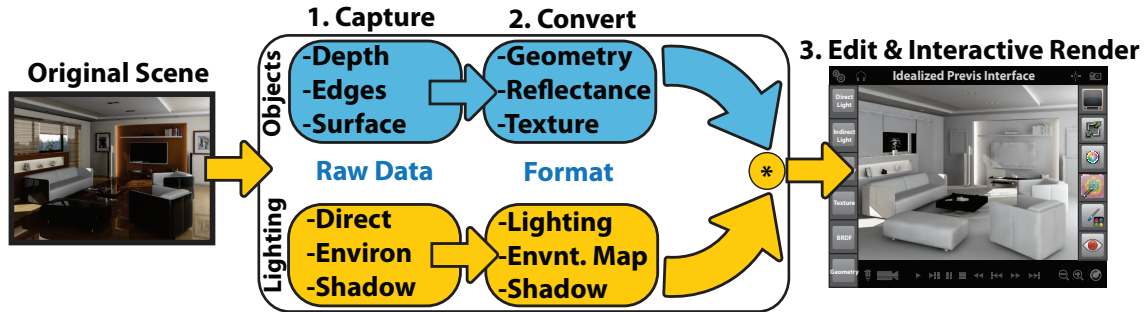


Figure 1.4: The workflow of an idealized previs tool on a programmable camera: 1) Raw data is sensed and then 2) converted into scene data to represent the scene virtually. The scene data can then be 3) edited and interactively rendered using a 3D rendering application.

When filming a movie, ideally a full capture of all aspects of the set and actors would be performed in order to have full control of the artistic "look and feel" of the movie. Full scene capture consists of capturing a geometric representation of each object, including the spatial relationships that object has with the camera, lights and other objects. Additionally, each object has reflectance and appearance data that

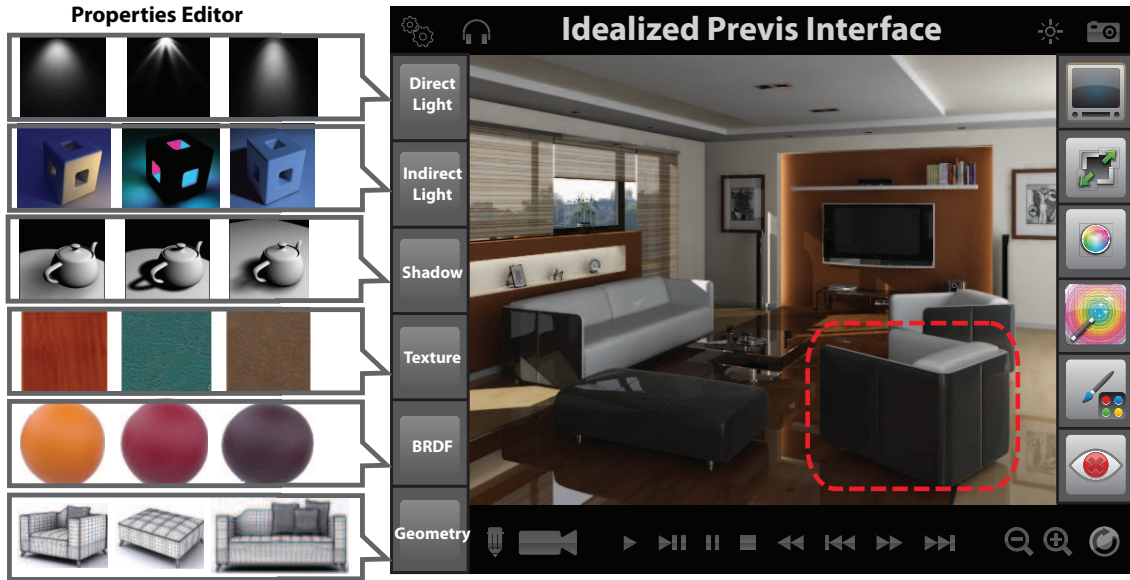


Figure 1.5: Our Vision: As seen in this camera interface mock up, the full vision for an idealized previs interface includes the ability to edit all of a real scene’s properties including but not limited to scene geometry, object reflectance, texture, shadows, and virtually edit real lights. A user would select the object in the scene to edit and use the properties editor to the left to modify it. This would provide on a camera the same level of control for visualization as tools dedicated to virtual scene rendering, such as Maya or 3D Studio max.

must also be captured, which includes texture, bidirectional reflectance distribution functions (BRDFs), subsurface light scattering, and light that objects may emit. Finally to complete the capture, all lighting within the scene must be captured or estimated, including direct, shadows, indirect, and environmental lighting. Full capture of all aspects of a scene is currently feasible, but directors often choose to only capture and view a small portion of the available scene data if any at all. Generally, scene capture falls under the duties of the visual effects crew, in which the captured data is primarily used in post-production tasks. Therefore, the director has little or no direct control over the captured data or how it is visualized and usually only sees the end result of the visual effect. This produces a huge creativity void for the director who usually defers to creativity decisions to the

visual effects crew on matters of post-production. Also, little if anything is done to integrate the captured data into a single interface for viewing and editing. The data is usually captured in a disparate way for specific post-production tasks and therefore distributed to individual visual effects personnel for performing that specific task, such as incorporating into visual effects or calibration.

Unfortunately we believe that such an idealized previs interface cannot be realized until a suitable camera architecture is developed for this purpose. Recent advances in manufacturing and technology have made untethered computing, mobile graphics, and high quality imaging ubiquitous. In turn, this ubiquity has driven down the cost of mobile devices and made them widely available. The combination of abundant computing power, high quality imaging, and low acquisition costs have created an opportunity for low-budget, independent filmmakers, and amateurs to acquire the same high-quality technology that is used by big movie studios to do sophisticated Previsualization and interactive visualization of scenes. Unfortunately, these technologies have not been integrated into a framework specifically designed for low-cost Previsualization for the masses. We call this disparity *The Interactive Previsualization Problem*, which entails providing accurate scene capture, modeling and interaction techniques for all types of previs that can be used in conjunction with low cost imaging and abundant computing power. To alleviate this disparity, we've developed a fully programmable camera architecture which can be utilized to implement the aforementioned idealized previs interface. The camera architecture for this project, which we call *PCam*, can be programmed to accommodate an arbitrary processing pipeline to accommodate any level of scene processing complexity. For example, PCam could be programmed to capture video, apply color correction, integrate virtual objects, and allow modifications to real objects in the scene virtually as depicted in Figure 1.5 all of which we describe as necessary for an ideal

previs tool.

We believe that PCam is an ideal solution to the previously mentioned disparity for previs because it provides a low-cost, mobile image capture system that can be arbitrarily programmed to fit a director’s visualization needs. PCam would solve the limited rendering capabilities of *on-set previs* devices by providing rendering capabilities similar to 3D previs systems while providing live video feed, thereby bypassing the limitations of 3D previs. Additionally, PCam could also integrate capture methods and processing capabilities to integrate captured scene data for real-time rendering unlike Technical previs. Compared to Figure 1.1, Figure 1.6 presents a new workflow that puts the director more in control of the choices and the creative aspects of the whole look of the film and the movie production time is drastically reduced. With the exception of marketing and distribution, almost all post-production tasks could be moved to production stage.

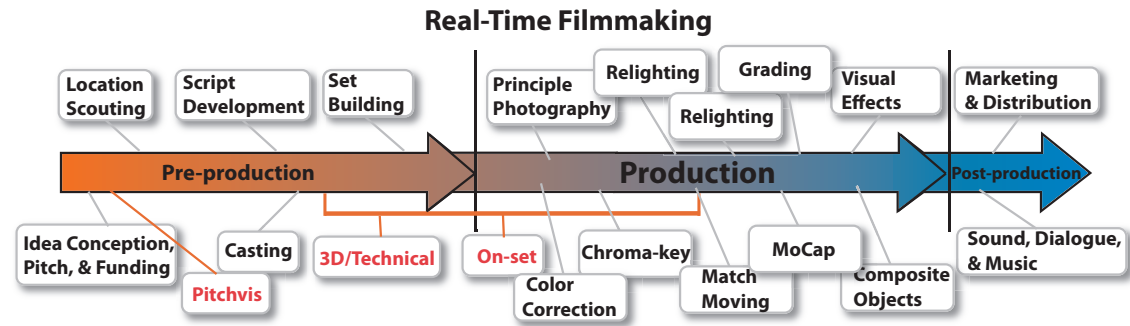


Figure 1.6: Real-time movie making timeline (modified from Figure 1.1): By moving most of the post-production tasks into production, we can envision a time when blockbuster movies can be made in real-time. This means that directors and cinematographers can view what the actual final look of a scene or shot when the video is filmed rather than months later. For example, replacing Green-screen or Chroma-key elements with virtual objects can take several days to weeks and not in real-time, as this task is primarily performed by a different group of people called the visual effects group.

As previously mentioned, full capture of the all scene properties is currently feasible and has been studied academically. While integrating all the aforementioned

capabilities in to PCam would be a great proof of concept, to do so would be herculean undertaking and would drastically increase the cost and time for this project. Therefore, for this dissertation we have chosen to research an area of scene capture that has not garnered much attention as the other scene properties. Specifically we focused on capturing and estimating the lighting, independent of the other scene properties. Scene lighting has a dramatic effect on the artistic aspects of the scene by evoking mood, directing attention to something specific, and allowing the director to convey thoughts and ideas. Figure 1.7 highlights several examples of the use of cinematic lighting; for example the top left of the figure draws your attention to the oncoming train but the top center of the figure focuses your attention on the hero and the top right of the figure gives an angelic aura to the person in the scene.

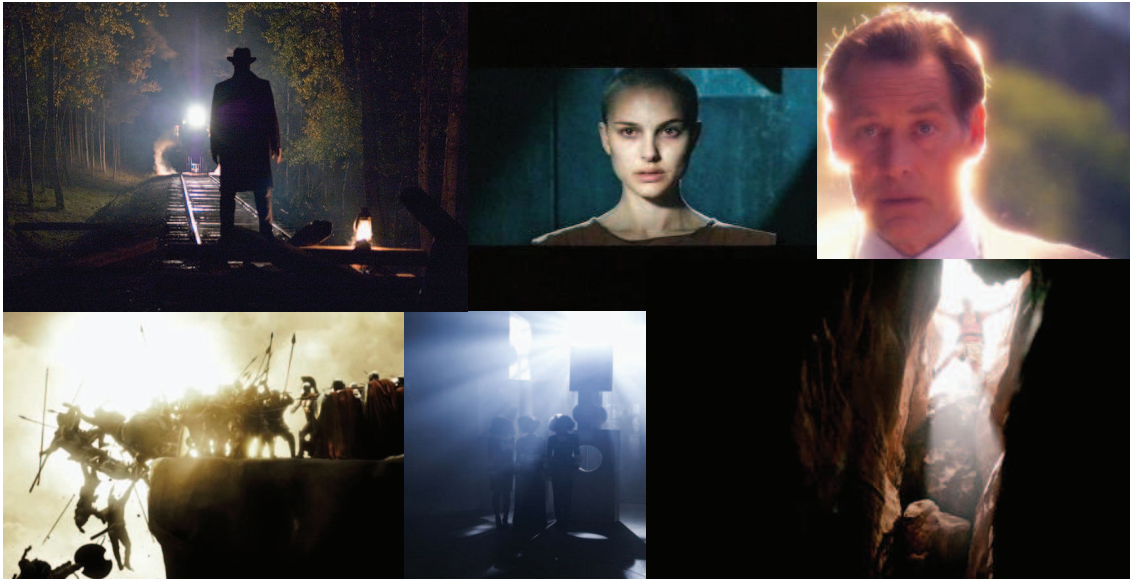


Figure 1.7: A collage of example lighting concepts; top left is a scene from the HBO show "*Dexter*"(2010), top center is a spotlight on the main character from the movie "*V for Vendetta*"(2005), and left top is a backlight shot of another character from the HBO show "*Dexter*"(2010). Bottom left is a backlit image of a scene from the movie "*300*" (2006), center another backlit image from a Japanese music video, and bottom right is illumination focusing our attention in the movie "*127 hours*" (2010)[bEP11].

In addition to PCam, the work of this dissertation specifically focused on a novel method for capturing and relighting the direct lighting emitted from multiple individual light sources and their relationship to the other parts of the scene. We call this method *Symmetric Lighting* and it not only provides the ability to capture but edit and manipulate the lighting so as to provide the director with tools to preview physical lighting changes, virtually, in real-time. This would provide directors the freedom to iteratively change the on-set lighting, fine tune the color, intensity, and direction with instant feedback, without having to wait on the lighting crew to physically alter the properties of the light or completely change the lights altogether. Our personal communications with a movie industry insider and academy award winning researcher Paul Debevec, highlights this issue; "Since more than half the time on-set is involved in setting up lighting, relighting enables directors to correct inconsistencies in lighting of actors or props only" [Lin10]. Additionally, relighting has become an active research area in the movie industry in recent years due to what Paul Debevec describes is the issue that "committing to a particular lighting choices at the time of filming is troublesome when so many other elements are decided in post-production" [Lin10]. These inconsistencies in lighting degrade the quality of the look of a movie if not fixed in post-production or can cause a reshoot with lighting changes, which increases the expenditures of the movie, making relighting an attractive tool for directors. Unfortunately, current relighting techniques only focus on correcting the lighting of the actors separate from the set, whereas the aim with our Symmetric Lighting method is to allow the director the ability to get it right in accordance with his or her vision, the first time, on-set. Additionally, our relighting can be integrated into the previous described ideal previs interface implemented on PCam for interactive relighting, which Paul Debevec says "[In cinematic lighting design] it is important for the director or lighting artist to change the lighting

interactively” [Lin10].

Symmetric Lighting provides a first step toward complete lighting control by allowing the user to edit and manipulate the direct lighting contributions. Our future work includes a vision of complete control of lighting, which entails being able to control indirect lighting phenomena as well. By adding indirect lighting control we can create plausible and implausible renderings that can accommodate the artistic vision of directors to create works that use cinematic lighting for evoking affective responses from their viewers as is seen in Figure 1.7. A vision of being able to render completely new lighting throughout the scene is the ultimate goal of relighting. This brings forth ideas of using techniques such as ray tracing or photon mapping for real and natural scenes in ways that were only intended for virtual scenes. Such a vision would include phenomena such as subsurface scattering and participating media effects even when these properties may not exist in the real scene, to provide an unprecedented control of the look and feeling of the scene.

1.2.1 Summary of Dissertation Challenges

Listed below are the over-arching challenges faced during the development of this dissertation as well as for the fully-realized future version of our previs interface. Specifically, these challenges arise when developing a software framework and hardware technologies that is intended for capturing, analyzing, and rendering of all scene properties, at interactive rates, on a programmable camera. We present these challenges to provide a broad-scoped view of the complexity involved in fusing the different previs types and their respective technologies.

- ***Integrating Scene Data from Different Capture Techniques:*** Faithfully reconstructing a real scene digitally can be done in several ways, such as with laser scans, stereo, sonar, or direct measurement. The benefits and

limitations of these techniques vary drastically in speed, accuracy, and cost, which can affect how easily they can be integrated into a previs framework. The challenge lies in understanding and determining which capture technique is most suitable for *on-set previs* and determining the best methods for integrating this data into our previs framework. Additionally, once a capture method is chosen, what methods for noise reduction is appropriate for use within an *on-set previs* framework. Finally, immediate feedback is imperative for *on-set previs* and providing less emphasis on the quality of render is generally acceptable in the previs community. Therefore, we try to incorporate a capture technique that is fast but provides a good trade off with respect to speed and high-quality scene capture and render.

- ***Data Inversion and Conversion:*** Most capture techniques provide data that requires conversion in order to be re-rendered in previs, such as point data captured from a laser scanner, which needs to be converted to mesh data before rendering. Also, some data from capture techniques will need further processing or inversion in order to provide the appropriate raw data for conversion to renderable data, such as separating lighting from appearance data in order to generate raw texture information that can be converted to texture maps. The challenge is providing the appropriate methods for conversion and inversion which can be performed on a programmable camera platform in a reasonable amount of time for previs purposes.
- ***Eliminate Noise in Captured Data:*** Measurement noise and ambiguity occurs in all capture techniques to varying degrees. The challenge is, once a capture method is chosen, what methods for noise reduction is appropriate for use within *on-set previs* framework?

- ***Investigate Sampling Rates to Balance Speed and Quantity of Captured Data:*** Reducing the number of spatial and temporal samples when capturing scene data can speed up acquisition but can lead to sparse data and visibly inconsistent reconstruction. Too many samples can produce high-quality data but at the expense of speed of acquisition and storage costs. The challenge is choosing a capture technique that is fast but provides a good trade off with respect to speed and high-quality scene data. For previs, immediate feedback is imperative and quality of render to a lesser degree.
- ***Devise Optimal Scene Representation :*** Once scene data is captured, the reconstructed geometry of the scene can be represented in many different ways, such as point data, triangle meshes, or image-based data. Each choice of representation can impact the visualization technique and reconstruction time. The challenge is in understanding the trade offs with each representation and choosing one that is most appropriate for the *on-set previs* framework (e.g., speed, accuracy, interactivity, etc.).
- ***Visualize Reconstructed Scenes:*** Techniques for rendering the scene data for Previsualization are intimately tied to the representation of the reconstructed scene. Although, surveys have been done [SCD⁺06] regarding various data reconstruction methods, there is no standard framework for rendering reconstructed scene data. The challenge is choosing an appropriate rendering technique that is compatible with the reconstruction of the captured data and achieves the desired results for interactive and 3D previsualization.
- ***Flexible Platform for an Ever Changing Workflow:*** Developing a previs method that can satisfy all types of artistic visions and previewing desires would be an intractable task. Because we cannot know what every director or

cinematographer may want or need, anticipating these specific needs is futile. Therefore, our challenge is in designing a previs interface for *on-set previs* and programmable camera that is flexible enough to be customized to fit the needs of most users. Also, because we assume that in the future more tasks will be incorporated into the production phase of filmmaking; another challenge is developing an architecture that scales with the increasing demand of processing power for each additional task.

In summary, the capture, conversion, representation, and visualization techniques employed in this dissertation must take into account the end goal of interactive previs. To do this, proper capture techniques must be chosen which balance error rates, acquisition time, and acquisition data size. Additionally, in order to provide interactive visualization of the scene data, the appropriate scene representation, sampling, and reconstruction techniques must also be employed in the framework. Even though preventive measures will be taken to minimize error, the primary goal of this *on-set previs* project is to approximate the scene to aid in planning and previewing how the shot would look. Therefore, the resulting data capture techniques may result in higher-error rates due to environmental and object properties such as excess ambient lighting, occlusions, and specular surfaces. Even though many of the capture techniques that may be used for previs have been extensively studied [DWT⁺02, WGT⁺05, HS98, FLW02, Deb98a, SNB07, GH00, HPB06a, OKP⁺08, RKKS⁺07, PTMD07], there is no standard reconstruction and rendering technique. This is especially true when considering the use of graphics hardware for accelerating the reconstruction and visualization of the captured data.

Therefore, this dissertation works toward establishing a framework for high-quality previs which includes automatic modeling of scene lighting using low-cost "off-the-shelf" camera technology, novel scene analysis algorithms and representa-

tion techniques, and interactive visualization. Our research and development of a system for capturing and interactive rendering of scenes for previs could alleviate the need for directors to wait for time consuming and complicated effects to be applied by third parties (visual effects personnel and indie filmmakers). If successful in alleviating this time gap, the benefits could result in higher-quality movies with fewer reshoots and less time spent on production because feedback to the director would be almost instantaneous. In the future, in conjunction with adding full scene capture, automatic scene modeling, and interactive rendering, this work could also lead to advances that allow for high-quality cinematic movie making to be accomplished in real-time, potentially spawning new areas of film making and rapid visual story telling. Additionally, this project could provide a path that leads to a commoditization of high-quality movie tools embedded in the camera that could be programmed to perform all the major steps of production and post-production with the same quality seen in large budget studios. This could lead to a revolution of high-quality movies made by low-budget and hobbyist movie makers thus democratizing high-quality movie making.

To achieve interactive *on-set previs* of elements normally performed in post-production, a confluence of a new programmable camera architecture, novel GPU amenable photometric scene analysis and image processing algorithms, and visualization applications must be carefully constructed to maximize throughput and provide adaptive visualization. To fully appreciate the scope of the approach of this project and the challenges of Previsualization, one must realize that Previsualization is truly an "end-to-end" technique. Raw data is sensed in the form of photons, then converted and interpreted into a representation that can be reasoned about and visualized in an intuitive way. Disciplines such as computer vision strive to reason about sensed visual data from ill-constrained and noisy input but generally lack

intuitive and aesthetic visualization. On the other hand, computer graphics tend to focus on intuitive and aesthetic visualization and is generally initialized with well-defined input, such as geometry, textures, and lighting. In *previs*, a formidable challenge lies with developing a technique that spans several disciplines including digital photography for capturing and sensing scene data, computer graphics for conversion and rendering data, and computer vision for capture and understanding scene data. The convergence of these three fields has been described as Computational Photography¹ [Fer06]. This dissertation views *Interactive Previsualization* as being within the realm of Computational Photography, as it strives to sense and reason about data such as in Computer Vision as well as provide intuitive and aesthetic visualization of data as in Computer Graphics.

1.3 Thesis Statement

The goal of this dissertation is to improve the capabilities, realism, accuracy, speed, and cost of *on-set previs* for visualizing movie sets. We developed a framework composed of three main functions, capture, analysis, and visualization as illustrated in Figure 1.8. Each of these stages will be embedded in a single platform, a programmable camera back-end, which we call PCam [LAW09]. PCam not only alleviates the current gap between capture and processing within today’s cameras but also provides a novel visual programming interface to facilitate the implementation of capture, analysis, and visualization algorithms. Much academic work and many algorithms exist that can capture parts of a movie set and convert the raw data to 3D digital geometry, texture, and reflectance information (as described in Section

¹Alternatively Computational Photography has been defined as any technique that tries to record a richer visual experience through the use of plentiful computing, digital sensors, modern optics, actuators, probes and smart lights [RTM⁺06].

1.2), with the exception of on-set lighting. Therefore, this dissertation focused its efforts on developing a novel photometric capture, analysis, and processing of physical on-set lighting with the goal of allowing for virtual manipulation of that lighting. We previously introduced this novel light capture method as *Symmetric Lighting*. The work done in this dissertation provides the director the ability to virtually edit the color, intensity, and locations of the physical lights. An ancillary benefit that is a consequence of light manipulation, the director now can identify and manipulate the shadows, provide perform white balance of multiple illuminants, and identify the color of lighting, all of which has not previously been possible in this fashion. Lastly, in this dissertation project we developed an interface specifically designed to integrate all of the aforementioned capabilities into a single application, thus providing a unique visualization tool that can be used for *on-set previs* of movie sets.

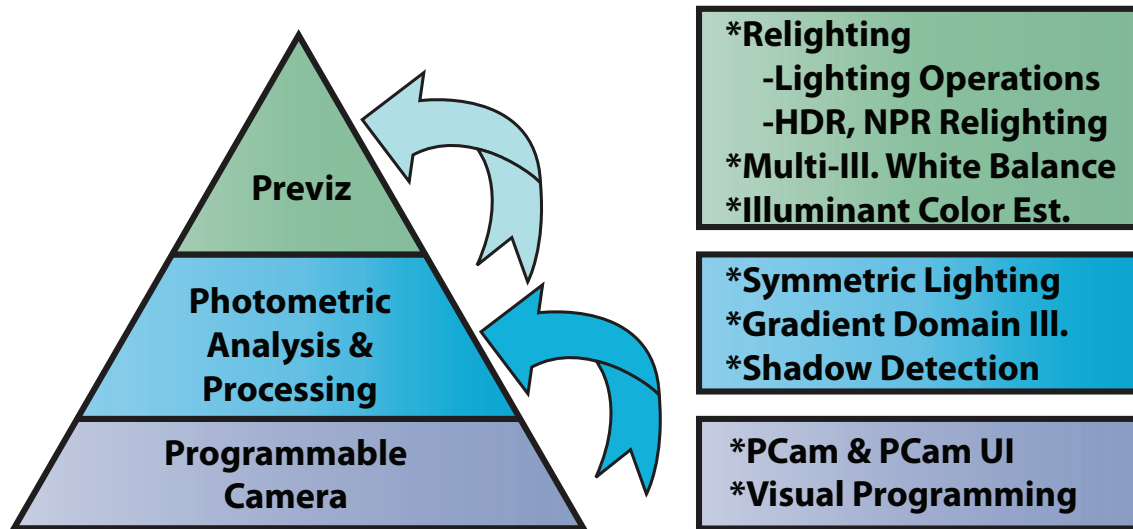


Figure 1.8: This image provides an illustration of the relationship of the developed Previsualization framework components. The data is then sampled and converted using a novel programmable camera. The camera input can be photometrically analyzed and manipulated on camera for interactive feedback. The converted representation can then be visualized and further manipulated on-camera.

Although the PCam framework does not eliminate the need for hand drawn, artist centric, storyboarding techniques as a conceptual medium, it will eliminate the need for tedious measurement and reconstruction of lighting environments and allow directors to see the implications of their cinematic choices instantly. Also, the on-set lighting captured via *Symmetric Lighting* could serve as input to other forms of previs as well as used in post-production for relighting and visual effects. This could eliminate the need for manual re-creation lighting in other areas of production and other computer generated effects. In addition to advancing the commercial field of Previsualization, this work makes several novel contributions to the academic communities of Computational Photography, Computer Vision, and Computer Graphics. This work could also have an impact in the commercial movie making field as well , with 50 to 75 percent of the on-set time being spent being spent dealing with and adjusting lighting². With on-set time being the most expensive part of the movie making process on an hourly-basis, we believe the work in this dissertation can prove increase the value of previs by reducing the cost associated with on-set lighting adjustment. For example, Spiderman 2's on-set production cost was 45 million dollars. Then if we assume that a reduction in on-set time of only ten percent could result in a savings of 4.5 million dollars, on-set lighting previs would only need to reduce the time adjusting lighting to no more than 65 percent of the time.

1.4 Dissertation Contributions

The primary academic contributions of this dissertation include :

1. PCam: A Programmable Camera Architecture and User Interface

²Refer to the previous quote by Paul Debevec as well as our interview of industry professionals in Section 6.5

(Chapters 3 & 4)

- **Novel Programmable Camera back-end :** The architecture of modern digital cameras can generally be decomposed into three main stages: image capture, image adjustment, and image encoding and storage. While the technology that implements each camera stage has evolved since its invention, the core design of digital cameras has not changed and today remains unprogrammable. To address this issue, a programmable camera backend was developed, which allows users to redefine, re-order, and modify arbitrary camera pipelines for previs.
- **Novel Visual Programming Interface :** Current programmable cameras only have text-based programming interfaces[ATP⁺10], which provide an Application Programmer Interface (API) which not only requires a user to understand but also be proficient in a high-level programming language thus making programming current programmable cameras difficult. To mitigate this problem, a novel visual programming interface for programmable cameras was developed during this work, which can be utilized for PCam as well as other programmable camera architectures.
- **Simplified Programming Model :** Programmable camera programming has a high learning curve because of the prerequisite knowledge needed to understand the inner workings of a camera as well as being familiar with the implementations of the required image processing and computer vision algorithms. To reduce the learning curve and make programmable cameras more accessible to a wide audience we designed a simplified programming model and camera pipeline abstraction that reduces the learning curve for non-technical users.

- **Iterative Programming and Instant Feedback Capability :** Current development process for programmable cameras require development to be done somewhere other than the picture taking environment, then travel to and from the picture taking environment to take and review the image that was captured using a programmable camera. This can be time consuming as trivial changes to the camera pipeline require traveling back to the development environment. WYSIWYG feedback was built into the UI, which allows users to immediately view changes to camera pipelines as they develop or edit filters, shortening the development cycle of previous Computational Photography development approaches.

2. Photometric Analysis & Symmetric Lighting (Chapter 5)

- **Novel Light Capture Technique :** We developed a new technique for capturing the combined influence each light on a particular pixel within an image, which we call *Symmetric Lighting*. This technique is a novel light capture method that estimates through global optimization, the mixture of multiple light sources at each pixel and store this information in an illumination map, which we refer to as the Beta (β) map. The β map accounts for each light's contribution separately, allowing for the manipulation of each light source independently of each other.
- **Novel Light Editing Method :** We constructed gradient domain operations manipulate function using first order statistics (i.e., derivatives) in order to affect groups of pixels in a meaningful way. It has been shown that the Human Visual System (HVS) is more sensitive to changes in local contrasts as opposed to absolute levels [RBH08]. It is also been shown that image gradients correlate to differences in contrast [BZCC10]. This

motivates the use of gradient domain operations for manipulating the illumination map. A novel gradient based illumination operation was developed to untangle the influence of geometry from the illumination map. This method uses an edge suppression technique in the gradient domain which is then reconstructed using a Poisson solver.

- **Novel Shadow Detection Method** : Image-based shadow identification can be a challenging task especially when the objects casting shadows are stationary. Many academic papers and algorithms have been published based on different statistical features, intensity, chromaticity, physical, or other assumptions about the scene. We developed a novel method based on Symmetric Lighting in conjunction with a threshold watershed algorithm that can identify the shadows within an image. Our detection method includes the identification of umbra and penumbra regions within the image, making editing of soft shadows possible. Additionally, our method allows for identification of shadows corresponding to specific lights within the scene, which to our knowledge is not currently feasible with other methods. Unlike other methods, we make no assumptions about the scene properties such as, geometry, texture, or spatial or temporal clues that may exist within a scene.

3. Visualization & Previsualization (Chapter 6)

- **Novel Real-Time Relighting Method** : Capturing complex lighting environments for use in rendering applications require sophisticated setup, travel to location, and dense sampling of the lighting environment. We developed a new real-time relighting method that can account for multiple illumination sources including complex, spatially-varying light-

ing environments. Beyond typical relighting, our application allows for manipulation of the light distribution and location of the lights. Also, we developed a novel color editing capability for each individual lights as well as a log-based range scaling method, which can produce High Dynamic Range images from low dynamic range lighting data.

- **Novel Multi-illuminant White Balance :** Most current research in white balance methods make the simplifying assumption that the entire photographed scene is illuminated by a single light source. However, most real-life scenes contain multiple lights and the illumination recorded at each camera pixel is due to the combined influence of these light sources. A fully automated multi-light white balance method was developed that requires no user interaction and determines color constancy in low light and low color images.
- **Novel Light Color Estimation Method :** The most common way to measure the color of light is to use a device called a Spectrophotometer which uses diffraction grating to analyze the spectrum of the light. These devices can be very expensive and generally require a laboratory setting to be used effectively. We developed a light color estimation method that utilizes the β map from our *Symmetric Lighting* method to estimate the color of a physical light from camera images without the need for separate and expensive hardware-based estimation method such as a spectrophotometer.

Although not a direct contribution but more of an over-arching theme, this dissertation leverages graphics hardware (GPUs) to speed up the capture, processing, and visualization of data captured by computer vision techniques. To my knowl-

edge, this is the first full end to end light capture for *on-set previs* implemented on graphics hardware. Using graphics hardware in this fashion should help to bridge the gap between the vision and graphics communities by providing software, algorithms, and evaluations for techniques that utilize graphics hardware for solving vision problems and the visualization vision data.

1.5 Outline of Dissertation

The remainder of this dissertation is organized as follows: Chapter 2 discusses research in the related areas of this project and provides background knowledge to the work done; Chapter 3 describes the programmable camera architecture and Chapter 4 outlines the visual programming user interface. Chapter 5 outlines the photometric analysis and processing that serves as the analytical foundation for Symmetric Lighting and defines a novel data structure for storing lighting contributions per pixel of each light, called the β map. Chapter 6 describes the visualization and Previsualization applications that are now possible as a result of the programmable camera architecture and the photometric analysis and processing algorithms as well as the quantitative assessments of all the contributions made in this dissertation. Chapter 7 outlines work that this project can incorporate in the future. Finally, Chapter 8 provides the final conclusions followed by the Appendix, which includes definitions, selected programming code, and additional renderings.

Chapter 2

Related Work

2.1 Programmable Imaging

2.1.1 Programmable Camera Components

Shree Nayar has envisioned the concept of a programmable imaging system [Nay06] as one that controls the properties of the optical components that capture light, while dynamically changing the software to process the variations in input due to changes in these properties. Although recent research in Computational Photography has demonstrated that it is feasible and useful to have various optical components be programmable, little progress has been made to develop an architecture for dynamically adapting the software that processes and reconstructs the captured data. These programmable components strive to extract information from light fields that current camera components cannot. Multi-flash[RTF⁺05], flash/no-flash pairs[PAH⁺04] and projector-based active illumination[ZN06a, MNBN07] techniques have been used to extract depth information, denoise images, separate local and global reflection components and in general produce higher quality images from a scene. Coded [LFDF07] and programmable [LLW⁺08, VRA⁺07, ZN06b] apertures have been used to ex-

tract depth information from a scene, capture light fields, and provide information for refocusing images. Programmable and coded lenses in the form of wavefront coding[ERDC95] and programmable Digital Micro-Mirror Arrays[NBB04, NB03] have been used to extend the camera’s depth of field, increase the dynamic range, and facilitate object recognition. Shutter programmability[RAT06] has been shown to reduce or alleviate information loss from motion blur in order to reconstruct an unblurred image. Programming the imaging sensor involves modifying sensor parameters, such as exposure time and pixel demosaicing, to capture incoming light resulting in increased dynamic range of the sensor. Nayar et al.[NB03] developed a technique to dynamically adapt the exposure time for individual pixels on a sensor instead of using a predetermined exposure time for the whole sensor. Johansson et al.[JLMM05] developed a CMOS image sensor with programmable image processing capabilities integrated into the imager.

Although these advancements provide some level of programmability on the front end of the camera, these techniques still require external processing to decode and render the captured information in order to generate the final image. Therefore the fundamental limitation to developing a fully programmable camera lies in the fact that current camera architectures cannot process the results of these techniques on-camera. This limitation causes current camera architectures and the new programmable components to fall short of the idea of a programmable imaging system envisioned by Nayar[Nay06]. Therefore, the P-Cam architecture could be used to fulfill the need for programmable on-camera processing to fill the gap in current research toward a fully programmable and repurposable camera.

2.1.2 Scriptable Cameras

For some time now, camera manufacturers have made digital still cameras (DSC) with scripting capabilities such as the Kodak DCS260 (with Flashpoint's Digita OS), the Minolta Dimage 1500 and the more recent Canon cameras with Digic II and III processors. Scripting within cameras gives the camera user the ability to trigger existing camera functionality by automating a sequence of steps, normally performed by the camera user themselves, through the use of small programs called scripts. For example, a firmware "enhancement" called CHDK (Cannon Hacker's Development Kit) lets Cannon Powershot users write camera scripts that mostly emulate button clicks or menu selections the user would perform while using the camera. Unfortunately, the scripting capabilities of these cameras are limited to the existing functionality within the camera and add no image processing capabilities. Also, emerging Computational Photography techniques such as refocusing, geometry reconstruction, motion deblurring, or automatically generating High Dynamic Range Imagery. Unlike the P-Cam architecture, scriptable cameras do not support arbitrary image processing capabilities, therefore cameras exhibiting only scripting like programmability cannot be repurposed for tasks other than accurate copying of light intensities. In essence, scriptable cameras only exhibit a small subset of functionality provided by P-Cam.

2.1.3 Smart Cameras & Other Camera-Computer Combinations

Smart Cameras (a.k.a Intelligent Cameras) are self-contained cameras with dedicated embedded computer systems. Smart cameras come in a wide range of configurations and target a particular market, usually industrial or robotic vision. These

cameras can be programmed to meet the needs of the camera user and have tightly coupled image processing systems based on software and DSP hardware. In addition to the image processing capabilities, Smart Cameras can have a wide range of additional capabilities, such as communication interfaces, external memory, self-illumination, and movement controls. Smart Cameras are currently based on a slower DSP and CPU architectures instead of the significantly more efficient stream-based design [CLL05] we are proposing to use with P-Cam. Another difference between P-Cam and Smart Cameras are that Smart Cameras are targeted for specific vertical applications and are not reconfigurable on-the-fly as in P-Cam. For example, most industrial assembly lines use some form of Smart Camera for product inspection. These, like most Smart Cameras, cannot be reconfigured without reprogramming of the camera's software and possibly hardware. So repurposing an assembly line Smart Camera to be a "Point & Shoot" style camera on-the-fly is not feasible. Smart Cameras are built with application specific design requirements (vertical design). Whereas the scalability and flexibility of P-Cam allows it to be used for a wide range of camera applications spanning camera phones, to "Point & Shoot" Cameras, to professional and cinematic cameras (horizontal design). Scalability for Smart Cameras is hard to gauge due to the plethora of different hardware and software components used to build the wide variety of Smart Cameras. In addition to scalability and flexibility issues, Smart Cameras are generally cost prohibitive for consumer level cameras, whereas with current projections for Stream Processors being produced for around \$16 per chip making it a cost effective replacement for current embedded processors. Also, most Smart Cameras have limited image resolution, usually ranging from VGA to XGA standards since they are used mostly for industrial applications. P-Cam's architecture does not put a fundamental limitation on the hardware used for capturing images.

2.1.4 Programmable Cameras

Because the area of programmable cameras is such a new field, there are few works to cite in this area. The two main frameworks for providing programmable camera pipelines is the work done by Adams et al. called the FrankenCamera [ATP⁺10]. The FrankenCamera provides an API and a general paradigm for programming camera pipelines. This API allows the camera user to write programs that control the camera and then upload them to the camera to replace the existing pipeline. The FrankenCamera uses pure C programming and does not provide a graphical camera UI to the user. Since only one pipeline is loaded at a time, the current camera pipeline is usually replaced. Because the process of installing a new pipeline is controlled by a development computer no additional UI elements are needed and the existing camera UI can be utilized. Development is not done on the camera, so the user is required to transport the camera to and from the development environment in order to make changes to the existing camera pipeline or create another one. This differs from the approach described here in that it provides features within the UI to create, modify, and test new and existing programmable camera pipelines.

Chapter 3

PCam: A Programmable Camera Architecture

3.1 Overview

The architecture of modern digital cameras can be decomposed into three main stages: 1) Image Capture, 2) Image Adjustment and Encoding, and 3) Storage. While the technology in each camera stage has evolved, the core design of digital cameras has not changed. Recent Computational Photography research has demonstrated how augmented camera components can extract and encode scene data such as depth, refocusing information, and higher dynamic range [RTM⁺06] in ways not possible with traditional camera components, by making traditional camera components such as the flash, lenses, apertures, shutters, exposure, and sensors programmable. Shree Nayar has further envisioned an entire *programmable imaging system* [Nay06] as one that controls individual programmable camera components, while dynamically manipulating and processing camera input using the system's software. Unfortunately Nayar's vision has not been realized because **full**

programmability beyond the image sensor (programmable backend) is still not possible with current camera architectures thus limiting their capabilities. Consequently, many advanced computational photography algorithms are implemented by transferring image data generated by programmable and coded components to a host computer for off-camera processing in order to render their desired effect.

To address this issue, we describe a programmable camera backend (P-Cam) that achieves Nayar’s vision and can be used to implement many state-of-the-art computational photography algorithms on-camera. As concrete examples, we have implemented several image processing, stylized rendering, and matting algorithms on-camera. High Dynamic Range techniques, tone mapping, color filters, and object recognition are also discussed as future work. These implemented algorithms and the proposed future work are examples of algorithms that currently have to be run off-camera as a post-process using current cameras.

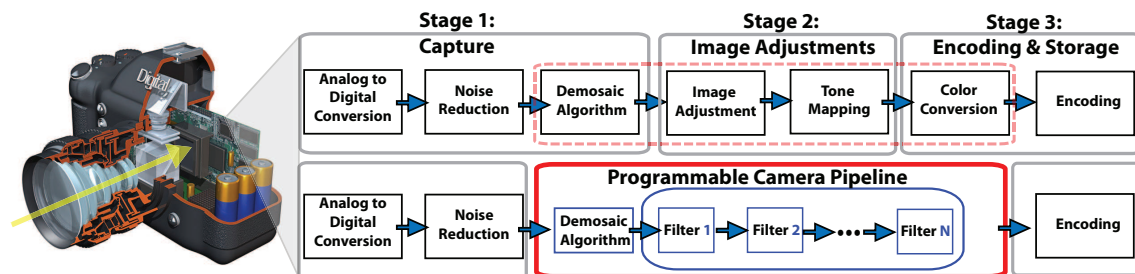


Figure 3.1: Comparison between traditional camera pipelines (top) and current programmable camera pipeline architectures (bottom).

PCam allows users to redefine, re-order, and modify the camera pipeline. User programs (or functions), which we call Camera Shaders, are executed on-camera during image capture. Since PCam can be reprogrammed for different specific tasks, a single camera can mimic the capabilities multiple specialized cameras. To facilitate the enhanced processing requirements of camera shaders and high pixel resolutions, PCam uses a stream processor-based design which can perform billions of operations

per second in conjunction with tile-based image partitioning to increase processing efficiency. This architecture overcomes the limitations of scalar processors and imaging Digital Signal Processors (DSPs) currently used in camera architectures, while providing user programmability of the camera components.

The rest of the section is organized into three sections; Section 3.2 describes the PCam architecture, Section 3.3 describes the computation model for PCam, and section 3.4 discusses the implementations.

3.2 Tile-based Streaming Architecture

3.2.1 Overview

The architecture we now describe shall be the basis for a new generation of programmable digital camera architectures we call P-Cam. P-Cam allows for arbitrary shader programs to manipulate video and image frames on-camera during image capture and can be integrated into future digital camera designs. P-Cam’s architecture applies existing stream processor technologies and tile-based ideas in a new domain, namely to create a programmable digital camera. P-Cam is flexible and designed to work with a wide range of camera configurations while maintaining efficiency and scalability.

The rest of this section describes the three main features of the architecture, the stream processor, tile-based data structure, and supporting features. Although low-level architectural and implementation details are presented in later sections, we emphasize that our main contribution is in proposing a novel camera architecture that shall eventually be fabricated as camera hardware. In this paper, we describe our architecture and develop a simple prototype that sufficiently demonstrates our idea’s feasibility. In section 3.4.1 we shall describe potential implementations of

P-Cam as well as our prototype. In the following subsections, we describe and justify the use of tiles as a core data structure, the use of programmable stream-based processors, as well as supporting architectural features such as texture decompression, out-of-core texture fetching, datatype with user-defined precision, and high quality anti-aliasing. Since our design has not been fabricated in its final hardware form, we are currently unable to provide accurate performance measurements. For instance, the entire camera shall be contained on a single circuit board with high bandwidth data paths between integrated circuit, which is orders of magnitude faster than external cabling we use in our prototype. Throughout this paper, we try to give adequate architectural details that make our vision concrete, but we are also careful to avoid unnecessarily committing to a specific components and processors or restricting the architecture to a particular camera configuration.

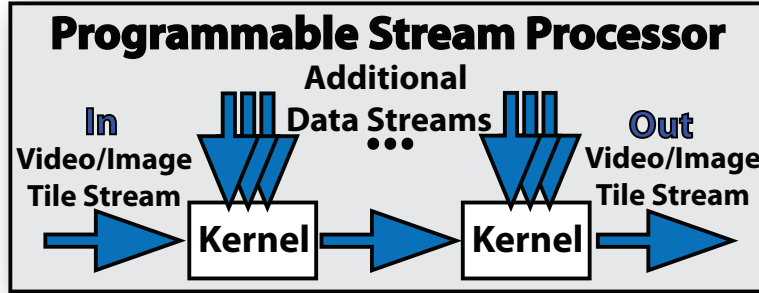


Figure 3.2: An Illustration of an abstract stream processing model. For our architecture, the main data stream will be the image or video stream transferred from the image sensor. The power of stream processor comes from the ability to define a kernel operation that will operate on an entire stream.

3.2.2 Stream Processor

Multimedia processing currently requires tremendous amounts of processing power, on the order of 10-100 billion operations per second to achieve real-time perfor-

mance [Rix01, KRD⁺03]. To keep pace with evolving trends and user needs, programmability has become a standard feature on emerging processors. For these reasons, we chose to center our architecture around programmable multimedia stream processors. Since programmable GPUs have converged to a stream-based design, in many ways, P-Cam’s stream processor is architecturally similar to GPUs used on mobile devices. However, we do not tie P-Cam’s processor to a specific processor. We assume a stream processor with at least the capabilities of the Imagine stream processor [KDR⁺02]. Since digital cameras are battery-powered portable devices, the amount of power and heat they dissipate is a concern. We expect that in implementing P-Cam, state of the art power efficiency techniques such as low-power circuit design and voltage scaling shall be used. However, a full description of these low-power techniques are implementation details that are beyond the scope of this paper. Moreover, compared to mobile GPUs that run applications such as mobile gaming or video processing with sustained high workloads, taking pictures only takes milliseconds.

To increase efficiency, stream processors capitalize on *data, instruction and task parallelism and locality*. Data is mapped to streams and when possible, tasks are executed as independent kernels. Imaging and multimedia operations tend to be sequential chains of operations that are repeated on groups of pixels. Thus, they inherently are amenable to high levels of data and instruction-level parallelization [KRD⁺03]. Therefore, organizing applications into data streams and kernels prepares the application for stream processing and to take advantage of the homogeneity of multi-media applications. The P-Cam architecture feeds images as tiles into the stream processor that has been programmed with user-specified algorithms using Camera Shaders as kernels.

The stream processor in P-Cam’s architecture in Figure 3.3 is now described.

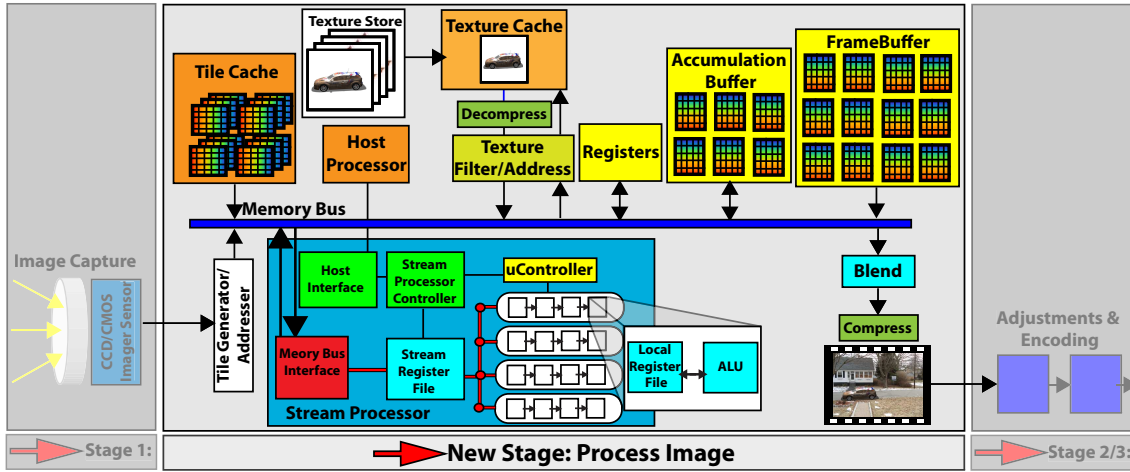


Figure 3.3: The PCam architecture. This is detailed view of the new camera stage shown in Figure 3.1. Image data from an image/video source are converted into tiles. The tiles are then cached and streamed through a series of Camera Shaders (depicted as rows of white boxes within the stream processor). Camera Shaders allow user programmability within the architecture by mapping code to kernels. Stream processor can access additional streams from either texture memory, kernel output, or data registers.

For efficiency, the processor features a memory hierarchy. At the highest level of the memory hierarchy is a global memory bank (SDRAM), where data is stored before and after processing. The secondary level of memory, the Stream Register File (SRF), stores stream data on the stream processor for ALUs to read input, share data and makes it possible one ALU's output can be chained to another ALU's input. At the lowest level, each ALU contains a Local Register File (LRF), where data and instructions used by a single ALU are stored during operations. The memory hierarchy exploits fast on-chip bandwidth between the LRF and SRF and stores data on-chip only when necessary. The result is an efficient processor that provides a speed-up factor of up to 20-times over scalar processors [KDR⁺02]. The tile cache acts as a data stream queue for the stream processor and improves performance when tiles are re-used; groups of tiles are removed in parallel as they are needed by the stream processor (see Figure 3.3) and transferred to the SRF.

This transfer is done in bulk to maximize efficiency, utilize the full bandwidth of the memory lines, and aids in latency hiding. From the SRF, tiles are distributed to the kernels and stored in the LRFs to be operated on. The results of the operations are returned to the SRF to be redistributed to other kernels or propagated back out to global memory when processing is complete. Because the stream processor can have many kernels and each kernel applies the same operation to each tile, many tiles can be processed simultaneously. This parallelism and locality are the core of what makes the stream processor efficient.

The purpose of the stream processor is to apply the programmer-defined tasks to the stream of image tiles and additional data streams as illustrated in Figure 3.2. The tasks, which we call Camera Shaders, are written in a high-level language and translated into stream and kernel instructions, which are stored in the Application Processor (see Figure 3.3). Similar to the Imagine stream processor, stream-level instructions are responsible for routing and scheduling streams and kernel-level instructions operate on one or more data streams. The Application Processor issues routing commands for routing stream to and from the SRF as well to and from various LRF clusters. The kernel instructions are also issued from the Application processor to clusters of LRFs. Once the video stream has all kernel operations applied, the tiles are routed to an accumulation buffer and subsequently copied to the framebuffer to be routed to the rest of the camera pipeline.

3.2.3 Tile Based Data Structure

As mentioned in the previous subsection, P-Cam operates on images divided into tiles. There are several reasons we can justify partitioning images into tile. First, the streams of data fed into the stream processor must consist of small records in order to efficiently fill and fit the two types of memory used in stream processors

(SRF & LRF). Second, comparing tiled images to non-partioned images, processing tiles reduces the number of data lanes needed to transfer data, since bandwidth is then proportional to the tile size not the image. Additionally, it has been shown [AMH02] that memory requirements for intermediate storage of data is also proportional to the tile size. By using tiles we can reduce the memory requirements by over 33% compared to non-partioned images. Third, tile-based partitioning of the image ensures that the architecture scales appropriately even when high resolution images are used. For example, high-end camera devices with large image capture capabilities (beyond the 4k video standard), would use tiles sizes matched to stream processor not the input image. Therefore, this configuration might use two stream processors in parallel to process the tiles with minimal change to the architecture. Non-partioned image frames implementations would not scale as well and would be combersome. Next we will describe how tiles are generated and how they propagate through P-Cam.

As outlined in Figure 3.1, the standard digital image capture process must prepare image data before it can be used as input to P-Cam. At the start of the rendering process, an image frame generated by the image sensor collects light signals that are converted to a digital representation resembling a square matrix of RGB pixels called a Bayer pattern (also called a Color Filter Array) [Bay76]. This pattern can be configured many ways but we shall assume that a standard pattern consists of twice as many green pixels as red or blue. To produce an array of RGB pixels, the pattern must be demosaiced or interpolated to obtain the appropriate RGB pixel values, which we consider as the raw image data. The raw image data is then transfered to a memory location outside the imager so that subsequent images can be processed. The image data is then divided into tiles by the tile generator, stored in a cache, then streamed as input to the stream processor. In addition to generating

tiles, The Tile Generator is also responsible for ordering tiles, storing tiles in the Tile Cache, and generating texture coordinates for each tile which are all part of the tile data. The texture coordinates are passed to the stream processor's application processor to prepare additional data streams as input to stream processor. Ordering is used by the stream processor to reassemble the tiles into a coherent frame at the end of processing.

The Tile Cache is used as an intermediate tile storage after they are generated by the Tile Generator and before they are requested by the stream processor. As the Tile Cache fills with tiles, the stream of the tiles can be requested by the stream processor's application processor. As seen in Figure 3.3 the Tile Cache is interfaced indirectly by the SRF via the D-RAM interface (facilitates communication between on-chip memory and off-chip memory), which transfers tiles to and from the SRF. As tiles are streamed, they are transferred to the SRF before being distributed to individual LRFs. The size of the tile, the on-chip storage capabilities of the SRF, and the number of ALU/LRF pairs dictate the number of tiles that can be processed simultaneously. The goal is to use tile sizes that provide maximum data locality and allow for kernel and stream scheduling to fully utilize the high-bandwidth between the SRF and LRF clusters. This balance is dictated by the particular configuration of the stream processor used to implement P-Cam and is not dictated by the architecture itself. The performance is maximized when the SRF is consistently filled to capacity with tiles, and just before tiles begin spilling back into global memory [Owe02]. Tile size and number of tiles are dictated by the stream processor and camera configuration and are known ahead of time. Therefore, the system can be tuned to this specific configuration. This is not true when rendering 3D scenes on GPUs such as in an OpenGL pipeline. The batch size is dictated by the scene size (how many vertices in a scene) therefore batching in this context requires dynamic

sizing. For our case, the frame size and capture rate is known ahead of time, and therefore the throughput is well-known and the tile size can be static and thus the performance is predictable.

3.2.4 Supporting Architectural Features

In addition to the previously mentioned architectural features, there are two supporting features which are important to the performance, flexibility, and precision of P-Cam. These features are texturing and high-quality anti-aliasing.

3.2.4.1 Texturing

In P-Cam, texture coordinates can be generated in two ways, by the Tile Generator or dynamically within a kernel. Dynamic coordinate generation requires that texture coordinates be generated as an output stream from a kernel, since stream processors only output streams. The stream of texture coordinates can be used by the stream processor to generate and schedule a subsequent stream of texture tiles. To make tile storage efficient in P-Cam, texture files are compressed in memory and decompressed during the fetching process. In addition to compressed textures, P-Cam also offers out-of-core texture fetching, which can fetch textures from secondary memory outside of main memory.

Compression: P-Cam uses texture compression to reduce storage requirements for textures and LUTs. As, previously mentioned, cinematic-quality rendering requires many textures. In order to accommodate a large number of textures, the architecture will require ample global memory and an efficient lossless decompression technique implemented in hardware. As with other features of P-Cam, the implementation will dictate the exact specifics and technique used, but we assume at a minimum that the decompression hardware used in P-Cam has the same features as

the iPACKMAN decompression hardware (also called the ETC or Ericsson Texture Compression) [SAM05].

Out-of-Core Texture Fetching: In conjunction with texture decompression, the need for large numbers of textures may exceed the limits of global memory within the P-Cam architecture. For such situations, we have developed a technique that allows for quick fetching and swapping of textures from locations outside global memory. Such a system would incur a time delay as a result of fetching from slower external memory. Texture compression helps mitigate this problem by producing smaller textures that require less time to load into memory, but still require the use of a latency hiding technique that allows for smart scheduling of streams of textures. While a kernel works on a texture stream, parts of that stream are used up and no longer needed. This allows new textures to replace parts of existing textures, thus when one stream ends transition to a new texture stream will be seamless.

3.2.4.2 High-Quality Anti-Aliasing

In addition to large texture requirements, high quality anti-aliasing is required for cinematic rendering. Tile-based rendering has the ability to provide four-times anti-aliasing without incurring additional computational expense. To provide higher quality anti-aliasing P-Cam offers three features that allow the user to program high-quality anti-aliasing. Texture filtering, filter functions, and an accumulation buffer that can be used for supersampling and downsampling tiles before producing the final image.

Language Comparison			
Language	Stream Processor	Graphics/Imaging	Camera API
CUDA, CTM	✓	✓	
Brook	✓		
Cg, HLSL, GLSL		✓	
Camera Shaders	✓	✓	✓

Figure 3.4: Comparison of several languages that are used for graphics/imaging and stream processing.

3.3 Camera Shader Framework

3.3.1 Overview

Historically, the term *shader* described a small program or function that performed a specialized task, usually in terms of graphics or imaging. In this spirit, we developed a framework called *Camera Shaders*, which allows developers to write shaders that manipulate incoming video or image during the camera’s capture process to influence the result of the image. Our camera programming framework has three components, 1) a language that supports stream programming, 2) functions for interfacing with the image capture hardware and attached camera, and 3) support for graphics and imaging programming. Each of these components exist in various other languages separately (see Figure 3.4), but no language or framework contains all three. For example, CUDA contains an imaging and graphics framework along with a stream programming language but lacks a camera interface. And GLSL (OpenGL Shading Language) supports graphics and imaging but doesn’t explicitly define a stream programming language. Since no framework exists that provides all three components, we describe such a framework. Thus, *Camera Shaders* enables

new functionality, through the writing of shaders, to become part of the camera system. The result is a programmable camera system that allows for a wide range of applications to be possible with a single camera hardware architecture.

As illustrated in Figure 3.5, the Camera Shader framework exposes four classes of functions; a camera interface, stream programming, math, and graphics and imaging. The stream and kernel functions allow the user to define and schedule data streams, kernels, as well as routing streams to and from kernels. For example, we may define a texture object as a data stream, then pass that stream into a kernel for processing. The result of that kernel can also be used as input to another kernel (see section 3.3.3 for more details on this example).

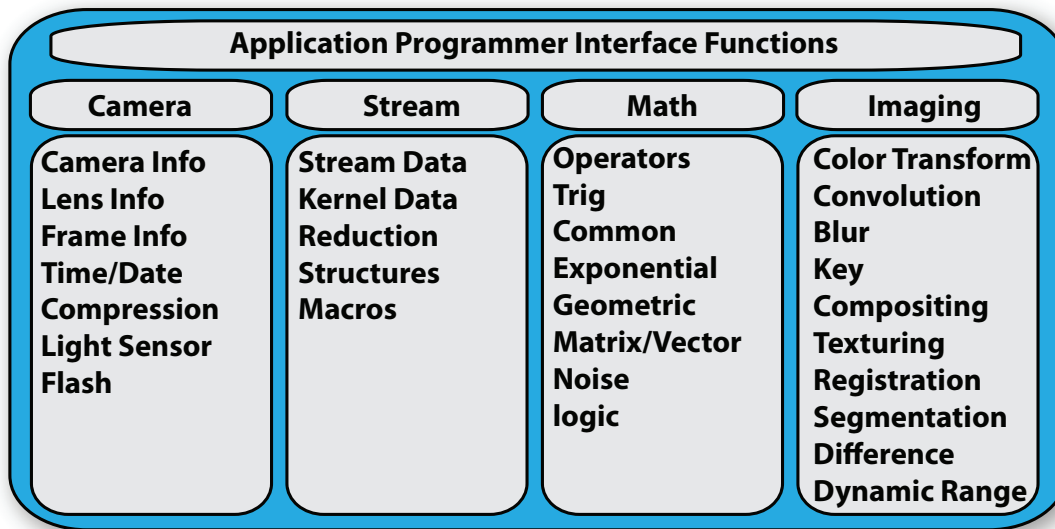


Figure 3.5: This figure outlines the groups of API functions; compositing, image processing, math, and camera functions.

3.3.2 Function Classifications

Stream Programming: As previously mentioned, stream datatypes, keywords, macros, and functions define stream programming constructs of our framework. Our frame-

work uses a modified version of the Brook [BFH⁺04] stream programming language. Streams are defined using the `<>` symbols, where a stream of 400 floats would be declared as ***float foo<400>***. A kernel function is declared using the keyword `kernel` preceding the function name, such as ***kernel void bar(float foo<400>)***.

Math Functions: The Camera Shader Framework provides a standard set of math functions that are available in many other shader languages. These functions include arithmetic types and operations, geometry functions, matrix operations, trigonometry, and miscellaneous functions.

Imaging Functions: A wide range of imaging functionality will be provided with the Camera Shader framework. The term *Imaging Operation* is used in a generic sense to mean any function that can be applied to an image. This includes compositing, image processing, color transformations, blending, and texturing.

Camera Functions: Current camera models provide users facilities for configuring some of the features within a camera such as exposure, shutter speed, and focus. In addition to configuration, cameras contain information about the environment and the camera itself such as light sensor data, timing, and battery level. Our framework provides access and manipulation capabilities to a wide range of features and data related to the camera. This data can be used within the shaders to enhance and manipulate the images during the capture process.

3.3.3 Example Shader

Figure 3.6 is a visual illustration of a technique called Chroma keying (a.k.a. blue/-green screen) and Figure 3.7 provides psuedo-code for two Camera Shaders that implement this example technique. Chroma keying replaces a solid color background with a different color from foreground objects in order to extract it and replace it with a virtual background. In the shaders, we can map these operations

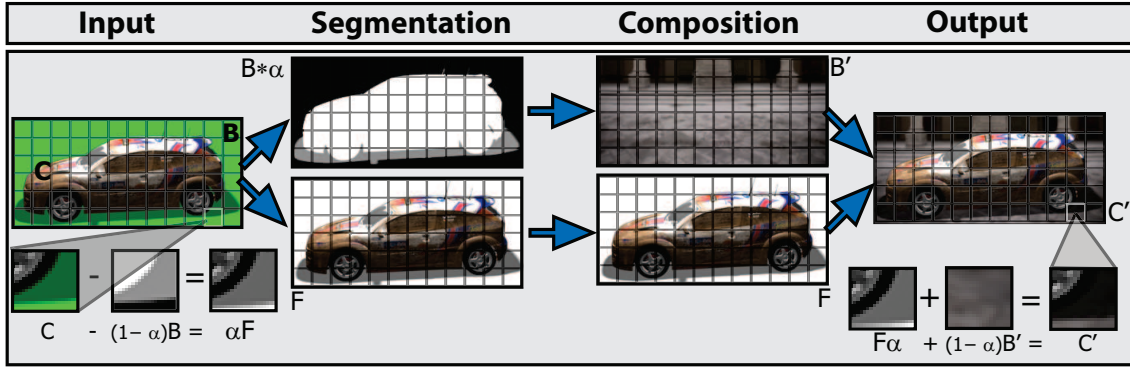


Figure 3.6: *Chroma keying*: Using color differences, foreground and background are separated and foreground is combined with a new background. The foreground (F) and the background (B) are separated from the original image (C), the 2nd pass is the composition pass where the extracted foreground F is composited with a new background B' to produce a final image C' . The simplified formulas for both passes are reversing the process of the other pass. This technique relies on alpha (α) and information at each pixel only.

to the streams and kernels in two passes. The first kernel has one input stream, the original video/image, and one output stream that is the result of masking the foreground objects to remove the background. This task is accomplished using the provided **key** function within the kernel, which separates the foreground from the background, thus making the background completely transparent. The second shader takes as input three streams, the original video/image, the mask from the previous kernel, and the new background image. In this pass, the mask is used to extract the foreground objects again from the video/image frame as well as mask the new background. The foreground objects are then composited with the new background to form the final image. The foreground object is then composited with the background stream and then copied to the out stream.

The output of this kernel could now be passed along to the next kernel without the use of intermediate buffers. The subsequent kernel could apply an additional operation on the stream such as color correction, or another effect. It should be apparent that applying several effects (previously would be considered post process)

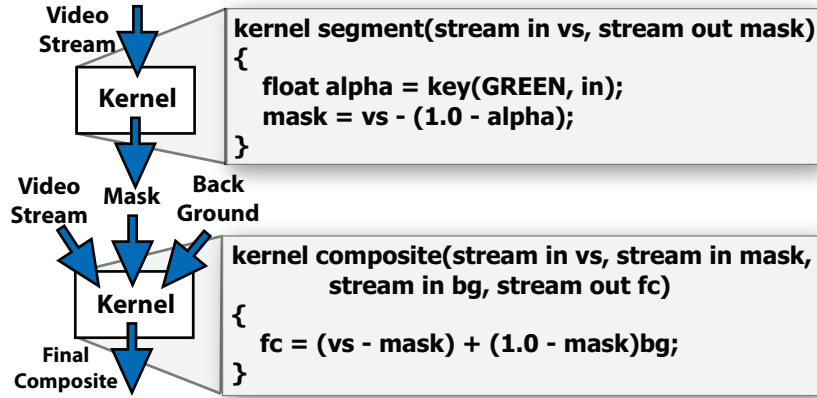


Figure 3.7: This example illustrated in Figure 3.6 outlines directing the flow of data streams from one kernel to another. The input to the composite function is a stream of tiles from the capture device and texture data stream for the virtual background. The overall effect of this operation is called chroma keying, which removes a color coded background with a virtual background.

would result in the real-time visualization of the visual effects.

3.4 Implementation

The implementation of the PCam architectures has gone through two iterations, version 1, which is the original prototype and version 2, which is the current version of PCam from which the rest of this dissertation was implemented. We provide a description of the implementation of version 1 for completeness, which is described in Section 3.4.1. The current version of PCam is subsequently described in Section 3.4.2. Additional conceptual features of PCam, which will be part of a version 3, are described in Section 7.4 as part of the Future Work chapter.

3.4.1 PCam Version 1

3.4.1.1 Overview

Version 1 of PCam consisted of two separate applications for the development and execution of camera pipelines, 1) the on-camera software that captured and processed images where the camera pipelines were used, and 2) a Workbench application for developing camera pipelines on a desktop computer. Camera pipelines were developed and tested on the Workbench application and then transferred over to the camera when ready for use on-camera. The implementation of both the Workbench application and the camera interfaces went through two phases before completion, the review phase and implementation phase. The review phase consisted of evaluating three separate UI frameworks for possible implementation of the Workbench and camera interfaces. The second phase, the implementation phase, consisted of implementing the Workbench and camera interfaces using our choice of UI framework. For this project, we chose the Qt framework [Nok09] due to availability on the target computing platform (Beagleboard [Bea09]), the feature set it provided, and its compatibility with the chosen programming language (Python) memory footprint. In the following sections we describe the review process, the implementation, and the software and hardware used to implement the Workbench and camera interfaces.

3.4.1.2 UI Server Comparison

Each choice of UI framework depends on an underlying graphics rendering environment in order to visually display their widgets. Because we are using Linux as the target operating system kernel (Angstrom as the OS), we have three choices for graphics rendering environments, the X windows system, Qtopia (Trolltech/Nokia), and using the frame buffer directly known as DirectFB. By far the most widely used

environment on all Linux and Unix kernels is the X windowing system which provides a client/server configuration for graphics rendering. Qtopia is a self-contained windowing system implementing the Qt UI framework and is targeted for embedded systems. Several UI frameworks have the capability to directly access the frame buffer, such as GTK, thereby bypassing the overhead of an intermediate hardware abstraction layer. Because each framework considered depends on an underlying graphics environment, these dependencies have to be taken into consideration when choosing. Table 3.1 provides a comparison of the criteria used for determining the best UI framework for this project.

	Api	Mem.	M.M. Wigs.	On BB	Script	UI Des.	RT
1	Qt	5MB (No X)	Yes	Yes	Yes	Yes	Yes
2	Flex (Flash)	<1MB (No X)	Yes	Yes	Yes	No	Yes
3	Gtk	12-15MB	No	Yes	Yes	Yes	Yes
4	FB (No X)	N/A	N/A	N/A	N/A	Yes	N/A

Table 3.1: A comparison of various features of four different UI frameworks we considered for implementing the camera interface.

The X Windows system (X) allows reuse of many existing UI framework for embedded UI thereby making the available number of frameworks much greater than other graphics environments. Some examples of UI framework currently running on the Beagleboard with X are Qt (not Qtopia), GTK, FLTK, Motif, and Java's AWT. Because X was designed in a client/server paradigm, the overhead associated with communication and multiple processes between the server and client can have performance implications. Additionally, X servers can be particularly useful in computing environments which require several graphical applications to run simultaneously, such as mobile internet devices (MID).

DirectFB provide a high-level abstraction of the Linux frame buffer interface. This option provides access to the underlying hardware via an abstraction but does

not provide any graphics or rendering routines. This is a popular choice for developing new UI frameworks, such as with Clutter [Pro09] for leveraging OpenGL [SGI09] for rendering UI widgets. It requires drivers for OpenGL ES 2.0, which are currently unavailable in the public domain for the Beagleboard. This can be very fast as in future releases may take full advantage of graphics hardware for acceleration.

The embedded version of the QT UI framework called Qtopia [Nok09] is capable of running directly on the frame buffer, enabling ported or new applications developed using Qt to run on an embedded system without alleviating the performance cost overhead of using X. Qtopia is an integrated graphic software stack providing the graphic environment, support libraries, and widget toolkit. Qtopia provides an optimized version (somewhat restricted) version of the Qt framework to run on embedded platforms with a memory footprint around 5MB. Qtopia and Qt on X provide two options for running Qt based applications on embedded systems. Future versions will also provide graphics acceleration via graphics hardware, thereby allowing for faster rendering and integrated graphics widgets.

3.4.1.3 UI Frameworks API Review

Before determining the UI framework that would be used to implement the Workbench and camera interface, we reviewed three potential APIs for implementing our interface. These APIs were Qt, Flex [Ado09], and Glade [Gla09] with GTK [Gno09]. Qt is a cross platform UI framework originally developed by Trolltech (now owned by Nokia) in C++ with Python bindings. Flex is an open source framework based on Adobe's Flash for developing rich Internet applications to be run within a browser that is and uses an XML variant called MXML. Glade (User Interface Designer) and GTK+ are two complimentary tools for creating cross platform UIs written in C++ but has many bindings. Initially, we started the development of the interfaces

using all three frameworks.



Figure 3.8: Three screen shots of early UI from the each of the three sampled frameworks. Left, is the Glade/GTK+ running on the Beagleboard. Center is the Adobe Flex interface running within a browser. Right is the Qt interface.

Each of the available implementation frameworks we considered were viable options. Therefore, we used a method of elimination to determine the best framework for this project by considering performance, available multi-media widgets, and if it was currently working on the Beagleboard distribution operating system (Angstrom). We conducted an informal review to determine which framework would be the least viable option for our implementation. Considering that Qt provided a mature framework supported by a large company made it an attractive choice. But one very important criterion we had to consider was how it would perform in an embedded environment. Qt is ideal for embedded environments because it offers a specific version of its runtime for embedded systems called Qt Embedded. The other two choices did not offer an embedded option, although Adobe Flash (proprietary version of Flex) has been implemented on several mobile devices but currently not the Beagleboard. Due to the lack of Flex/Flash plugins for the Beagleboard browsers, it was currently impossible to run a Flex interface on the embedded system itself and required a web browser on an external machine in order to view the camera interface, which invalidated this choice.

The programming model for Flex was considerably different than the other two

(considering the web based client/server model) choices. GTK and Qt could be developed using a single program with separate classes which is more traditional for desktop and embedded programming. Flex required a webserver that was configured to run CGI scripts (for the Beagleboard, this was Apache). In addition to providing a familiar desktop programming model, QT also provided specific classes implemented to provide a Model/View/Controller programming implementation.

Another determining factor was that the filters (shaders) implemented for this project relied heavily on the use of OpenGL for rendering. This means that the UI framework would have to have an OpenGL compatible display widget. QT and GTK had such a component, but at the time of implementation it was not clear that Flex did. Because Qt had embedded support, was mature, provided additional programming features (MVC), and was OpenGL compatible, it was chosen over the others.

3.4.1.4 Qt UI Implementation

As previously mentioned, for implementing the UI of the Workbench application and camera interface, we decided to use the Qt UI framework. From Qt version 4.2 and above, the framework provides groupings of classes specifically designed to be used within a Model/View/Controller programming paradigm. Both interfaces used this paradigm for organizing the implementation software (Figure 3.9). The View consisted of the QT UI widgets. The Controller was Python code designed to react to and instigate actions that constituted the application logic. A light-weight database was used as the Model. This separation of components has the advantage of decoupling the UI from the code, and provides a clear segregation of the application components, thus minimizing interdependency and facilitating testing, refactoring, and reuse.

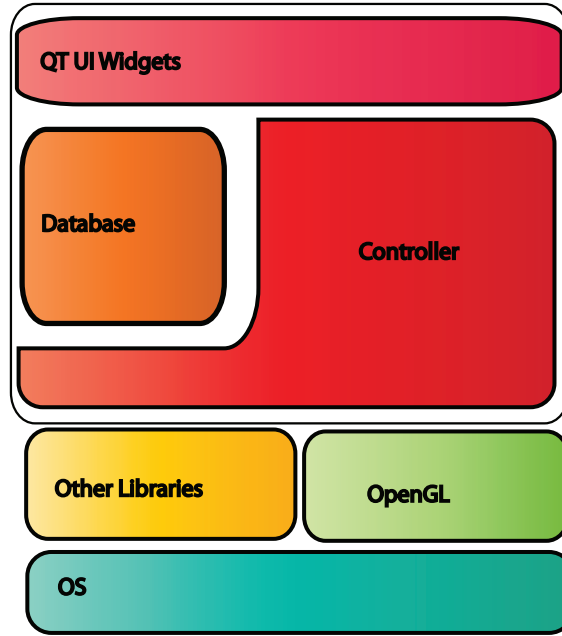


Figure 3.9: The software organization of used for both the Workbench applications the camera interfaces. The top layer represents the UI components used to interact with the user, the controller provides the functionality of both interfaces and interaction with the database. Direct access to the Opengl rendering framework was used to implement the shaders.

The Model of the application consisted of the shaders and their associated properties, which we used as filters. These properties were loaded and stored within a lightweight database provided by Qt called the `QtStandardItemModel`. Each shader was loaded and converted into a subclassed `QtStandardItem` called `PcamStandardItem`. Each item provided a tree structure to store the property values of the corresponding shader. A root item was created, and each node below the root was a shader property. Once the tree was completed, it could then be inserted into the database. The `QtStandardItemModel` provided many convenience functions for querying, listing, and updating various properties of each item.

The View of the application was implemented with two kinds of widgets, "simple" and "complex" widgets. The simple widgets, such as labels and buttons provide a basic look and feel with minimal interaction capabilities. The complex widgets

provided a way of viewing models created with the `QtStandardItemModel` database. The complex widgets constituted the View portion of the MVC paradigm used because they allowed the view of the model to be separate from the actual storage of the items. The view widgets used were the Qt provided `QTableView` and `QListView`. In addition to providing views, these widgets allowed a certain level of interaction with the model through selection and drag and drop operations. The views also used delegate classes to provide special functionality such as non-standard rendering and interaction. These delegates, called `QtStyledItemDelegates` were used to provide customized rendering of the pipeline (`QListView`), which include an image, name, and activation checkbox for each filter. The properties of each camera shader within the pipeline were displayed using the same model but a different view (`QTableView`). A customized `QTableView` was implemented to provide a hierarchal view and editing capabilities of the shader's properties. The View was a vertical table with the first column being the name of the property and the second column being the value of the property. Each property had a specific datatype (color, float, integer, enum, image, etc.) that required a specialized editor to be implemented. Another complex widget used within the interfaces was the OpenGL rendering widget. The OpenGL widget converted the current pipeline into a visual rendering using the underlying graphics hardware (GPU).

The controller part of the interfaces facilitates the interaction between the UI widgets, the databases, and other libraries. The controller takes interaction commands from the UI and performs a specific action related to editing this property, such as providing a widget to the user for editing. These actions could be the movement of shaders from the toolbox to the pipeline, editing a shader's property, or direct manipulation of the pipeline itself. The controller was implemented primarily in Python and facilitating interaction between Qt, OpenGL, and other libraries such

as OpenCV (access/control of camera).

3.4.1.5 Programming Languages & Standards

Independent of the UI framework, we used Python as the foundational programming language. Python is a high-level programming language used for general-purpose programming. It has a minimalistic syntax, a comprehensive set of libraries, and it is cross platform compatible. Because the intention is to utilize this project for the long-term interface for our programmable camera, cross-platform compatibility and bindings for each of the possible UI frameworks was an essential trait for our programming language choice. Another potential language choice was C/C++ but because of the difficulties in cross compiling and library availability on the Beagle-board, Python was the better choice of programming languages.

For representing shader assets we used the popular open digital asset standard called Collada. Collada is a file format standard used to exchange digital assets for rendering and image synthesis. For example, Collada is used to store 3D models and textures for games independent of the game engine or Digital Content Creation (DCC) tool used to manipulate them. Collada is based on XML and contains a wide variety of supported formats beyond just code for shaders, such as formatting of 3D models, textures, and scene graphs.

In addition to Collada, we used a simple XML based tree for organizing the Toolbox within the Workbench application. The XML format organized the default shaders into categories and supported meta-information as well as parameters for each shader. The Toolbox categorization and ordering is determined by an XML file, which is loaded at run time. When a particular shader within the pipeline is selected, a properties editor is activated. The properties editor allows the user to edit the specific properties defined by the shader as well as activating and naming

the particular shader instance. If the user changes the parameter of a specific shader and decides to provide this as a new default shader, the user can drag the modified shader to the Toolbox and save it. This modifies the original XML file used to configure and load the Toolbox at the start of the application.

3.4.1.6 Software Packages & Other Frameworks

The implementation of the workbench and the camera interface relied on three other software frameworks for performing rendering and image manipulation OpenGL, GLSL, and Python Image Library (PIL). OpenGL and GLSL (OpenGL shading language) provided the framework for implementing the camera shaders (i.e., filters) and PIL was used to perform image enhancements. The Workbench application and camera interface use both for providing previews of the current pipeline. In addition to providing a preview, the camera interface uses these two frameworks to render the final image.

After the captured image has been rendered, the images can be further modified within the camera interface in the "Picture Review" interface. There the user can adjust the brightness, sharpness, and contrast of the images that have been captured to further enhance the image. The implementation of the basic enhancements was done using the PIL module for python. The PIL module also provides basic conversion capabilities for images going between standard formats, such as JPG, GIF, and PNG, and the image formats used within Qt (Qimage, Qpixmap).

3.4.2 PCam Version 2

Version 2 of the PCam implementation contained the same features as version 1 but re-implemented on Android platform. Only the on-camera version was re-implemented (not the Workbench application) to take advantage of a stable de-

velopment platform provided by Google with the Android operating system (OS) and software development kit (SDK). Especially important to the implementation was the stable and working GPU drivers, which the previous version lacked. For this project we used Android version 2.2 OS and SDK code named Froyo [Goo10] for all the on-camera development. All implementation was developed using the standard development kit from Google without requiring the phone to be rooted or root access to the OS kernel.

The hardware consisted of a Motorola Droid smartphone [Mot09] which has a Texas Instruments (TI) OMAP 3430 CPU with a PowerVR SGX 530 GPU. The specific model used for this project was the A853 with 256 MB of RAM and 512 MB of flash memory with an additional 16 GB micro-SD memory. The screen consisted of an 854 x 480 pixel FWVGA display made with TFT LCD technology. The camera hardware consisted of a single rear-facing 5.0 megapixel camera with LED flash. We used a smartphone for this version due to the wide availability of the hardware. Although a smartphone was used as the platform for development, we used it solely as a camera and did not utilize any of the phone features. The PowerVR SGX 530 GPU is capable of executing pixel shader programs supporting OpenGL ES 2.0 [Gro08] and DirectX 10.1 with Shader Model 4.1. Of particular importance to this project was the compatibility with OpenGL ES 2.0, which replaces a fixed function rendering pipeline on the GPU with a fully programmable pipeline defined by the shaders. This allowed us to implement the PCam as series of low-level shaders that defined our custom OpenGL ES 2.0 rendering pipeline completely on the GPU. Camera shaders are a slightly higher-level abstraction to the low-level GPU shaders that are used to define the rendering pipeline and are only concerned with processing camera images.

The software implementation of PCam was implemented using the Eclipse [IBM01]

integrated development environment (IDE) with the java programming language. The Android SDK 2.2 provided a plugin for the Eclipse IDE to aid in development and debugging, which included an emulator for testing and debugging projects. Due to the dependence of specific hardware for this project (camera and GPU) and the inability of the emulator to emulate the camera or GPU, it was only valuable for certain simple UI development tasks. The Android Eclipse plugin also provided a debugging interface for compatible hardware that allowed the developer to run a debug version of their project directly on a smartphone attached to the development computer. To be consistent with the OpenGL ES 2.0 standard as well as provide cross-platform compatibility, we developed our GPU shaders using the GLSL shader language. Specific implementation details and GLSL code for relighting are presented in Section 6.2.6 as well as in the Appendix B. For camera pipelines, GLSL code was used to setup the pipeline rendering context in the form of a vertex and fragment shader. Using multi-pass rendering, a vertex and fragment shader were used to render a screen-aligned quad then save the quad to off-screen texture memory for the first pass. Then the Camera Shaders that made up a particular camera pipeline were then chained together in sequence, with the input of the last shader being used as texture input to the next shader. After the last Camera Shader was rendered, a final shader rendered the final image to the screen for the user to see. Additionally, each shader could reference additional parameters which could be set using the visual programming user interface described in Chapter 4.

Chapter 4

PCamUI: A Visual Programming User Interface for PCam

4.1 Overview

Computational Photography (CP) research has recently become popular in the graphics community and many promising techniques have been published. In fact, in 2009 nearly 40% of all papers submitted to the ACM SIGGRAPH conference were in the area of computational photography [Lev10]. Despite its increasing interest in the graphics research community, a majority of the proposed CP techniques are still inaccessible to non-technical users and have only been evaluated by small groups of researchers. This has led to a disconnect between the CP community and the non-technical users who might utilize their techniques. We would like to better understand how the non-technical user might want or need these new techniques. For example, do photographers use bilateral filters in their photography and if so, how? We will begin to better answer this and other related questions only after the barriers keeping non-technical users from utilizing these techniques have been

lowered. We attribute the lack of dissemination of these techniques to the high-level

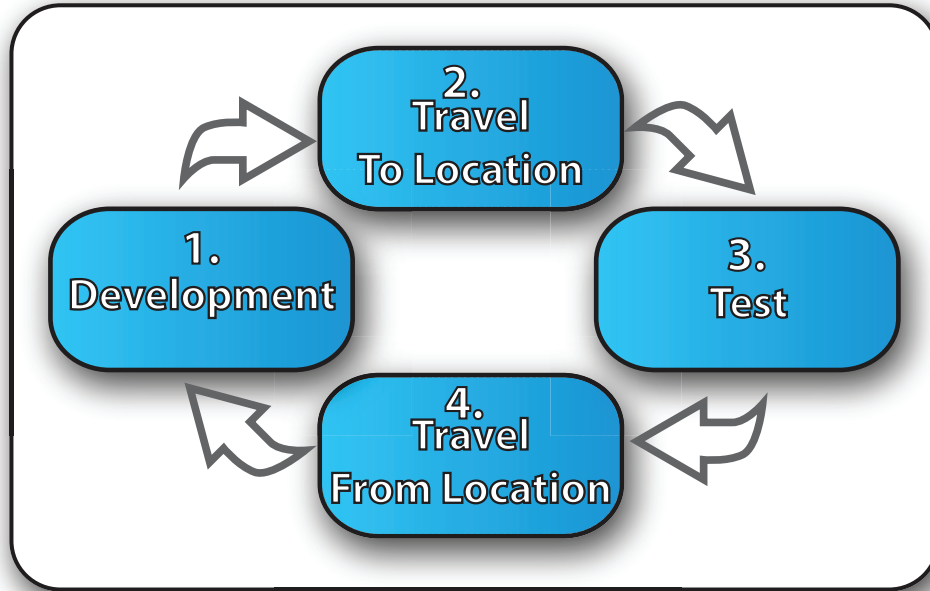


Figure 4.1: Current development process for programmable cameras using the API approach. Development is done somewhere other than the target picture taking environment, then the programmable camera needs to be transported, tested, and transported back for adjust to the algorithm. This process is repeated until the user is satisfied with the algorithm.

of prerequisite knowledge required to implement these techniques and to several shortcomings in the typical development cycle for implementing them. So what is the typical development cycle and what kind of knowledge is required to implement these techniques? The most common development approach is a four step process in which the CP technique is programmed in Matlab or in a high-level programming language such as C/C++. The user then goes to the desired location to capture image data and returns to the computing environment where the data is processed by the programmed implementation (this cycle is illustrated in Figure 4.1). Despite the success of this approach in the research community, there are three major shortcomings for non-technical users applying this approach. First, users must un-

derstand, what are often, highly-mathematical algorithms and they need to possess significant programming skills to implement them. Secondly, if the captured images are not sufficient or were captured with error, this would not be discovered until post-processing of the images, requiring the user to recapture the image data. This development cycle would iterate until the user was satisfied with the implemented algorithm and the captured image data. But since this iterative cycle could take a considerable amount of time, the original scene may have changed or been lost completely. Third, this development style is not consistent with the way most photographers work. Photography is a visual endeavor and photographers generally work in a creative cycle where they iterate between taking pictures, applying quick edits or effects to the pictures, inspecting them, and retaking bad shots, all in rapid succession.

Recently, a second development approach for implementing CP techniques has been proposed which is intended to improve the typical four step development cycle just mentioned. This approach involves the use of a camera with a programmable back-end, such as the FrankenCamera [ATP⁺10] to execute a programmed CP technique on-camera in the desired location alleviating the need to transport the images data back to the development computer. At first glance, programmable cameras seem to mitigate the shortcomings of the previously mentioned development cycle. However closer examination reveals that the development cycle remains largely the same. This is because any change that is needed to modify the programmable camera program requires traveling back to the development environment to make the changes and re-upload the modified program to the camera. Travel could be minimized by carrying a laptop for development and uploading of new camera pipelines, but this still only a partial reduction in time delay. As previously mentioned photographers tend to work on a creative cycle that iterates quickly from shot to shot,

so development needs to be quick. Thus, all the shortcomings that exist in the previous development approach also exist with the programmable camera approach. Additionally, these cameras impose a steep learning curve on non-technical users since they are programmed through an Application Programmer Interface (API) that requires in depth knowledge of the inner workings of camera pipeline. As reported by the designers of the FrankenCamera, the system was designed for use by C P practitioners [ATP⁺10].

We describe a new User Interface (UI) for use with programmable cameras that addresses the shortcomings of both the previously mentioned development approaches for applying CP techniques. Our UI provides a visual programming interface that abstracts all the technical details of programmable camera pipelines to simply arranging a sequence of visual blocks. Each visual block, which we call filters, is automatically mapped to an atomic operation and its underlying code that performs the operation in the camera pipeline. Our UI also provides an extensive library of pre-programmed filters, which the user can easily add to existing or newly created camera pipelines, without requiring programming knowledge or an understanding of the technical details of the inner workings of the camera. Additionally, the interface provides immediate WYSIWYG feedback of filter and pipeline changes., thereby alleviating the shortcomings of the previous CP development approaches by allowing quicker turn-around time for creating and editing programmed camera pipelines (seconds vs. hours). The interface also provides a simplified abstraction of the camera pipeline that is more intuitive to program than current programmable cameras, that minimizes the learning curve for creating and testing CP techniques. Additionally, since programming is done on the camera, the user does not need to travel back and forth within the development.

The main contributions of this work are:

- A novel visual programming interface for programmable cameras.
- A simplified programming model and camera pipeline abstraction that reduces the learning curve for non-technical users.
- WYSIWYG feedback that allows users to immediately view changes to camera pipelines as they develop or edit filters, shortening the long development cycles of previous approaches.
- An extensive library of pre-programmed filters that facilitates rapid prototyping of new CP techniques and camera pipeline development.

The rest of this section is organized as follows; Section 4.2 gives a quick overview of programmable camera pipelines, their current UI and the target audience is described in section 4.3. Section 4.4 provides an overview of the main tasks our UI is designed to perform. Section 4.5 describes envisaged user interactions with the UI and how they relate to typical interactions on existing camera UIs.

4.2 Background

Before presenting the programmable camera User Interface, we give a brief overview of the image capture process used in traditional cameras by describing the role of each stage in that process. We then contrast the traditional camera capture process with that of programmable camera CP.

Traditional or non-programmable camera pipelines in current commodity cameras generally have a fixed three-stage pipeline [RSYD05] that converts rays of light that enter the camera to a final image (Figure 3.1). These three stages can be categorized as Capture, Image Adjustment, and Encoding & Storage as shown. In

the first stage, light enters the optics and is focused on the image sensor. The sensor then converts the photons into a digital signal that is processed and eventually demosaiced into a raw image. The second stage, which is responsible for most of the color rendering of the image, performs a series of operations on the image such as white balancing, gamma correction, tone-mapping, color conversion, and other image adjustments. This stage is generally camera manufacturer specific and subject to trade secrets. The last stage converts the image to its final color space and encodes the image to storage as a JPEG, PNG, or any other format supported by the camera.

Programmable cameras generally replace stage 2 (Image Adjustment) with an arbitrary color rendering pipeline stage that can be programmed by the user. By providing users with access to this stage of the camera, programmable camera pipelines allow significant control over the resulting image.

Recently, several different architectures for programmable cameras have been proposed in the scientific community. As the specific details of any particular programmable camera architecture are not the focus of the described UI design, the our approach is hardware agnostic, providing a simplified abstraction for pipeline operations into on a visual filter paradigm and abstracting the camera pipeline into a sequence of filters (described in the next section). As such, we believe that the our UI design would work in conjunction with any programmable camera architecture and the underlying hardware and software.

4.3 Target Audience & Context

In order to best describe the target audience, it is useful to place camera users on a continuum based on the extent of control they require and/or desire over how

the camera renders an image as illustrated in Figure 4.2. We can assume these demands will, in turn, dictate the type of camera a user needs. On one end of the continuum is the novice camera user. The novice user requires and/or desires little or no control over the camera; they literally just want it to work when they click the button. As a result, they are likely to select a "point and shoot" type camera to fulfill their photography needs. Next on the continuum would be the experienced but non-technical camera user. The experienced user has an advanced working knowledge of camera functions and desires more control over a broader selection of features with more sophisticated settings. Experienced users are likely to select professional type cameras such as the SLR style that allow for interchanging lenses and the ability to exert greater control over the produced images. However, the experienced user does not typically have (nor desires to have) the technical understanding of the complex inner workings of the camera. Which leads us to the other end of the continuum - the technical camera user. Technical users are simply those who indeed possess the in-depth, technical knowledge of every aspect of the camera pipeline and functionality (e.g., academic researchers in the field of computer vision or computational photography). Cameras utilized by technical users are generally specialty devices sold by specialty camera manufacturer, such

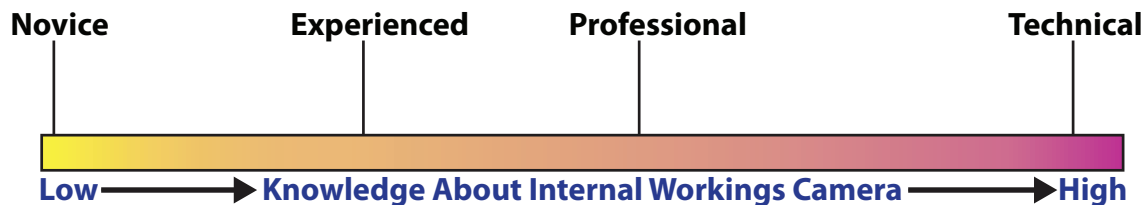


Figure 4.2: A continuum that describes the target audience with the left hand side labeling the novice user all the way to the right handside which labels the technical camera user. The continuum axis is in terms of technical knowledge regarding the inner workings of the camera. Although, this figure may not classify every person, it is used as a general guide of how we targeted our users for PCam.

as cameras for machine/computer vision, security, image processing, and laboratory use.

The target audience for the developed UI is the experienced, non-technical user. Because he lacks the necessary technical knowledge and may be unwilling to tackle the steep learning curve, the experienced user is unable to take advantage of the greater level of control over image manipulations afforded by programmable cameras. Given that the new UI would not require highly technical knowledge and would minimize the learning curve with the use of programmable cameras, it appears highly suited to meet the needs of experienced users who desire to optimize the extent of control over image capture. Although the experienced user is the target audience, the described UI could also be advantageous to technical users since it incorporates a simplified programming model and pipeline abstraction that could result in increased efficiency over existing textual-base programming interfaces.

4.4 Filter-Based Abstraction for On-Camera Processing

4.4.1 Visual Filters and Camera Pipelines

As described in the previous section, stage two of the camera pipeline is comprised of a sequence of image manipulation and adjustment operations. At a fundamental level, image manipulation involves receiving an input image, modifying it, and outputting the modified image. This behavior we call *filtering*. Therefore, any image manipulation operation that conforms to this behavior is referred to as a filter. We can then use this idea to simplify the notion of a camera pipeline to be a sequence of filters, which are chained together so that the output of one filter is the input to

another. This results in an accumulation of all the filter effects on the input image, which dictates how the final image looks.

Furthermore, we can define a programmable camera as a camera which has a pipeline with filters that can be interchanged with other filters. The inspiration for this pattern comes from the Visual Design Pattern (VDP) [STST00] concept found in the visual dataflow programming field. VDPs specify a predetermined layout or organizational structure for a program, with empty slots where a particular type of process can be inserted dynamically at run time.

In particular, the camera pipeline abstraction contains empty slots which are filled by the user with filters (which are themselves abstractions for image manipulation operations). These abstractions help to simplify the means by which a programmable camera pipeline can be constructed by reducing the complexity to that of simply arranging filters.

A filter has three attributes associated with it: the underlying programming code that implements the operations, a set of parameters that control the filter's behavior, and an icon which displays visually how the image data looks when operated on by this filter with the current set of parameters. The users do not see the code that performs the operation or the mechanisms that execute the code; users only interact with the filter by manipulating the parameters (including its position in the pipeline) and receives visual feedback on the rendering of the filter.

The described UI has a full library of available filters for user to utilize for creating new camera pipelines. The provided filters can be grouped into several categories including color conversion, image processing, computer vision, and some miscellaneous effects. All of the filters provided generally perform a single task in order to facilitate the ease at which they can be chained together to create more sophisticated effects. The provided UI does not enforce this but users should consider

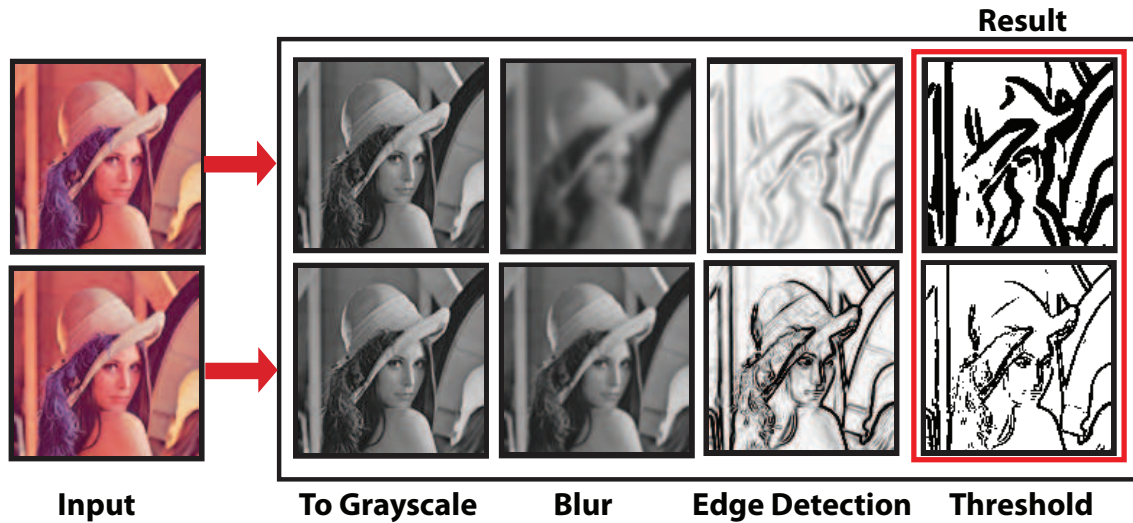


Figure 4.3: A sample programmable camera pipeline with four filters. The first filter applied to the input image converts the image to grayscale. The second filter applies a Gaussian blur. The third filter is a Sobel edge detection algorithm. Fourth filter is a threshold operation. The two rows have different results due the Blur filter has a different parameter value.

this when designing their own filters. The current UI provides just over fifty filters. We envision that future versions of the UI will provide mechanisms for users to share individually developed filters as well as pipelines.

4.5 Task Determination

Our UI was designed to make three tasks easier on-camera: 1) Selecting, cascading and re-ordering filters that are available in a filter library, to create new camera pipelines 2) Testing and previewing camera pipelines and 3) Editing and fine-tuning the parameters of individual filters after they have been added to the currently active pipeline. We have not yet included a task for creating new filters from scratch on the camera because this task requires programming and compiling, which is beyond the scope of the visual programming paradigm described here. However, we have implemented and provided several filters covering a broad range of image manipu-

lation operations, and made them available in a filter library to users. Despite our best efforts, it is inevitable that users will want to use additional filters not available in our filter library. In future work, we hope to provide a mechanism for designing new filters without violating our visual programming paradigm. Our main camera tasks are now described in some detail as well as how users interact with the UI to accomplish each task.

4.5.1 Task 1: Pipeline Creation

The pipeline of camera stages determines how the raw image generated by the camera’s image sensor, is rendered [RSYD05]. The UI allows users to reconfigure the programmable camera stages (Stage 2 in Figure 3.1) using simple graphical interactions. Specifically, our UI allows users to create new camera pipelines by selecting filters from a filter library, chaining multiple filters together, and re-ordering them. The process of creating a pipeline is initiated from the selection camera screen (Figure 4.4). Initially, an empty camera pipeline (no filters) is created. The user can then select filters from a list to populate the pipeline. The idea of creating an empty pipeline with ”slots” that can be filled with filters is an instance of a VDP, an abstraction that was originated by Shizuki, et al [STST00].

A valid pipeline must contain at least one filter. Therefore, once an empty pipeline is created, the user is immediately presented with a filter selection dialog. Once a filter is selected, it is added to the current pipeline. The user can then either add more filters to the pipeline or edit the properties of the pipeline as a whole (as opposed to editing individual filters, which is discussed in the next section). This process is repeated until the user is satisfied with the filters that make up the pipeline. The user may also change the sequence of filters in the current pipeline.

Because of the limited screen space on the camera for user interaction, we keep

the screens for each task very specific. Each screen focuses on a few actions. Each filter that is added to the pipeline has pre-determined, default parameters that cannot be modified until the pipeline creation process is finished. Once satisfied with the created pipeline, the user can indicate this by selecting *Done*. At this point the user is brought to the pipeline editing testing screen where additional pipeline editing, testing and previewing can occur.

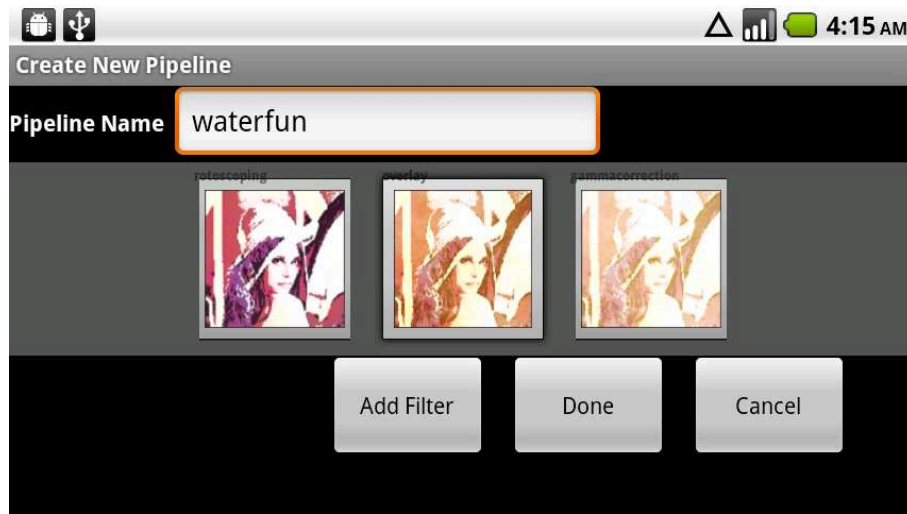


Figure 4.4: Screen shot of the pipeline creation window. This newly created pipeline has been given the name *fun* and has four filters associated with it. The filters are ordered left to right, with the leftmost filter being applied to the input image first.

4.5.2 Tasks: 2 & 3, Filter and Pipeline Editing

To avoid redundancy during exposition, we shall describe both the pipeline and filter testing tasks (Figure 4.6, Tasks 2 & 3) at the same time. To aid testing, the our UI includes a preview window (see Figure 4.5) that shows images resulting from any changes made to any selected filters and the cumulative effect on the pipeline. Previewing the results of the entire pipeline or just selected filters is a novel idea for use with programmable cameras. Also, since graphics rendering is a subjective endeavor, users like to inspect the effects of any changes before making new ones.

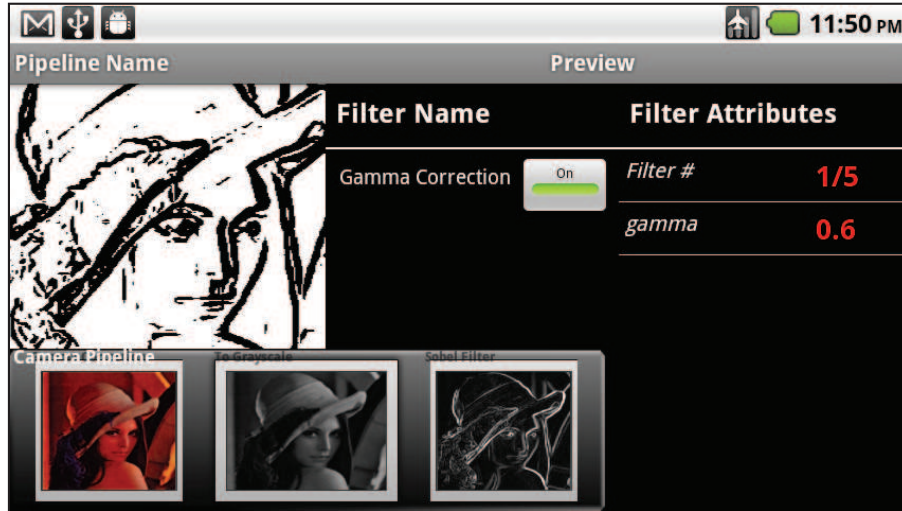


Figure 4.5: screen shot of the pipeline editing window with the standing example of edge detection camera pipeline.

We believe that the immediate feedback on-location is a powerful means for rapid prototyping of pipelines because it helps to facilitate the often complex task of building a pipeline of filters by constantly providing visual feedback . Additionally, we believe that our claim that the UI described in this work reduces the learning curve for experienced users is a direct result of the visual feedback provided by the UI. Abram and Whitted found that during visual programming of Shader Networks for 3D rendering, users did not need a complete technical understanding of the underlying Shader algorithms or implementations, as long as they were provided with immediate previews and the ability to "tinker" with shader parameters [AW90].

The filter testing task allows the user to change the values of filter parameters in order to influence how a filter renders. When presented with the editing screen the user can select any filter in the currently selected pipeline for editing. Selecting a filter brings it into focus and widgets for editing its parameters are provided (Figure 4.5). The user can then change filter parameters by manipulating their respective widgets and immediately previewing the effects of parameter values that they choose on the rendering of that filter. Supported parameter types are floating

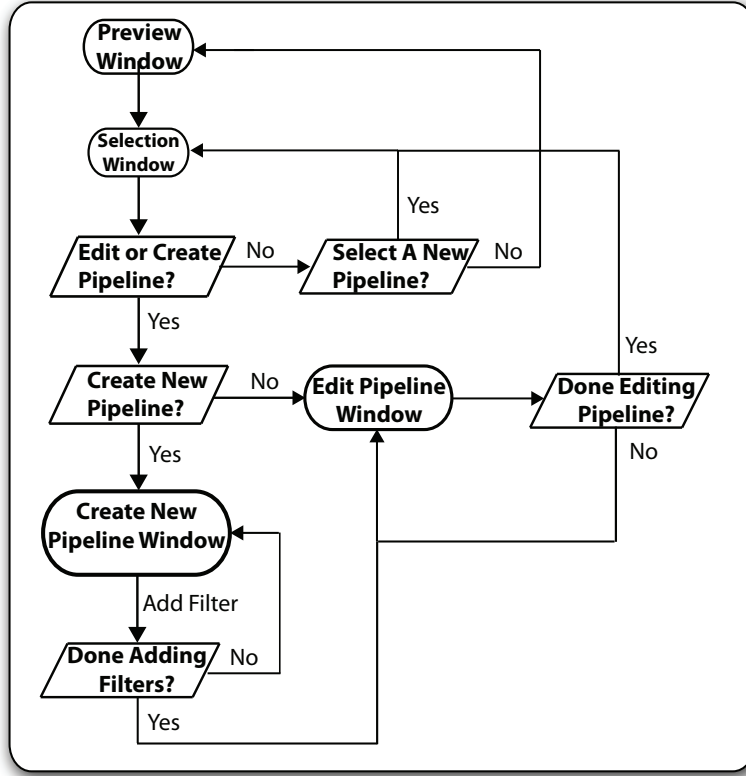


Figure 4.6: Flow diagram depicting the interaction and flow between the various windows of the UI described in this work.

point, integer, boolean, matrix, and vector values. After the user edits the value of a filter parameter, an internal variable is updated and the parameter values are automatically propagated to the renderer where a preview is generated. The user can iterate between changing filter parameter values and previewing the filter's rendered look until they are satisfied. The pipeline testing task involves previewing the cumulative effect of modifying the filters that make up the pipeline. Similar to the filter testing task, the pipeline testing task is iterative and is finished when the user is satisfied with the visual result of the pipeline. One difference between filter testing and pipeline testing is that pipeline testing is intended to be progressive. During pipeline testing, the user can repeatedly activate and test any filter within a given pipeline until the user is satisfied with the accumulated result of the pipeline.

Additionally, the user can add, remove, and re-order filters as needed in order to finish the task.

A filter can be temporarily de-activated from the current pipeline by, toggling, the active widget next to the name of the filter. This removes the influence of this filter and the pipeline is rendered using the remaining active filters (see Figure 4.7). A filter can be removed from the current pipeline by delete widget. Removed filters can only be reintroduced into the pipeline by selecting them again from the filter library. To reduce visual clutter, adding a new filter to the pipeline is initiated using a pulldown menu, which presents a list of available filters to the user to choose from. Once a filter is selected it is automatically added to the end of the current pipeline. The user can then proceed to change the filter's position in the pipeline using the position widget for that filter.

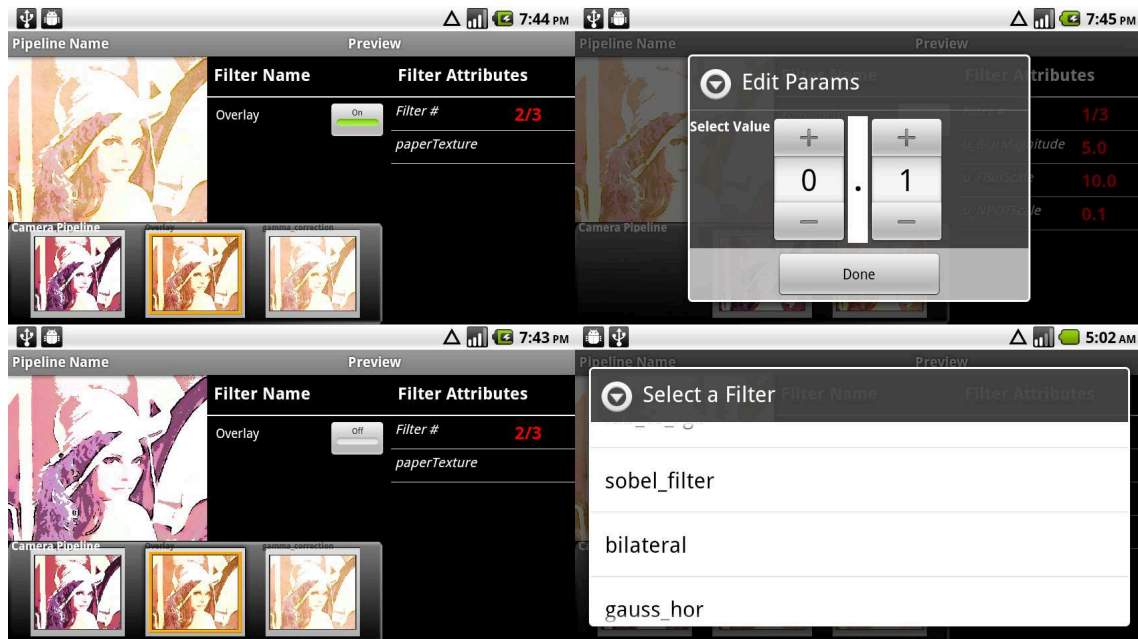


Figure 4.7: This figure shows four screenshots of widgets. Left side on the top and bottom show the toggle button for individual filters and how it effects the camera pipeline. Right show two widgets for editing filter parameters (top) and adding a new filter (bottom).

4.5.3 Layout & Interaction Scenarios

Our UI was designed to be used on the camera specifically for organizing, manipulating, and creating programmable camera pipelines. Because of this narrow focus, it is not intended to replace the existing camera UI but rather to augment them. As illustrated in Figure 4.8, the new camera UI layout hierarchy contains both the new UI as well as typical representations of a traditional camera UI. The functionality of both UI layouts are mutually exclusive with the exception of the preview window at the top. As a practical note, this means that any manufacturer-supplied traditional camera UI does not need to be modified. A user simply installs the UI implementation as extra software that augments the manufacturer’s UI. The rest of this section describes in more detail the relationship between the traditional and our interfaces as well as the function each new window performs. We end the section with a description of user interaction scenarios for the UI described in this work.

4.5.3.1 Layout

Our UI design provides three new additional windows to the traditional camera UI and repurposes a fourth for live previews of the programmed camera pipeline effects (see illustration in Figure 4.8). The preview window provides views for both the live programmable camera rendering and traditional camera rendering. Because the programmable camera pipelines can produce images that are potentially hard to discern, this window has a preview switcher widget that allows the user to toggle between the programmable camera pipeline and a traditional view to aid the camera user. In this way, during debugging, the user can switch to the traditional camera image when the image rendered by the programmable pipeline is indiscernible.

The pipeline selection window provides two functions: first it allows the user to select the active pipeline and acts as the entry point for pipeline creation (screen

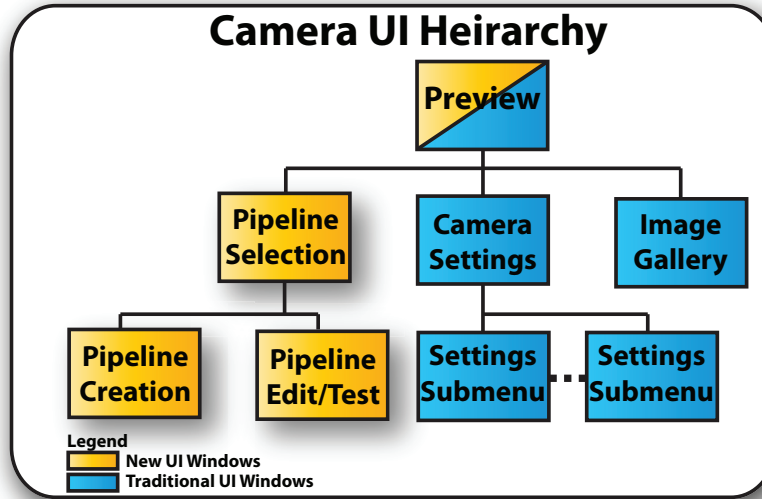


Figure 4.8: An Organizational chart of the UI layout (yellow) and the traditional camera UI layout (blue)

shot in Figure 4.4) and editing windows (screen shot in Figure 4.5). To start the new pipeline wizard (we use a wizard to enforce pipeline attributes are initialized, i.e., each pipeline must have a name and at least one filter), in the selection window, the user taps the *create new pipeline* button from the window's options menu. To select a pipeline for editing, the user chooses the desired pipeline from the list of pipelines presented and is transitioned to the pipeline editing window with the selected pipeline. The pipeline creation and editing windows have already been described in detail in the previous section, and all information and interactions needed to organize, create, and manipulate the camera pipelines is presented in these four windows. The complete flow between all the windows within the UI is mapped out in Figure 4.6.

4.5.3.2 Interaction Scenarios

Each interaction scenario with the programmable camera pipeline UI is displayed in an interaction diagram in Figure 4.10 and a corresponding screen view is pro-

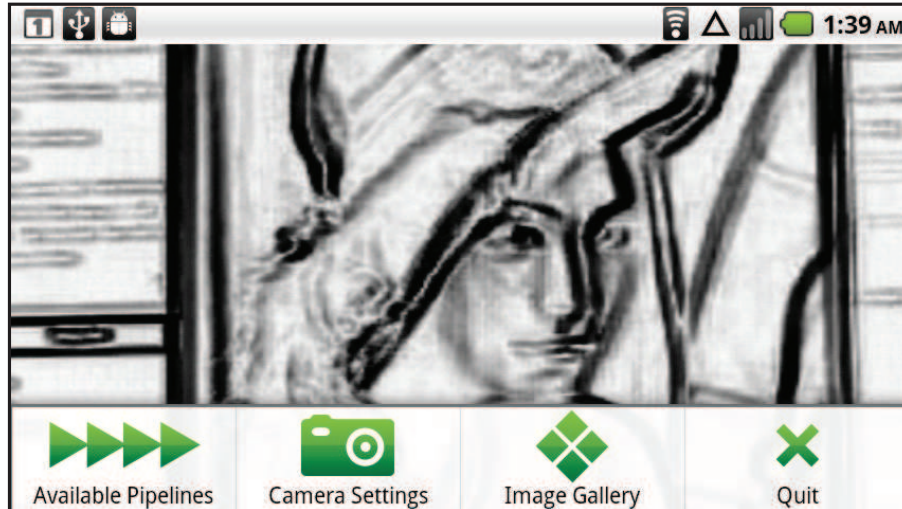


Figure 4.9: Screen shot of the preview window. The menu on the bottom can move onto and off the screen by toggling the menu button.

vided in Figures 4.5, & 4.9. Interaction within the UI design starts at the preview window. From there, users navigate to the rest of the programmable camera UI through the pipeline selection window or to the traditional camera UI to edit camera properties or view images in the photo gallery. We have envisioned three main interaction scenarios, which we describe in this section, *pipeline selection*, *pipeline creation*, and *pipeline editing*. We differentiate interactions from tasks by the navigation and relationship of the task window to the other windows in the UI. In each scenario, navigating to a new window essentially places that new window on top of an interaction stack. When the user finishes interacting with that window (either by finishing the activity or clicking the back button) the current window is removed from the stack and the previous window is reinstated as the current window. For example, the *pipeline creation* scenario places four windows on the stack when the user is on the window that is actually responsible for the creation of the pipeline.

The first interaction scenario allows the user to select a programmable camera pipeline from the available pipelines. When on the preview window, the user navigates to the selection window, selects the desired pipeline and clicks the back button.

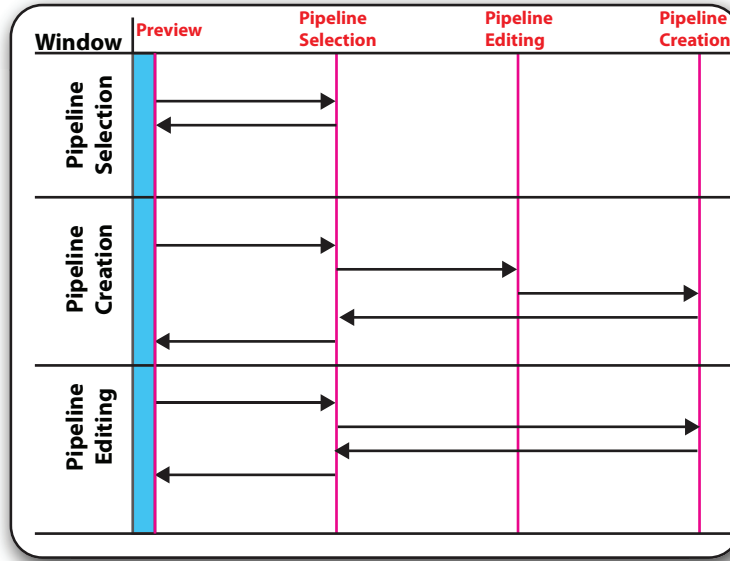


Figure 4.10: Interaction diagram

By selecting a pipeline the user activating that pipeline for use in preview and image capture, therefore when returning to the preview window the user will now see the new activated pipeline.

4.6 Conclusion & Future Work

In conclusion we have shown the potential benefit of using our UI design over existing programmable camera user interfaces. We believe that the described interface provides a unique way to create and edit programmable camera pipelines outside the typical laboratory development environment. In addition to being able to create and edit camera pipelines in the picture-taking environment, we believe that the unique visual feedback method for programmable camera pipeline development would be beneficial for any programming environment, not just for "on-the-fly" development. We also believe that the visual feedback provided by the system can lower the learning curve for CP algorithms and programming camera pipelines, making them

accessible to a wide range of experienced and expert camera users.

In future work, because the process of creating filters on-camera can be more complicated than the other three tasks we reserve this improvement for future work. In order to accomplish this, a new metaphor for inserting small programs into the pipeline has to be created. Because this is generally a text based-activity, this would be a difficult task to perform on-camera. Additionally, we would like to perform a larger scale study of non-technical end users to determine the benefits and utility of our visual programming interface, which could be novel and informative. Finally, the current implementation lacks some convenience features such as an Undo function. There are several instances where an Undo operation might be helpful, such as moving and deleting filters from the pipeline. Future versions of the UI will contain additional convenience feature to provide a better user experience with our programmable camera.

Chapter 5

Symmetric Lighting Capture and Relighting

In Section 1.3 we described the components of this dissertation as a pyramid of contributions with a programmable camera as the foundation, photometric analysis and processing as the second tier, and the final tier as the previs components. The previous chapter, Chapter 4, concluded the description of the foundation of this pyramid by providing a description of the programmable camera user interface and visual programming paradigm and implementation. This chapter describes the contributions listed in the second tier, photometric analysis and processing, which describes our Symmetric Lighting and relighting methods.

5.1 Overview

Relighting is the process of producing a new image of a previously rendered scene under new lighting conditions. What often makes relighting difficult is that information about the scene necessary to perform relighting is often missing or incomplete.

Therefore, to invert the pre-existing lighting in a scene, usually information about the scene, such as geometry, reflectance, appearance, or lighting must be estimated or captured directly. Capturing or measuring light is a difficult endeavor which requires expensive measurement hardware, complicated lighting setup, and environments that can be controlled and have known properties. The difficulty of light capture is because the problem is ill-posed due to unknown scene properties such as geometry, reflectance, and object appearance. For example, if we wanted to infer lighting from the reflectance of an object, neglecting atmospheric effects within the scene and assuming surface reflectance is not dependent on the wavelength of light (light is a ray not a wave), translucent, changing over time, in motion, or have significant subsurface scattering, then reflectance functions can be considered to have six degrees of freedom [Jen09]. Six degrees of freedom makes it difficult to infer any lighting properties with an unknown reflectance function, therefore this problem is often addressed by alternate means. For example, photographers and cinematographers often get around this problem by directly measuring the light at a point using a device called a light meter [DeC97]. The Light meter produces a single incident luminance value, which is then used to set camera exposure settings. This is analogous to using a single pixel camera to estimate light intensity (not color) at a single point in the scene, which is inadequate for estimating the illumination for an entire scene. To produce the same number of data points as a modest camera, a Light meter would need to be moved to more than 10 million locations in the scene, which is impractical. Consequently, this description is similar in spirit to a state of the art relighting facility proposed by Debevec et al. [DWT⁺02] performs relighting, which is extremely complicated having fifty High-Definition high-speed video cameras and more than one thousand controllable Light Emitting Diode (LED) lights as seen in Figure 5.1 (c).

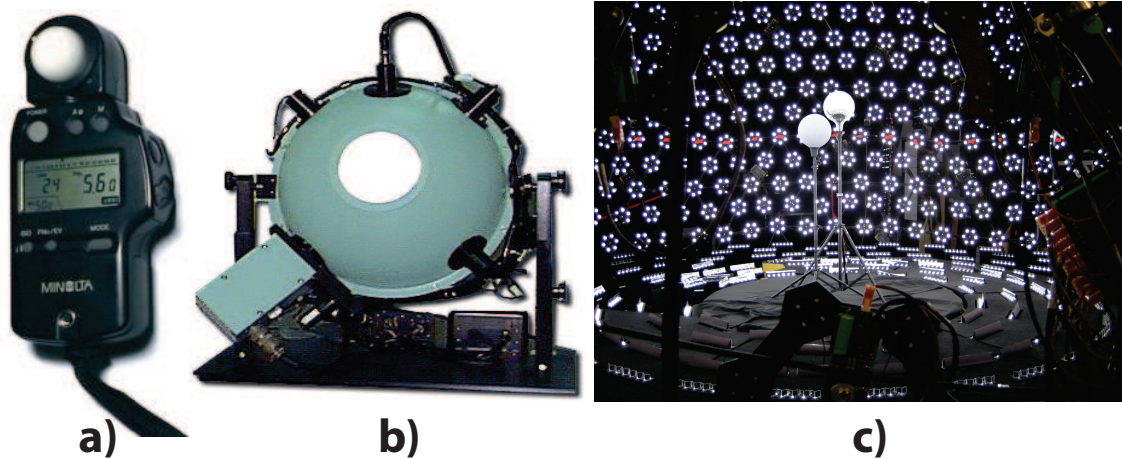


Figure 5.1: Images of devices used to capture lighting with increasing degrees of complexity and cost (left to right). Left (a): A digital light meter used for photography to estimate the intensity of the ambient light[[wik12](#)]. Center (b): is an Integrating Sphere used to determine the color of light from a particular light source which is placed in the circle opening at the front[[DeC97](#)]. Right (c): the Light Stage 6, state-of-the-art light capture and relighting facility at the USC Institute for Creative Studies [[Lin10](#)].

This chapter describes our novel photometric and active illumination method for capturing the lighting of a scene without explicitly knowing or measuring its geometry, reflectance, or appearance. We call this method Symmetric Lighting. Our method captures the proportion of illumination contributed by each light within the scene using only a single camera and no other light measurement equipment or facilities. We accomplish this by modeling the light and its interaction with the scene as a system of symmetric equations, where each light is coded with a specific and unique light color making these equations simple and efficient to solve. Symmetric Lighting facilitates the separation of lighting from all other scene properties within a camera image, making photorealistic relighting a simple and efficient operation that can be performed on-camera and on-location. In contrast to the state-of-the-art relighting [[DWT⁺02](#)], which requires orders of magnitude more processing capabilities, many high-speed cameras and lights, and is performed at a separate

facility as a post process not during the shoot in contrast to our method which is real-time.

Our main focus in this dissertation is relighting, but several additional contributions result from our Symmetric Lighting capture method. These additional contributions stem from the development of a novel data structure for storing the results of our capture method, which we call the β (Beta) map. The β map, which is derived from the camera image, represents the contribution of each light source (β value) at every pixel in the camera image. Therefore, each β value in the β map corresponds to the same location in the camera image. Image processing methods can be used to modify or interpret the β map making it a powerful representation for editing the lighting environment. In this chapter we describe image processing methods for shadow detection, and gradient-based methods for detecting and manipulating other light properties. In the next chapter and in the context of previs, we describe a novel multi-illumination white balance, color correction, and light color estimation methods that are made possible by processing of the β map. Although we derive several image-based methods for manipulating lighting via processing the β map, we hypothesize that similar methods can be utilized for capturing and manipulating many more lighting and reflectance phenomena, such as subsurface scatter, transparent and emissive surfaces, and Global Illumination, which will be examined in future work.

The majority of previous efforts for relighting a scene have focused on capturing the reflectance [DHT⁺00a, MWL⁺99, ZREB06, LKG⁺03, GCHS05, WRWHL07] of objects within a scene. Capturing geometry [FRC⁺05, FRC⁺08, SCMS01, Dye01, SCD⁺06, SCD⁺06], and appearance [LKG⁺03, WLL⁺09, FH09, SH98, KBD07, DvGNK99] of objects. Prior work on capturing lighting environments make restrictive assumptions such as requiring the lighting to be emanating from an in-

finitely distant location in order to create environment maps [BN76, MH84, Deb98b, GSHG98, UGY06]. But little effort has gone into developing techniques for capturing and manipulating the pre-existing lighting within a scene. The work presented here focuses on *identifying the influences of individual light sources have on a scene, quantifying this influence, and providing the capability to manipulate these quantities, while avoiding the complexity of measuring or inferring the other properties of the scene*. Our relighting method allows for previsualization and editing of on-set lighting in real-time, which cannot be accomplished using previous methods. This will enable directors and cinematographers to virtually preview and edit on-set lighting for greater artistic control and faster decisions on physical lighting changes, resulting in higher quality lighting in movies, fewer reshoots, and less on-set time.

The main contribution of these techniques are:

- Develop a technique for determining the relative light contributions per pixel for each individual light source, which we call *Symmetric Lighting*.
- Simplified and efficient relighting that can be done on-camera, in real-time.
- A novel data structure for storing *Symmetric Lighting* calculations called β map, which allows for image-based light manipulation beyond relighting.
- Shadow detection method that leverages the lighting information stored within the β map to detect cast and self-shadows using a threshold watershed method.
- A gradient-based method for providing second order statistics for manipulating the distribution of photons for each light, which can be applied to the β map.
- A computationally efficient implementation for each technique that make these techniques amenable to hardware implementation and executable in real-time.

5.2 Symmetric Lighting Theory

5.2.1 Image Formation

5.2.1.1 Camera Model

Before progressing, we will lay out the basic image model from which our photometric processing and image capture techniques are derived. We assume that all calculations are done in a linear RGB space $\lambda = \{R, G, B\}$ and that an image sensor provides only three color (RGB) values after a raw image is demosaiced. Furthermore, this work performs image processing in the camera’s native RGB space, which has been shown to produce less color distortion [Vig04]. These three values, which form the pixel of an image C , are the result of the illumination L within the scene and how the scene reflects this light, represented by R in the following equation. Therefore, our input image can be simply stated as:

$$C = R * L \tag{5.1}$$

Even though this model appears to be simple, there is a significant amount of complexity in the interaction of these terms. In Equation 5.2 we expand on Equation 5.1 by assuming that the sensitivities of the image sensor S , are narrow band and thus modeled as a delta function. Furthermore, if we assume that illumination is uniform across the scene¹, then we can consider the influence of the geometry G on the reflectance to be equivalent to 1. Later in this section we will relax the geometry assumption, but for now it serves to simplify our model for ease of explanation. Thus, after incorporating these assumptions, our image model in Equation 5.1 can

¹In general this is a restrictive assumption that does not reflect reality but is often made in the context of illumination estimation [Hor06]

be expanded with respect to the previously defined λ :

$$C_\lambda = G \int_{\Omega} R_\lambda L_\lambda S_\lambda \quad (5.2)$$

The expanded model allows for more complicated lighting calculations that incorporates all lighting contributions within a closed hemisphere, namely Ω , where Ω is the spherical coordinates denoted θ for azimuth and ϕ for the polar angle. The hemispherical formulation is common in rendering contexts [DBB02] as it provides convenient parameterization. Additionally, it also provides convenient way to implicitly parameterize the *importance* associated with lighting samples as natural scenes tend to contain a majority of primary light sources away from the horizon (i.e., $\phi \approx \pi/2$) [DWA04] (the cosine of the angle between light and object normal is close to 0).

5.2.1.2 Physically-based Lighting Model

We further expand the model to a more physically-based model that we will utilize later on in this section. The model for Equation 5.3, often referred to as the Rendering Equation introduced by Kajiya [Kaj86] incorporates global effects such as interreflections from indirect light sources (also called mutual illumination in other scientific domains) as well as light emitting surfaces. This lighting model mathematically formulates the equilibrium of distributed light energy within the scene in order to create a system that has a steady state observed and measured in the physical world. This steady state is consistent with the conservative nature of the rendering equation and compliments the bidirectional reflectance distribution function (BRDF), which is often used in conjunction with the rendering equation. The notion of conservation of energy and reciprocity is a key feature of the Rendering equation and of one of the novel contributions of the dissertation, which will be

discussed in section 5.2.2.

$$I(x, \vec{w}) = L_e(x, \vec{w}) + \int_{\Omega} R(x, \vec{w}, \overleftarrow{w}) L_i(x, \overleftarrow{w}) (N_x \cdot \overleftarrow{w}) d\overleftarrow{w} \quad (5.3)$$

The Rendering Equation is generally defined in terms of wavelength of light and time, but as this work does not consider either, for simplicity we ignore both. Also, the Rendering equation defines light emitting surfaces L_e , but this work does not consider this phenomena or other phenomena not accounted for in the rendering equation such as light scattering within a volume of participating media. This formulation is an integral equation that accounts for all incoming directions \overleftarrow{w} for which the angles are appropriately attenuated by a cosine factor calculated as $N_x \cdot \overleftarrow{w}$. In general this continuous function is difficult to evaluate unless approximated. We approximate by discretizing the integral and if the number of light sources is known to be N and assuming that there are no emitting light or surfaces, we can rewrite Equation 5.3 as such:

$$I(x, \vec{w}) = \sum_{i=0}^N R_i(x, \vec{w}, \overleftarrow{w}_i) L_i(x, \overleftarrow{w}_i) (N_x \cdot \overleftarrow{w}_i) \quad (5.4)$$

Where i in this case refers to the i th light source.

5.2.2 Light Contribution Estimation through Symmetry

The goal of this section is to describe the novel light capture technique we developed, which determines the relative contributions of each light illuminating a scene. Due to the complexity of the interaction between the lighting and the scene being illuminated, this is a non-trivial task involving up to 7 variables that are difficult to infer or measure from a single pixel sample (see Appendix C for a description of these variables). Despite this complexity, the technique presented here provides

straight-forward and computationally efficient method for manipulating the individual lights in a scene independently from the other lights without limiting their influence on a scene’s illumination. Our method, called Symmetric Lighting, shows that it is relatively simple to avoid much of the complexity associated with many variables that are explicit and implicit in the Rendering Equation 5.3 by performing simple interactions with the lighting environment through a technique known as Active Illumination. Active Illumination is the name given to a class of Computational Photography [RTM⁺06] methods that manipulate only the lighting in a scene in order to estimate scene properties. In our Symmetric Lighting method, we manipulate the lighting by alternating the color each light emits, which allows us to estimate the proportion each light contributes to illuminating a given camera pixel. Using the image model from Equation 5.1, we can describe the technique for estimating the lighting within a scene without knowledge the scene geometry, reflectance, and appearance of the objects within the scene.

As previously mentioned, most techniques for relighting or light estimation acquire the reflectance function of objects within the scene [Hor06]. This in turn is where much of the aforementioned complexity originates. If we consider the path of a single photon within a scene, it may interact with many surfaces and be reflected, refracted, and scattered any number of times. As can be seen from Equations 5.1 and 5.2, the pixel values captured by the camera are dependent on the lighting, the scene reflectance, and appearance, so therefore cannot ignored. In order to avoid the complexity of measuring the reflectance and appearance, we have developed a technique to implicitly estimate its value without explicitly measuring it.

Most scenes are comprised of two or more light sources with distinct color and intensity [Hor06, Ebn08, HMP⁺08, KIT05]. Despite this fact, many of the techniques developed to measure light often make the assumption of a single light source, as is

common in white balancing [Buc80, LM71, FT04, vdWGG07, GG07], Bi-directional Reflectance Distribution Function (BRDF) capture [DHT⁺00a, MWL⁺99, ZREB06, LKG⁺03, GCHS05, WRWHL07], and illumination estimation [MG97, Hor06]. The research presented here does not make the restrictive assumption that the scene contains only a single light. Instead, we exploit the notion that most scenes are illuminated by multiple lights by calculating the differences in illumination at each camera pixel due to the different colored light sources. We assume that the light source *colors are distinct and known* by the user and we utilize this information to discover the proportion of light each source contributes to a given camera pixel $C_{i,j}$ from an image C with dimensions $\{i, j \in I \times J\}$ for our camera model in Equation 5.2. Let us assume that the total light impinging on a point P that is reflected into the camera and contributes to $C_{i,j}$ is some quantity of light, call it L_{total} . Then each light source will contribute some proportion of its photons to L_{total} seen by pixel $C_{i,j}$.

If we abide by the physical constraint that the number of photons is conserved and unobstructed in free space without volume scattering due to atmospheric effects or other participating medium, and we assume that if a scene contains two lights, then the proportions from two lights l_1 and l_2 forms a linear relationship from the additive nature of light. This relationship can then be stated as a linear interpolation between the two lights.

$$L_{total} = l_1 * \beta + (1 - \beta) * l_2 \quad (5.5)$$

Then for each camera pixel $C_{i,j}$, there is a value β , which describes the proportion of the total number photons that each light contributes to illuminating the pixel $C_{i,j}$ captured by the camera. We then extrapolate this idea for all camera pixels $C_{i,j}$ that form the 2D camera image C with dimensions $I \times J$ and $\{i, j \in I \times J\}$. We then

form the notion of the image's corresponding β map with the same dimensions $I \times J$, where $\{\forall C_{i,j} \in C, i, j \in I \times J \mid \exists \beta_{i,j} \in \mathfrak{R}_{[0-1]} \mapsto C_{i,j}\}$. In other words, every camera pixel has a β value and every camera image has a β map (the β map is discussed in more detail in Section 5.3). If we assume that the camera pixels intensities are linear with respect to the scene illumination intensity², then the proportion between the lights and the camera model from Equation 5.1 is preserved. This idea is illustrated in Figure 5.2.

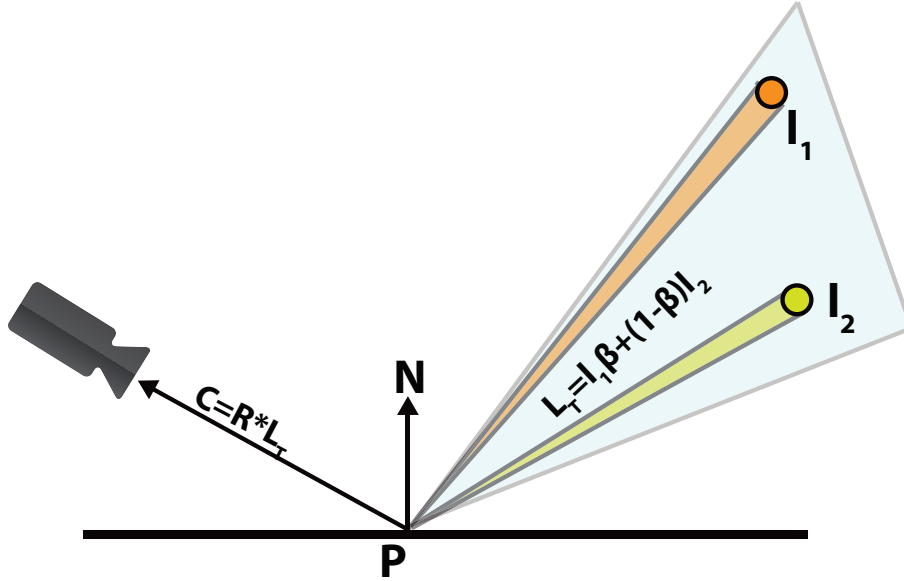


Figure 5.2: For two lights sources illuminating point P reflecting into the camera which captures the interaction between the point being illuminated and the sources. Symmetric Lighting determines how much each light source contributes to the light total received by the camera, which we call the β value, by solving Equation 5.7 by interchanging the intensity values (i.e., color) of l_1 and l_2 .

Finding the relative contribution of each light to illuminating a point is the same as determining β from Equation 5.5. There are many ways to determine the β value at each pixel for a scene with two lights such as illustrated in Figure 5.2, most of which are not feasible. For example, we could measure the orientation of the lights

²Although cameras do not in general produce pixels that have intensities that are linear with respect to the illumination, linear correction is often applied to make the camera's response curve linear [RKAJ08].

and surface normal of each point corresponding to a camera pixel, but this would be tedious and error prone. On the other hand, determining the β values numerically by inference using only the pixel values from a single camera image and solving for the β values using the equation in Figure 5.3 is not feasible either, due to the ill-constrained nature of this problem, which would result in a non-unique solution for the β values. Actually, the solution set for this problem is $\beta \subseteq \mathbb{R}[0 - 1]$, which contains an infinite number of solutions for β . In other words, with a single camera image, for each pixel and β value, we have a single equation with two unknown variables as seen in Figure 5.3.

$$\mathbf{C} = \mathbf{R} * (\mathbf{I}_1 \beta + (1-\beta) \mathbf{I}_2) \quad \Rightarrow \quad \begin{array}{l} \text{1 Equation, 2 Unknowns} \\ \beta = \text{any value } [0-1] \end{array}$$

Unknown
Unknown
Unknown

Figure 5.3: The general camera model from Equation 5.1 with two lights. Here we show that with only a single image captured C , that this single equation has two unknown variables and cannot be solved for uniquely.

To solve for β we introduce the novel idea of *Symmetric Lighting*, which allows us to introduce further constraints on the ill-constrained formulation for determining β . The idea is that while keeping the *camera and light positions constant and assuming the lights emit different colored light*, we acquire two images of the scene. After acquiring the first image, the second image is acquired with the colors between lights swapped³. If we consider each corresponding pixel location $\{i, j\}$ in both camera images $C_{1i,j}$ and $C_{2i,j}$, then each pixel value (an RGB triplet) results from the two light $\{l_1, l_2\}$ and the reflection function R as indicated in the equation from Figure 5.3. Since BRDFs in general are parameterized by the orientation of the lights and

³As previously stated, Symmetric Lighting assumes the lights emit different colored lights. In Section 5.2.3.2 we describe what happens when this assumption is violated.

camera and the orientation remains constant, R remains constant between the two equations. Also, due to reciprocity and not moving the camera or lights (we assume that reflectance is not wavelength dependent), the β value or proportion will remain constant between the two equations while the pixel values $C_{1i,j}$ and $C_{2i,j}$ change. Therefore, capturing two images provides us with two equations (one from each image), while maintaining the same number of unknown variables β and R .

Because all variables remain constant except for the light color values, in the case of a scene with two lights, we can solve for β by setting up two simultaneous equations. We assume that the reflection and appearance of the point being illuminated in the scene does not vary with time, which allows for the substitution of the implicit value for the reflectance function R , giving a constrained objective function for solving a single unknown, β .

$$\begin{aligned} C_1 &= R * L_{T12} \quad | \quad L_{T12} = l_1 * \beta + (1 - \beta) * l_2 \\ C_2 &= R * L_{T21} \quad | \quad L_{T21} = l_2 * \beta + (1 - \beta) * l_1 \\ \frac{C_1}{L_{T12}} &= R = \frac{C_2}{L_{T21}} \end{aligned} \tag{5.6}$$

Where L_{T12} and L_{T21} are the lighting color values from the right hand side of Equation 5.5. To determine the optimal value for β in the least squares sense we minimize the following objective function.

$$\underset{\beta \in [0-1]}{argmin} \|C_1 * (l_2 * \beta + (1 - \beta) * l_1) - C_2 * (l_1 * \beta + (1 - \beta) * l_2)\|^2 \tag{5.7}$$

For each pixel location $\{i, j\}$, there are two different pixel values, $C_{1i,j}$ and $C_{2i,j}$, which are the direct result of our image model from Equation 5.1. For each pixel location, solving Equation 5.6 to determine the β for each pixel, essentially

constructs a β map for the camera image C. We now have a map that provides the proportions each light contributes to each pixel in the camera image C. In Section 5.3 we show how to use the β map to perform relighting.

To gain further intuition on how our Symmetric Light capture works, we present a geometric interpretation using Euclidian geometry in the 3D space of the RGB color cube as illustrated in Figure 5.4. If we plot the light values of l_1 and l_2 in the RGB cube, they form two points in the cube. By definition, a line segment consists of every point between two endpoints. If we consider the line segment created by the color values of lights l_1 and l_2 , this line segment $\overline{l_1 l_2}$ spans the range of possible color values that any mixture of these lights can have. If we then multiply the light line segment $\overline{l_1 l_2}$ by a pixel of the color C_1 , this corresponds to a transform of the line segment to a new location in the RGB cube, call it $\overline{C_1 l_1 l_2}$. The new line segment $\overline{C_1 l_1 l_2}$ spans a color range, where every point on the line segment is some interpolation of the end points $C_1 l_1$ and $C_1 l_2$ where β is the interpolation variable. If we also consider a second transformation of $\overline{l_1 l_2}$ by a different pixel color C_2 , this forms a different line segment, call it $\overline{C_2 l_1 l_2}$. Since the two line segments $\overline{C_1 l_1 l_2}$ and $\overline{C_2 l_1 l_2}$ are related by Equations 5.6 and 5.7 and our objective function finds the proper interpolant value β that solves these equations. Then the solution is just the β value where $\overline{C_1 l_1 l_2}$ and $\overline{C_2 l_1 l_2}$ intersect as indicated in Figure 5.4.

The geometric interpretation provides a different way to solve the objective function in Equation 5.6 by finding the intersection of the two lines $\overline{C_1 l_2 l_1}$ and $\overline{C_2 l_1 l_2}$. This formulation of our model works for reflection functions that are not dependent on the orientation or position of the light sources. In other words, Equations 5.7 and 5.6 assumes a Lambertian reflection function. If this assumption is violated, the estimate of the β value will contain some error ϵ (Epsilon). This error can geometrically be interpreted as the distance between the two lines $\overline{C_1 l_2 l_1}$ and $\overline{C_2 l_1 l_2}$,

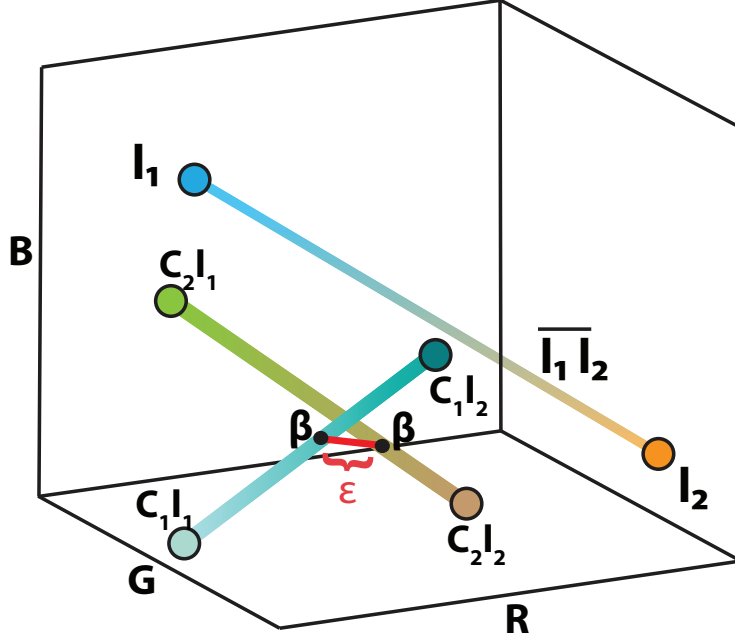


Figure 5.4: The relationship between camera pixel C_1 and C_2 located in successive images at the same pixel location (i,j) and the two lights that provide a relative contribution in linear RGB space. The two light sources l_1 and l_2 form a line $\overline{l_1 l_2}$. Each pixel in the image can be considered illumination by some linear interpolation of l_1 & l_2 denoted β . A solution to Equation 5.7 is the intersection of the two line $\overline{C_1 l_1}$ and $\overline{C_2 l_2}$, if there is one. Any error in the estimate of the β value can be interpreted as the closest distance between the two line segments, labeled ϵ in the diagram.

which is labeled in Figure 5.4 as ϵ . In the next section we expand this model to include more than two light sources as well as describe how to extend our technique beyond the Lambertian reflectance assumption and describe a formal model for the error ϵ .

5.2.3 Expanded Symmetric Lighting

In the previous Subsection we introduced our theory of Symmetric Lighting, which we showed how to determine what proportion of two lights contribute to illuminating a point given the assumption of Lambertian reflectance. Because real world scenes often present reflection and lighting that is more complicated than just a diffuse re-

flexion with two lights sources, in Subsection 5.2.3.1 we present an expanded view of our idea of Symmetric Lighting. In particular we generalize Symmetric Lighting to n-lights without any knowledge about their physical location. Additionally, we expand the geometric interpretation for Symmetric Lighting from the previous subsection by abstractly relating lighting configuration with n-lights to that of an 2-dimensional polygon. This geometric interpretation helps build intuition for the mathematical representation of Symmetric Lighting as well as providing a simplified explanation of degenerate cases in the lighting setup. Also, it is reasonable to assume that most scenes will violate the Lambertian reflectance assumption and that a certain amount of error will be present in the β map as a results. In Subsections 5.2.3.3 and 5.2.3.4 we model the error resulting from specular reflections and utilize this model to develop an expanded Symmetric Lighting model for use on scenes that exhibit non-diffuse reflections.

5.2.3.1 Symmetric Lighting in N-Lights

Recall Equation 5.6, which shows the relationship between a pixel in a single image $C_{i,j}$, the reflectance R , and the proportions of each light illuminating a point L_T . The first step towards generalizing Symmetric Lighting is extending the L_T to n-lights, which was previously defined for only two lights by $l_1 * \beta + (1 - \beta) * l_2$ and β representing the proportions. The generalization of L_T to n-lights and their respective proportions now follow the form:

$$(\beta_1, \dots, \beta_n) \in \mathbb{R}_+ \quad \left| \quad \sum_{n=1}^n \beta_i = 1 \text{ and } \beta_i \geq 0 \text{ for all } i \right. \quad (5.8)$$

A lighting configuration with n-lights can essentially be modeled as an 2-dimensional geometric object or polygon with $\{\beta_1, \dots, \beta_n\}$ representing the polygon's Barycen-

tric coordinates. Thus giving L_T the following representation.

$$L_T = B \cdot L = \left\{ \sum_{n=1}^n \beta_i * l_i \right\} \quad (5.9)$$

This represents the set of all lights defined by the system and for every light there is a corresponding proportion β_i that quantifies its contribution. This provides a system of equations in terms of the lights and their proportions that is always fully determined and therefore always has a solution. These system of equations are :

$$\begin{array}{cc|c} C_{i_1} = & R * L_{T_1} & \\ \vdots & \vdots & \\ C_{i_n} = & R * L_{T_n} & \end{array} \left| \begin{array}{c} L_{T_i} | T_i \neq T_{i+1} \end{array} \right. \quad (5.10)$$

What Equation 5.10 show is that L_{T_i} is the i th lighting total after rotating the light colors l_i to a different lighting locations. This is analogous to the light colors being vertices on the polygon and each light total L_{T_i} is a rotation of these vertices, which is illustrated in Figure 5.5. For the two-light case, we used the term "swap" in place of "rotation" because we had not introduced the geometric notion of the lighting. Since in the two-light case, the geometric representation is a line, therefore the notion of rotation holds for lines. From here onward we will use the term "light rotation" in place of swap as it appropriately characterizes the operation. The light rotation is restricted to placing l_i in a previously unvisited location (non-repeating) of the light locations. Because the proportion of lighting impinging on a point β_i is determined by the light's location, β_i remains fixed, whereas rotating l_i provides additional equations for balancing and solving 5.10. We use a non-repeating rotation instead of a random permutation of the lights in order to describe the lighting Equation 5.10 as a positive definite matrix, which

allows us to use a more efficient method for solving the simultaneous equations. We describe the complexity of solving these equation in more detail in Section 5.2.3.2.

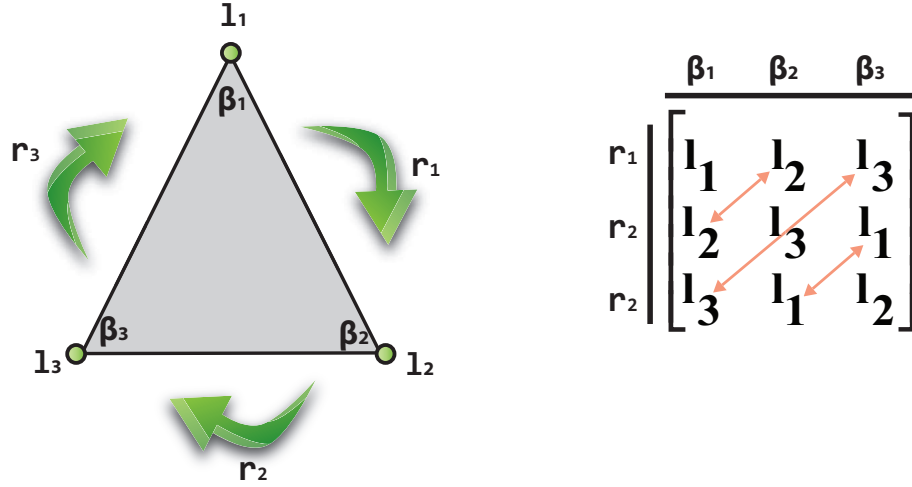


Figure 5.5: A three vertex polygon (triangle) representing a three light capture system. The geometric relationship shows the symmetry of the system and how, through rotations of the light colors l_i we can have a system of equations representing the lighting. Additionally, we have listed the lighting values in matrix form with the corresponding rotations (left side of matrix) and the proportions (β values top of matrix). This figure illustrates that rotating l_i does not change the proportions β_i , which are determined by the light locations but only changes the resulting camera pixel value C_i based on the change in light values l_i in our system of Equations 5.10.

As can be seen in Figure 5.5, the process of capturing the proportions of lights is based on a symmetric relationship between the lights and their mathematical representation. The light colors l_i are rotated about a center axis to produce the system listed in Equation 5.10. In Figure 5.5 for example, we have three lights $\{l_1, l_2, l_3\}$, three locations (vertices), three proportions $\{\beta_1, \beta_2, \beta_3\}$, and three rotations $\{r_1, r_2, r_3\}$. Therefore we can consider the geometric interpretation of this configuration as being a regular polygon (symmetric) with the number of vertices, n , corresponding to the number of lights in L_T . The rotations of lights to produce the

necessary system of equations, results in a lighting matrix that can be represented as a positive definite symmetric matrix as seen in Figure 5.5. The symmetry within the lighting setup and system of equations is the reason our light capture technique is referred to as *Symmetric Lighting*.


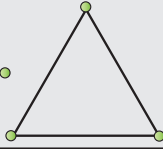
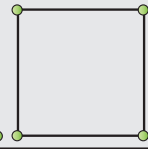
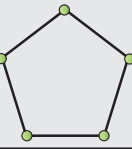
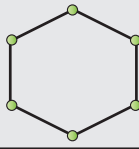
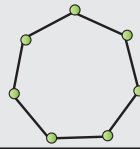
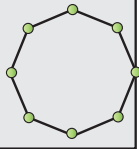
Image							
Name	Line Segment	Triangle	Square	Pentagon	Hexagon	Heptagon	Octagon
Schlafl #	2	3	4	5	6	7	8
# of Vertices	2	3	4	5	6	7	8

Figure 5.6: A listing of the first seven n-vertex polygons in 2-dimensional Euclidean space that corresponds to the lighting configuration with Schlafl numbered polygons equaling the number of lights.

Figure 5.6 lists the first seven two dimensional Euclidean space polygons which can be used to geometrically interpret the first seven light configurations with the each geometric representation having the same number of vertices as lights. Our method of Symmetric Lighting can be applied to an infinite number of lights in theory, which essentially corresponds to an infinite vertex polygon or a circle. But in reality, only a finite number of lights will be used.

Given the relationship described by Equation 5.10, we can rewrite the system of equations to be in an inhomogeneous matrix representation of the form $Ax = b$.

$$L = \begin{bmatrix} l_1 & \cdots & l_n \\ \vdots & \ddots & \vdots \\ l_n & \cdots & l_i \end{bmatrix}, \quad B = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}, \quad C = \begin{bmatrix} C_{i_1} \\ \vdots \\ C_{i_n} \end{bmatrix} \quad (5.11)$$

$$R \begin{pmatrix} L \\ B \\ C \end{pmatrix} \times \begin{pmatrix} L \\ B \\ C \end{pmatrix} = \begin{pmatrix} L \\ B \\ C \end{pmatrix} \implies RLB = C$$

Where L is the $n \times n$ symmetric lighting matrix of coefficients, B is a $1 \times n$ matrix of variable proportions, and C is a $1 \times n$ matrix of pixel values for each equation. For all equations corresponding to a single pixel location i with n -lights, R is a single value. This can be interpreted as the value for R simply being a scale value for the coefficient matrix L , which does not change the proportions of light. So since we are not explicitly solving for R , we can essentially set R to the value 1 and remove it from the matrix multiplication. The result is a system of linear equations in the $Ax = b$ form that has a symmetric positive definite coefficient matrix.

Given that we have a matrix representation of our light capture method stated in Equation 5.11 as a linear system of equations, this allows us to choose from many matrix solvers that can provide solutions to our equations. But since we know that our coefficient matrix is symmetric and given the geometric nature of our lighting setup illustrated in Figures 5.6 and 5.5 we can further restrict the search space for our solution and provide a faster solution. Because the geometry of our lighting corresponds to a regular 2-dimensional polygon with matching light numbers and vertices, our solution set is in the subspace spanned by the area of the polygon. Furthermore, our solution is the Barycentric coordinates of the polygon corresponding to an interior point of the polygon. For example, assume that our lighting setup was the same as Figure 5.5, with three lights corresponding to a polygon of the named triangle. Then the solution is the Barycentric coordinates of the triangle that uniquely satisfies $LB = C$, where the coordinates are the proportions $\beta_1, \beta_2, \beta_3$ corresponding to ratios between lights one, two, and three. This implies that a good lighting setup would be one where the Barycentric coordinates have the same span (length) and are not too close together to as to cause the polygon shape to degenerate (e.g., go from triangle to line segment, without reducing the number of vertices).

5.2.3.2 Degenerate Cases and Computational Complexity

Degenerate forms: A failure case in terms of our lighting capture method described previously is any situation where a unique solution for $Ax = b$ cannot be determined, which occurs when either no solution exists or more than one solution exists. This may happen when the color values for l_i and l_j are too close together thereby making the solution set for $LB = C$ to be not unique.

A geometric interpretation of this situation is one that causes a polygon of vertex number n to have a degenerate form of a lesser class of polygon (i.e. a class of polygon where the number of vertices is $< n$). For example, assume we have a light configuration that has three lights and therefore can be modeled as a triangle polygon as illustrated in Figure 5.7. When the light color l_i of the setup has the same value l_{i+1} then the geometry changes from a triangle to a line without changing the number of variables. This is referred to as the degenerate form of the polygon. Referring to Figure 5.6, every polygon with n vertices is the degenerate form of polygons with greater than n vertices. For our previous example, the line segment is a degenerate form of a triangle. In the context of our lighting setup, degeneracy changes the solution set from being unique to one that has an infinite number of solutions (i.e., every point collinear to the overlapping lines).

Another way to think about the degenerate case is that we have the same number of equations but one pair of these equations that are exactly the same. This essentially gives us two equations with three lights and three variables. Therefore the we have a system of equations that becomes underdetermined and has an infinite solution set over $[0 - 1]$, as every proportion is a solution to this equation. As a final note with respect to lighting colors being unique, it is often the case that many lighting configurations have light colors that are the same. To use our Symmetric Lighting method each light that is to be controlled separately must have a unique

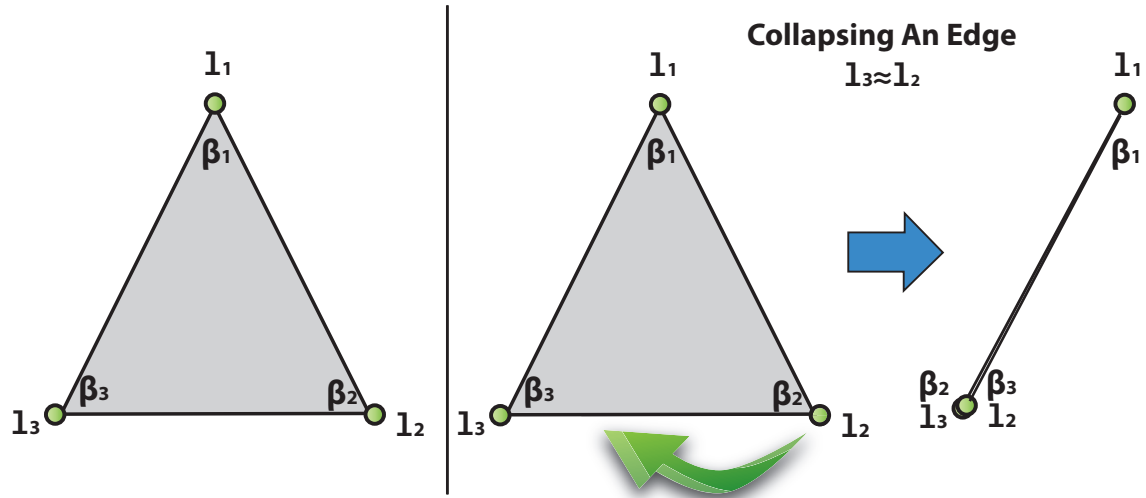


Figure 5.7: The degenerate form of a triangle is a line segment. All polygons of n -vertices are degenerate forms of polygons of $n+1$ vertices.

color. So a lighting setup that uses our method will need to be designed with each light having a unique color. Much like, when using a method for performing Alpha Matting (Chroma-keying), a green or blue screen must be included in the setup of the scene. This may be perceived as a challenging aspect to using our method, but there are several ways such a system can be configured that allow for flexible implementation. We describe more detail on how the system can be implemented in Chapter 6.

Complexity of solving $Ax=b$: There are many ways to solve a system of linear equations such as with Gaussian Elimination [TB97] or Cramer's rule [TB97]. Because we have a coefficient matrix that is symmetric we can further decrease our complexity by using a method that exploits symmetry. The LU decomposition algorithm called developed by Alan Turing [Poo10] is such an algorithm. This method takes the coefficient matrix and decomposes it into a two matrices L and U , where L is the lower triangular portion of the coefficient matrix and U is the upper triangular

part. To further reduce the complexity of solving the Symmetric Lighting, we take into consideration that we have a matrix that is also positive-definite. This allows us to utilize a faster method of LU decomposition called Cholesky decomposition [TB97]. Like LU decomposition, the matrix A is decomposed into its upper and lower triangles, but since we have a positive-definite matrix we can take advantage of that by computing L and L^T , where L^T is the transpose of the L triangular matrix.

Cholesky decomposition method has a complexity on the order of $O(n^3)$ in terms of the size of the matrix A [TB97] or in our case the number of lights. This is about half of the running time of LU decomposition. As far as decomposition methods, the Cholesky method is known to be the most efficient method [JO06] for matrix decomposition and can be implemented on the GPU efficiently [JO06]. Also, since the decomposition is in terms of coefficient matrix and in our case the lighting matrix which is known ahead of time, we can compute the Cholesky method as a pre-process before rendering and then substituting its results back to solve the linear equations. Back substitution by itself has a known complexity on the order of $O(n)$ in the number of equations [Pre92].

Given the low number of lights in a configuration, the running time listed here is entirely acceptable as $O(n^3)$ of a lighting configuration with < 10 lights, which can be executed on a GPU in real time. In reality, if we preprocess the lighting matrix, then our run time complexity reduces to $O(n)$. Below, in Listing 5.1, is pseudocode for performing Cholesky decomposition adapted from [JO06].

Listing 5.1: Cholesky Decomposition pseudocode adapted from [JO06]

```

for i=1 to n-1 do
    A(i,i) = sqrt(A(i,i))           %Square root
    A(i+1:n,i) = A(i+1:n,i)/A(i,i) %Normalize
    for j = i+1 to n do             %Inner product subtract
        A(j,i+1) = A(j,i+1)-cross(trans(A(j,1:i)),A(i+1,1:i))
    end for
end for
A(n,n) = sqrt(A(n,n))

```

5.2.3.3 Error Estimation

If we again consider the case with a scene with only two lights, then the camera pixels $C_{1i,j}$ and $C_{2i,j}$ are the products of the lighting L_{T12} and L_{T21} and the reflectance R , it follows that our error would be bound by the lighting and reflectance. Of course this is only true if we have an ideal camera, but assuming that any camera errors manifest themselves in the estimates of L and R variables and also assume that the camera has been calibrated and that any noise process related to the camera has been modeled (i.e., subtraction of a dark image). When capturing the color values C_1 and C_2 , we are measuring the product of L and R . Therefore, we can estimate the lighting error u and reflectance error v (see Equations 5.13) as being due to errors in measurement ϵ_i and ϵ_r for each respectively and the total error E as the product of L and R and their associated errors.

$$u = L(1 + \epsilon_i) \quad (5.12)$$

$$v = R(1 + \epsilon_r) \quad (5.13)$$

As previously mentioned the reflection function is implicitly modeled and not

explicitly known in the Symmetric Lighting capture. This is one of the benefits of this technique, but with scenes that violate the Lambertian reflection assumption can cause errors associated with solving the simultaneous equations from Equation 5.6. Because the reflection function is unknown, we have to account for how it varies with the lighting orientation in the presence of specular reflections. Therefore we suggest that this will be the primary cause of error associated with the minimization of the objective function for finding a β based on diffuse reflection. Additional errors will result from measurement of the light sources. If we assume both errors are relative to the measurement or approximated value of L and R respectively, then we can represent their relative linear relationship and estimate total error as the multiplication of the error estimates. In Equation 5.14 we substitute back in the values from Equation 5.13, which provides a model of the total error.

$$\begin{aligned}
E &= uv \\
E &= [L(1 + \epsilon_i)][R(1 + \epsilon_r)] \\
E &= LR(1 + \epsilon_r + \epsilon_i + \epsilon_r\epsilon_i)
\end{aligned} \tag{5.14}$$

Experiments for modeling the error were performed in this work using a synthetic scene rendered in a physically-based rendering system called V-Ray (version 1.5RC4). Using a synthetic scene and a physically-based rendering system gave us the benefit of knowing the lighting exactly, utilizing an ideal camera, thus eliminating any error associated with estimating L and C. Also, we eliminate Global Illumination from the tests, thus eliminating any noise from estimates of the light transport and an exact solution can be generated with direct lighting only. Therefore, any error in the synthetic render will be the result of the unknown reflection function, $E = LR(1 + \epsilon_r)$. Figure 5.8 shows the error associated with Symmetric Lighting capture using a specular reflectance function compared to the ground truth, which uses a

Lambertian reflectance function. The Absolute Difference between the calculated β maps of the two renderings is shown in the right column, with the differences of the specular version multiplied by ten for visualization purposes. Additional evaluations with respect to error are provided in Chapter 6.

The diffuse model provides a uniform distribution of reflection and is therefore not dependent on the direction of the incoming light. This reflection function is the basis for Equation 5.6 since it can be factored out from the light contribution estimation (i.e., estimating β). The β map from the Lambertian reflection experiment will be used to calculate the L_2 norm.

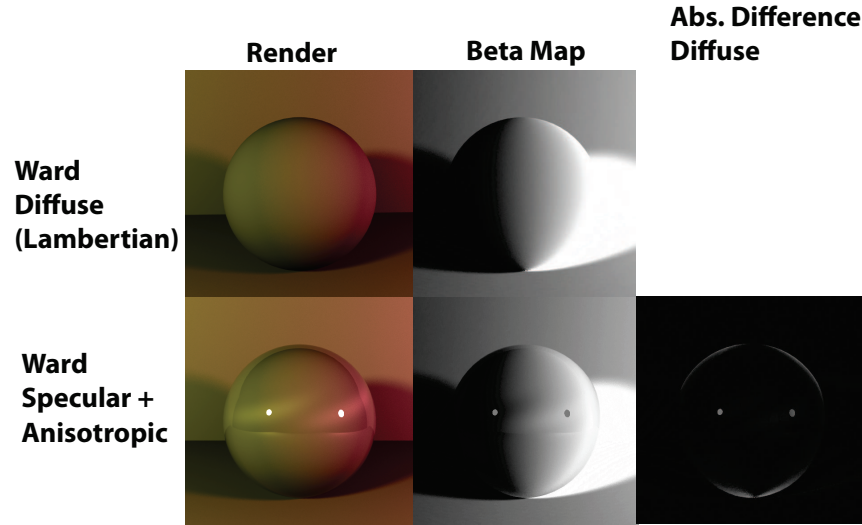


Figure 5.8: Error estimate due to specular reflection. The β maps (center column) for Lambertian (top row) and specular reflection (bottom row) are compared by calculating the absolute difference image between the β map of the non-Lambertian reflectance and the β map of the Lambertian reflectance (in this case the Ward Diffuse model). Each scene has the exact same setup, with only the reflectance of the geometry modified being different. Errors manifest themselves as differences in the β shown in the right column.

In terms of a two light system, we can define the absolute error to be the deviation of the pixel's β value from the β value for the same pixel of the ground truth. This is visually illustrated in Figure 5.4 as the minimum distance between the two line

segments $\overline{C_1l_1l_2}$ and $\overline{C_2l_1l_2}$. Therefore the error from specular reflection is bounded by $Max(d_{rgb}^2)$ is the maximum of the closest distance between the line segments $\overline{C_1l_1l_2}$ and $\overline{C_2l_1l_2}$. In other words, the domain of the solution to the system of equations for Symmetric Lighting is in the RGB color cube, the maximum error is bounded by the size of the cube, which is unity in all dimensions. In Subsection 5.2.3.4 we utilize this idea for extending the reflectance assumption for Symmetric Lighting to include specular reflections. The error associated with measuring L, will be dealt with in the next chapter.

$$E \leq \sqrt{Max(d_r^2) + Max(d_g^2) + Max(d_b^2)} \quad (5.15)$$

5.2.3.4 Minimizing Epsilon for Non-diffuse Reflections

For materials that exhibit typical and non-conducting behavior, a linear model for reflection is commonly assumed [WLL⁺09, DHT⁺00a, LPD07, MHP⁺07]. Specifically, the linear modeled for reflection is expressed as a sum of the reflectance terms, which generally includes separate diffuse and specular reflectance. To show that the Symmetric Lighting model can be extended to include non-diffuse reflection, in particular specular reflection, we utilize the previously mentioned linear model. We have already shown that the Symmetric Lighting formulation from Equation 5.7 assumes a Lambertian reflection, and a Lambertian model for the diffuse is understood to accurately represent reality with non-spatially varying functions [WLL⁺09], then an additional specular term is all that is required to extend our formulation.

First we assume that separating the diffuse and specular properties of a reflectance function is feasible not just in theory, but in practice. Indeed, several academic works have shown the plausibility of separating the diffuse and specular parts of an object's reflection function with relative ease either by color separation or using

polarized light and polarized filters. [NFB97, LPD07, DHT⁺00b, MHP⁺07, KSK92]. For the specular portion of the reflectance function R , we consider only the view-dependent points seen by the camera and not attempt to acquire the function as a whole. By eliminating the specularly from the camera images, we have only the diffuse portion remaining. We can then solve for the diffuse term as previously done using the Lambertian assumption. Then the β_i for each camera pixel C_i that was solved for the diffuse term is also used as the β value for the specular term. This is true because the β represents the proportion of each light source contributing to the total illumination at that point, and this value is independent of reflection function. Therefore, solving for the specular term can be done in the same fashion as solving for β through simultaneous equations.

$$C = C_{diff} + C_{spec} \quad (5.16)$$

$$C_{spec} = R_{s1}L_1\beta + R_{s2}L_2(1 - \beta) \quad (5.17)$$

Were C_{diff} is the result of Equation 5.6. Given two values for the C_{spec} from the separated terms and two images, we have two equations (one for each image) and two unknowns, R_{s1} and R_{s2} . Substituting the equations and solving for R_{s1} and R_{s2} gives enough information to utilize the specular in conjunction with the diffuse reflection without the need to approximate the specular reflection further. This idea is illustrated in Figure 5.9.

Another factor that may contribute to the error is Global Illumination (GI). In this research we assume that the error is not significant enough to warrant investigation at this time and is reserved for future work. This is mainly due to the assumption that for our main application of relighting for the purposes of Previsu-



Figure 5.9: Separation of diffuse (left) and specular (right) reflection for Symmetric Lighting.

alization, which can tolerate low amounts of error. Although, if it were determined to be a significant source of error, GI can also be removed or estimated using a similar method to that of separating diffuse and specular reflection using the technique proposed in [NKGR06].

5.3 The Beta Map

In this section we describe the main data structure developed for storing the results of our Symmetric Lighting method. This novel data structure, which we call the β map (Beta map), stores the normalized proportion of each light’s contribution to each camera pixel independent of reflection. The β map is analogous to the alpha, normal, and depth maps for a scene, in that it stores per-pixel information about the scene that, in our case this information is related to lighting evaluated from Equation 5.7. Additionally, like that of its analogues, the β map is image-based and stores the lighting and partial geometry data corresponding to pixel location of the camera image. In essence the β map stores pre-multiplied shading values, or the values of the light multiplied by the geometric information of the scene prior to it being multiplied by the reflectance and appearance values. In terms of the Rendering equation from Figure 5.10, this is the value of each light times visibility and the orientation of the surface normal and lighting direction. In the context of multi-pass rendering, such as done when using software such as Maya or 3D Studio Max, this is known as Light or Lighting pass [Bir06] and prior to this work was only available in synthetic rendering contexts and not with real scenes.

An important property of the β map is that the process of capturing the lighting information through Symmetric Lighting, we have extracted the reflectance and appearance information $R()$ from the camera image. As can be seen in the Figure 5.10 below, the β map consists of only a combination of scene lighting and geometry information. Another way to think about the β map is that it is a ”reduced complexity” version of the camera image (i.e., no reflection or appearance information) that makes inferring information about the lighting and geometry easier. Taking an image-based approach has the benefit of not requiring a transformation to map

$$C_i(x, \omega) = \int_{\Omega} \underbrace{R(x, \omega', \omega)}_{\text{Reflectance}} \underbrace{L_i(x, \omega') \underbrace{V(x)(-\omega' \cdot n)}_{\text{Geometric Orientation}}}_{\text{Lights}} d\omega$$

β -Map

Measure Over Hemisphere Ω

Figure 5.10: Here we present a modified version of the Rendering Equation introduced by Kajiya [Kaj86], where a camera pixel C_i is the result of multiplying all lighting contributions L_i and reflection R with respect to the scene geometry over a hemisphere. Symmetric Lighting allows us to separate the product of the reflectance and lighting and geometry, which we store the result in a data structure called the β map. Additionally, the β map records the proportion each light contributes at each pixel within the camera image providing the capability to distinguish each lights contributions.

the coordinates of one representation to another, unlike that of an atlas-based map, which allows for less complicated processing and interpretation of the maps. In that spirit, we have extended Equation 5.3 to include a visibility term $V()$ as indicated in Figure 5.10, which when interpreted in the context of the β map, can be used to determine which lights are visible at every pixel location in the camera image. Through the use of simple image processing techniques performed only on the β map, we can determine the visibility of a point located within the scene without the need for explicit geometry, normal, or depth information. In Ray Tracing, visibility is the main component in determining whether a point in the scene is in shadow. In Subsection 5.3.1 we describe a threshold watershed method for determining the location of shadows in the camera image individually for each light source.

If we consider most lights sources to follow some level of falloff as a result of distance or shape, then approximating the shape of the light can be considered a flow field as it reflects off a surfaces within the scene. Specifically, the gradients (first order) and divergence (second order) of the β map values (zeroth order) indicate the direction and magnitude of the flow of photons in the scene, which we use to gauge

the geometric information and lighting direction. We then utilize this information to identify how the flow of photons are influenced by the scene geometry, which allows us to decouple the geometry from the lighting within the β map to provide an approximate light distribution. This method, which we call *Light Sinks*, can be seen as an inversion of irradiance (photons after reflecting off a surface) to radiance (photons as emitted from source prior to reflecting off a surface) lighting values. As described in more detail in Section 5.3.2, this inverse process approximately restores the lighting values to a point in time right before photons reflect off the surfaces of the geometry in the scene, which can provide direct light editing.

In terms of the β map representation, grayscale maps as seen in Figure 5.10 are sufficient for maps that contain lighting from only two lights as their proportions are easily calculated. When using more than two lights, the map can be trivially extended to utilize the color channels of the maps images instead of grayscales across all channels. For example, if we had three lights having values $\beta_1, \beta_2, \text{ and } \beta_3 | \beta_1 + \beta_2 + \beta_3 = 1$, then each value for each pixel could be assigned to each color channel of an RGB image. Additionally, the alpha channel could be used for a fourth light, although visualizing such results might be unintuitive as alpha channel is typical reserved for opacity and any imaging program that support opacity could misinterpret such values. So, we prefer to use separate maps for numbers of light > 3 , instead of using the alpha channel.

Because we specify the β map in the form of an image, we can utilize this representation to take advantage of texture hardware on graphic processing units (GPUs) when performing relighting. In essence, the formulation actually becomes simplified as a result, since looping through all the pixels is no longer required as all relighting calculations are performed in parallel using shader programs on the GPU.

Using the formulation for Symmetric Lighting for two lights we can generate a β

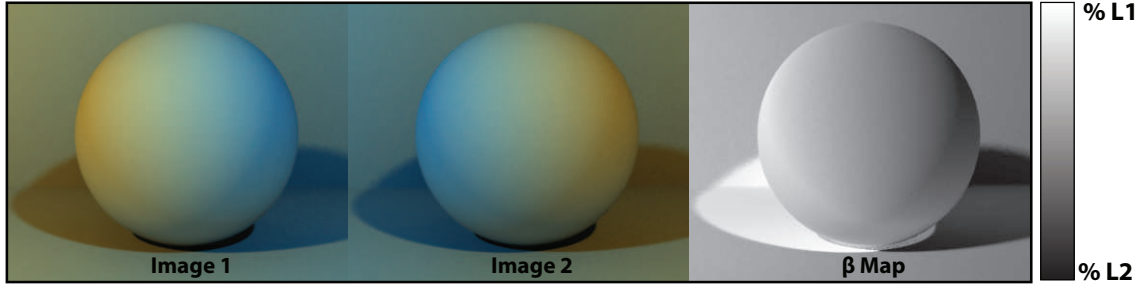


Figure 5.11: Beta Map Creation: input two images containing the same setup with the light colors interchanged resulting in a grayscale image where the pixel values correspond to the proportion of light each source provides for a particular pixel.

map that can be used to factor out the lighting information leaving only the values for $R()$, which is known as the reflectance image. This process can be considered a form of intrinsic images [BT78, Fer06, ED04], where a fully rendered image is decomposed into separate images containing only lighting (i.e., light pass), reflectance, and appearance data. Then relighting simply becomes the generation of a new lighting image, which is achieved by multiplying the β map by the new lighting values. Furthermore, a new camera image is generated by multiplying the reflectance albedo by the new lighting. This process is described in more detail in the Previsualization and Evaluation Section 6 and sample programming code is provided in the Appendix Section B. Figure 5.12 provides a visual illustration of this process.

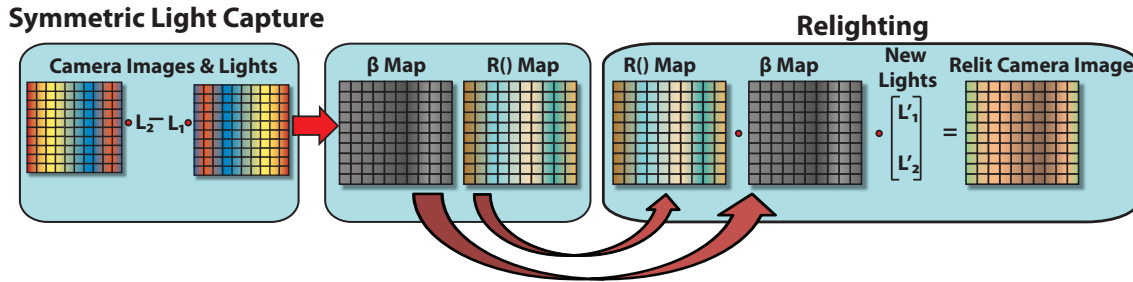


Figure 5.12: A simplified relighting example performed as a series of texture map operations as executed on graphics hardware. Left side is the Symmetric Lighting light capture, which results in two maps (center). Right shows relighting as the product of the new lights L' , β map, and the $R()$ map to produce a relit camera image.

In addition to relighting, we believe that the β map is a powerful data structure that can be purposed for many other tasks. For example in Section 5.3.1, we show how it can be used to solve multi-shadow detection. Additionally, we show how it can be used to solve multi-illuminant white balance in Section 6.4, and light color estimation in Section 6.3. Beyond the methods described in this work, there are several other problems which we plan to explore in future work that we could utilize the β map to solve. The following is a list of some of these potential methods:

- *Surface Normal Estimation* : In Subsection 5.3.2 we describe in more detail our idea of *Light Sinks*. It is possible, but not fully explored in this work, to estimate the orientation of the lights or the light normal using this method. Using the β map, and light normal, using the simplified formula for the Rendering equation which describes an attenuated cosine factor $N \cdot L$, we can solve for the surface normal N using a least-squares method.
- *Depth Estimation* : Physical lights exhibit a decrease in intensity with distance or fall-off called the Inverse-Square Law. With β maps, it could be possible to estimate the distance from the lights by tracking the β values that exhibit an inverse-square increase or decrease in value.
- *Global Illumination (GI) Estimation* : As stated previously Symmetric Lighting assumes a Lambertian reflectance model and therefore the presence of GI will manifest itself as error. If we develop a method for measuring the error associated with GI, this will be equivalent to measuring GI.
- *BRDF Estimation* : Given the camera model for Symmetric Lighting in Equation 5.1, the reflectance R is implicitly known and substituted out to solve for β . Once the value for β is found, it is straightforward to substitute R back in

and solve for R . Although our value for R contains reflectance and appearance information, it may be possible to further separate these values.

5.3.1 Shadow Detection Within The Beta Map

In the context of the beta map, we show a new way to detect shadows in an image with more than one light that is independent of shape, albedo, or reflectance of the object. Because most shadow detection algorithms assume a single light source, we provide an alternate definition of a shadow as a location that is illuminated by $n - 1$ or less lights, with the assumption of n lights within a scene. In other words, a point in the scene is in shadow if there is no direct illumination from one or more lights within the scene. Leveraging the information available with the β map makes searching for and detecting shadows straight forward, and essentially involves finding pixels with β values below some threshold.

Because we can separate the proportions of contributing lights into different color channels of the β map, searching for shadows associated with a particular light is trivial. In this case a simple thresholding of the β values in a particular channel will produce an estimate of the shadows present in the scene by segmenting the β map into two distinct regions. We can precisely define this as:

$$\beta_{st}\{x \in R^2 | f(x) = \beta, \beta \leq st\} \quad (5.18)$$

Every pixel location x in the β map $f()$ has a β value. Then if st is the shadow threshold, then β_{st} are the values of the β map that are equal to or below the threshold st . Conversely every $\beta > st$ is above the threshold. Coarse thresholding such as this can detect both self-shadows as well as cast shadows as seen in Figure 5.13.

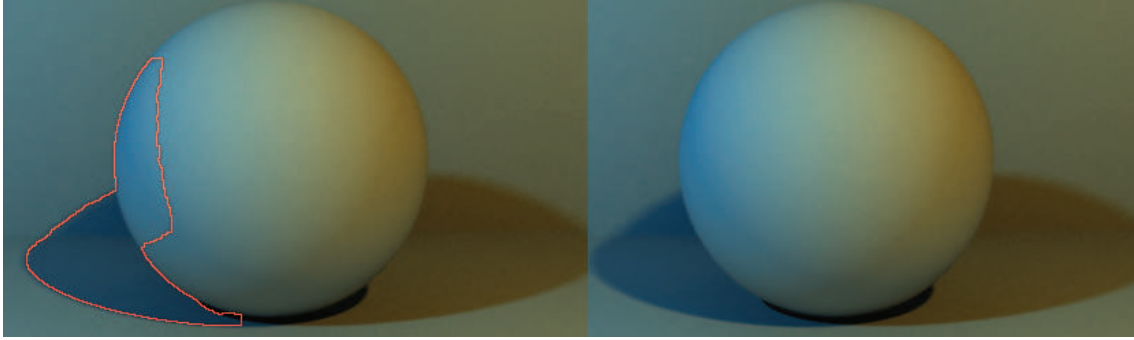


Figure 5.13: Example shadow detection using threshold values and a β map of the scene. The image on the left shows our ability to detect both cast and self-shadows from a single light source. The image on the right is for comparison.

Shadows can exist with different levels of intensities, especially when shadows from different lights overlap. In the context of cast shadows, the Umbra is a shadow component that is entirely blocked and therefore has low intensity values. The penumbra is only partially blocked and has higher intensity values than the Umbra but is still low relative to the rest of the scene [JW92]. In the presence of multiple light sources (or area lights), umbra generated from one light can have the intensity increased due to light coming from other sources changing it to a penumbra in the camera image.

To be able to capture different levels of intensities or umbra and penumbra we use the watershed transformation technique [BL79] based on β values within the β map. The watershed technique uses regional minima to establish seed locations for a "catchment basin" or areas where water would collect in terms of topology. In terms of shadow detection, these catchment basins represent areas of low β values representing shadows. The catchment basins are separated by higher areas called "watersheds" which in terms of topology force water to roll down and collect water in the catchment basin, like a ridge in a canyon. The watershed technique, which is widely used in image segmentation, identifies the catchment basin and watershed areas in order to demarcate regions of interest. For shadow detection, watersheds

represent the delineation of the shadow areas from other high intensity areas. If we assume the β map represents a grayscale topographic map, with high β values being areas with high elevations or watersheds and low values being catchment basins, then shadows will exist in the catchment basins.

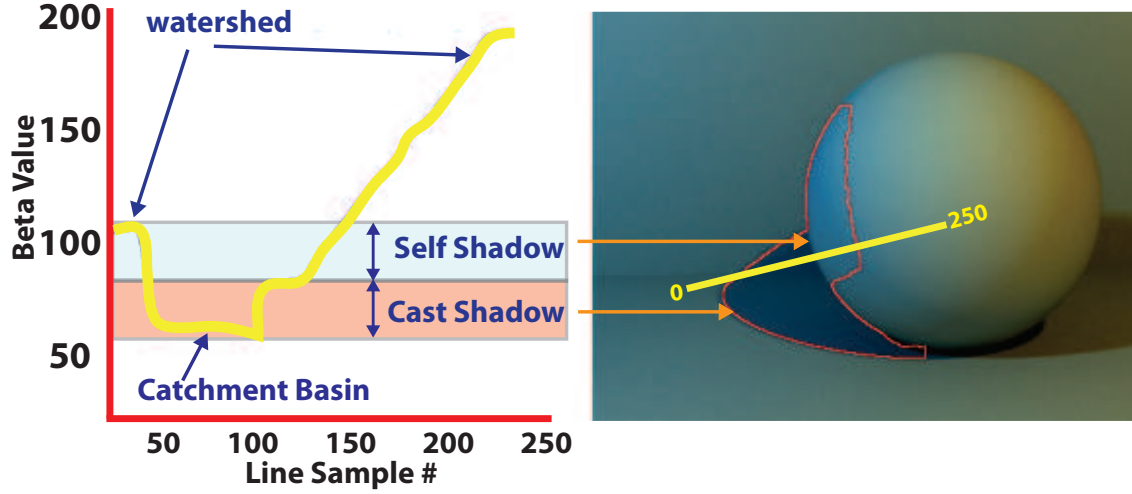


Figure 5.14: Example shadow detection using threshold values and a β map of the scene. The image on the left shows our ability to detect both cast and self-shadows from an individual single light source instead of grouping all shadows together, which is how previous shadow detection methods work. The image on the right is for comparison.

To use this technique, we let W be the set of all watersheds and W_m be a subset of W with minimum height m . Then we can define W as the following:

$$W = \bigcup_m W_m \quad (5.19)$$

To determine all W_m we need to determine all the points whose height is less than m $\{x \in X | x < m\}$ that belong to only one catchment basin. Also, if a point x has a value equal to m and it is the same distance from two catchment basins, then it belongs to W_m and is therefore a watershed. Figure 5.14 illustrates these definitions.

Our technique allows for separation of the umbra and penumbra for each light making it possible to see each light’s shadow individually. We do this by using the watershed method with windowed thresholds. By using multiple thresholds in conjunction with the watershed algorithm we can identify regions of different β levels (different catchment basins). This would be similar to the shadow detection illustrated in Figure 5.14 but with umbra and penumbra instead cast and self-shadows corresponding to umbrae and penumbrae of different shadows.

5.3.2 Gradient Domain Light Distribution

This section describes a novel technique for manipulating the individual lights within a scene by editing their photon distributions in images space. Small changes in lighting can have a dramatic effect on the how the image is rendered. In reality, lighting on a movie set is often moved, turned, and adjusted many times before and during a shot. Analogously in Relighting applications, users want to edit the parameters of the virtual or simulated lights in many of the same ways that they could physically and then reapply the new light settings to the scene. This includes the ability to reshape, translate, or rotate the lights. Combining virtual lights (i.e., lights that do not exist physically in the real scene) with fully captured scene information (i.e., geometry, reflectance, and appearance) is certainly feasible. Another technique used for Relighting applications is employing captured real-world lighting such as Environment Maps [HS98, FLW02] or Light Probes [Deb98a]. But these techniques assume the user considers the captured lighting as the desired lighting environment and at best only provides basic editing capabilities such as rotating the environment maps. Sampling techniques can be used to generate individual points on the maps that may act as individual lights. These are generally used to improve the rendering and are usually considered distant and therefore have a uniform distribution of

photons throughout the scene, and are not optimized for editing.

The β map from the previous section describes the proportion of the total illumination that each light contributes to a point in the scene for each pixel value in the map. Looking at the β map at different scales, reveals that it is also provides higher-ordered information about the proportion of light, which conveys the shape and distribution of light radiated. Because the β map also encodes geometric information regarding the relationship between the lights and the scene geometry, this gives the β the overall appearance of the geometric shape of the scene, which includes depth edges, visibility, and the cosine term relating the surface normal to the orientation of the lights. This section describes the method used to untangle the intrinsic geometric information from the illumination to create an illuminance image [BT78].

If we consider neighborhoods of pixels, then changes in proportions of light within the neighborhood can be represented as the gradient of the β values. The gradient represents the partial derivative of the lighting function in the spatial domain, thus provides first order information. We can then extrapolate this idea further to a larger scale which can encompass many pixel neighborhoods and greater areas of lighting influence. This higher-ordered information describes the distributions of photons within these local regions, which we can describe using the divergence operator. The divergence operator is the derivative of the gradient operator, and thus provides second order information about the lighting function. This operator provides some notion of the shape of the light relative to the scene. Therefore, we developed a technique for changing the shape of individual lights by altering the gradients and divergence of β map associated with the lighting without the need to physically alter the lighting system or being restricted to distant light sources.

The work presented here is in the same spirit of Fattal et al. [FLW02] where they

too consider lighting and the gradients associated with the lighting. The difference between their work and ours is that they are primarily focused on the dynamic range compression of the luminance values within a scene. We instead are similarly interested in luminance, but not necessarily the dynamic range but other properties such as the distribution of photons and the placement of those photons within a scene. Their operation, dynamic range compression, is performed by attenuating the magnitude of the gradients in areas that exhibit the high-end of the range and scaling up those in the low-end of the range slightly. Finally, this work is also draws from the ideas of Agrawal et al. [ARC06], who employed the use gradient domain operations to suppress depth edges, shadows, and spurious light. In their work, they observed that under variable lighting, an operator could be devised to suppress parts of the gradient-field in the target image using the gradients from another image of the same scene under different lighting. Our goal in using gradient domain operations differs from Agrawal et al., in that instead of suppressing the gradients of an image that correspond to shadows or spurious lighting, we have focused on suppressing the gradient edges of the geometric scene information and separating it from the lighting. This has allowed us to develop operators that provide the capability to refine the lighting to the user’s specification, such as changing shape, intensity, and location.

The main contributions of this section are:

- Novel Gradient domain approach to manipulating the illuminance of individual light sources independently to modify the lighting within an image.
- Simple but robust discriminator, based on separation of components and gradient histograms.
- Novel data structure for identifying areas of varying influences of a light and,

in turn adding more information to the β map.

5.3.2.1 Method

There are two motivating factors for using the gradients of the β map for editing the distribution of photons in a scene. First, the Human Visual System (HVS) is sensitive to local contrasts rather than absolute luminance values[RBH08]. It is known that image gradients are correlated with contrast differences [BZCC10] and larger differences in luminance values result in larger gradients. The second motivation for performing the gradient domain operations in the illuminance images is that performing gradient editing in the composite image ($R*L$) does not just edit the lighting gradient, but edits the reflectance gradient as well. More precisely, it will edit the gradient of $R*L$ including gradients that result from the appearance (i.e., texture) and geometry of the object in the scene. In order to precisely edit just the lighting, we need to discriminate between the lighting and the other properties of the scene. As suggested in [ARNL05, ARC06], the orientation of the image gradient remains stable under variable illumination, since a gradient is just a vector with magnitude and direction, changes in illumination must serve to scale the magnitude of the gradient. Therefore if we proceeded to edit the image gradients indiscriminately with some operation that performs an orientation change to the vector, then artifacts related to the modified gradients will become apparent.

The gradient of the β map from Section 5.2 is defined as:

$$\nabla\beta(x, y) = \frac{\partial\beta_x}{\partial x} + \frac{\partial\beta_y}{\partial y} \quad (5.20)$$

and $\frac{\partial\beta_{x,y}}{\partial x,y}$ is the partial derivative of the β map with respect to the spatial directions x and y . This can be interpreted as the change in proportion of the lighting of L_1 and L_2 , where the direction of $\nabla\beta(x, y)$ indicates the direction of the greatest

increase in proportion.

In particular, we find local maxima within the scenes gradient field using a divergence operator, which measures the gradient field's tendency toward a sink (negative divergence) or source (positive divergence). Since we are interested in areas that exhibit a "collecting of photons", we consider only areas of negative divergence, which we call *Light Sinks*.

$$\nabla \cdot \nabla \beta = \frac{\partial \nabla \beta}{\partial x} + \frac{\partial \nabla \beta}{\partial y} \leq 0 \quad (5.21)$$

In Equation 5.21 we define the gradient field β over the dimensions of the image $\beta(x, y)$, then the divergence of that field is defined over the partial derivative of the x direction $\partial \nabla \beta$ and the y direction $\partial \nabla \beta$ with respect to partial derivative of the x and y direction. The light sink then has a maximum divergence when the $\nabla \cdot \beta$ has a maximum negative value. Since we assume that each light will produce *at least one Light Sink*, we can cluster these values in the RGB cube to estimate the value of the light colors.

We then perform our gradient operations in the 2D spatial domain. Our operations, which we will define in the next subsection is applied directly to the gradients of the scalar field of the β map. It is important to keep in mind that, instead of a color image with three tri-stimulus values, we are working in a light proportion space defined by the β map creation process which has a single value for two lights or two values for three lights. This helps us to avoid any color artifacts that may result in the operation or the reconstruction of the β map.

$$\tilde{F}(x, y) = \nabla \beta(x, y) \phi(x, y) \quad (5.22)$$

Where $\tilde{F}(x, y)$ is the gradient field resulting from the operation of $\phi(x, y)$ and

the gradient of the β map. If $\tilde{F} = \nabla\tilde{\beta}$ in our newly modified β map, then $\nabla\tilde{\beta}$ must satisfy:

$$\frac{\partial^2\tilde{\beta}}{\partial x\partial y} = \frac{\partial^2\tilde{\beta}}{\partial y\partial x} \quad (5.23)$$

Because \tilde{F} has been shown to almost always not be integratable, then we can assume that $\tilde{\beta}$ will not satisfy Equation 5.23 and utilize other means for determining $\tilde{\beta}$. Mainly

$$\min \left\| \nabla\tilde{\beta} - \tilde{F} \right\|^2 \quad (5.24)$$

Which searches the space of all gradient fields to find some $\tilde{\beta}$ that has the minimal difference, in the least squares sense, between itself of the calculated gradients F from the operation in Equation 5.22. The new $\tilde{\beta}$ should satisfy the Euler-Lagrange equation allowing us to obtain the Poisson equation from the new $\tilde{\beta}$.

$$\nabla^2\tilde{\beta} = \nabla\tilde{F} \quad (5.25)$$

Where ∇^2 is the well-known *Laplacian operator* and $\nabla\tilde{F}$ is the divergence of the gradient field from Equation 5.21 without the condition of being ≤ 0 .

Edge Suppression : As previously mentioned, the β map has encoded geometric information about the scene from which it was created as well as relative contributions from the lights. Our goal is to untangle the geometric information from the lighting in order to generate an Illuminance map, which we could use to manipulate the spatial properties of the light such as location and shape. The β map contains geometric information about the scene such as depth edges (discontinuities), shadows, and the relative orientation between the surface normals within the scene and the light direction which manifest themselves as gradients as illustrated in Figure

5.15. The gradient edges caused by shadow boundaries and depth discontinuities, have large gradients relative to the other gradients within the map. Removing these gradients from the map would constitute a significant portion of the geometric information contained with the map.

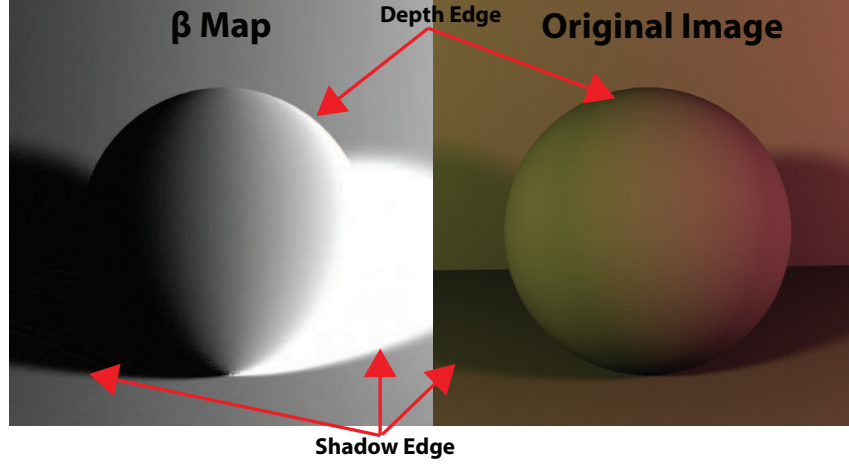


Figure 5.15: Large gradients corresponding to edges indicated by the red arrows.

We utilize the technique developed by Agrawal et al. [ARC06] for suppressing large gradients associated with these geometric features. Edge detection filter could also be used to identify large gradients in order to build a mask for suppressing. Such techniques can leave small gradient edges, which in our case are undesirable due to assumption that the illumination is smoothly varying and any gradients that do not constitute a transformation from one light to the next can be considered a result of geometric or texture properties of the scene and not the illumination.

This approach builds what is referred to as the projection tensor of two images. The tensor can then be used as an operator to remove parts of the gradient-field in another image from the same scene under different lighting.

$$\nabla \tilde{\beta} = D^b \nabla \beta \quad (5.26)$$

Where $\nabla\tilde{\beta}$ is the β map with the geometry edges removed. The Projection Tensor D^b is generated from two Gaussian smoothed tensors G_σ with variance σ . The Eigen values and vectors of G_σ are used for constructing the Cross Projection Tensor D in a similar fashion to that of Diffusion Tensors [Wei97]. This approach removes edges in $\nabla\beta$ that also exist in the rendered version of the image C resulting from R*L, but retains the edges from $\nabla\beta$ that do not exist in ∇C . In other words, this can be thought of in terms of a set of operations and formed in tautologies, which could used as bitwise operations on gradient edges within images. So the Cross Projection Tensor is equivalent to $(\nabla C \oplus \nabla\beta) \wedge \nabla\beta$, using the truth table in Table 5.1:

Edges In:		Retained:
∇I	$\nabla\beta$	$\nabla\tilde{\beta}$
0	0	0
1	0	0
1	1	0
0	1	1

Table 5.1: Truth Table For Cross Projection Tensor Operations

While we do not go beyond this observation of the gradient edge operations as set operators, but it might be interesting and is thus left for future work. Readers familiar with Agrawal et al.’s work in [ARC06] may notice that a benefit to this technique over their previous work [ARNL05] is that it utilizes the information across the color channels for tri-stimulus images (RGB, CIE Lab, etc). The β map we describe as input contains only a single channel of information while the background image contains tri-stimulus values. We can mitigate this issue in one of two ways; first, it is noted that the Cross Projection Tensor can be utilized for gray scale images. We can therefore convert the background image to gray scale to match the β map as color is not the primary concern, but the edges of the gradients.

Or secondly, we can convert the β map to tri-stimulus values by smearing the single channel across the other two to make a 3-channel grayscale image.

In Figure 5.16 we present some results of the edge suppression.



Figure 5.16: Results of the edge suppression technique. Top two images are the separated β map with multiplied lights, L_1 & L_2 from Figure 5.15. Bottom row, left are the two lights together multiplied with β map. Bottom row, right, is the edge suppressed version of the combined β map.

5.3.2.2 Gradient Domain Operations

In this section we describe a series of operations that can be performed using the gradient-domain editing technique presented in this section.

Scale: The scale operation performs a scaling to the magnitude of the gradient vectors associated with the currently selected light. This has the effect of changing the apparent or perceived falloff or brightness of the associated light. This operation provides a scalar $C | C \in \mathbb{R}$ and is subject to the constraint that $\beta + C \leq 1$.

Rotate: The rotate operation describes the rotation of the gradient vectors associated with the gradient-field belonging to the selected light. Rotation of the

magnitude of the vector at a pixel will have the effect of reorienting the light by the rotation amount. The operation provides a scalar value $V|V \in R$ and the vector rotation value is constrained to $V \leq 2\pi$.

Translate: The translate operation describes the translation of the gradient from one pixel to another in the spatial domain (x,y). This has the effect of moving the light in the spatial direction specified by the values x and y. This operation provides two scalar values $x, y|x, y \in I$ and correspond to the number of pixel values in term of L_0 or the Manhattan distance for which the gradients is to be translated.

Diffuser: This operations describes the process of diffusing the light as though some uniform scattering medium was placed in the lights path. This gives the effect of creating a more diffuse and softer light with fewer areas of non-uniform lighting. This is done by averaging the gradient magnitudes within regions in order to remove high-frequency or noise related gradients variations.

Sharpen: This operations has the opposite effect as the Diffuser operation above. It serves to accentuate the differences in magnitude between gradients within a small region.

5.4 Conclusion

In conclusion, this chapter introduced our novel photometric and active illumination method for capturing the lighting of a scene, which we call Symmetric Lighting. This method works without explicitly knowing or measuring its geometry, reflectance, or appearance. In this chapter we developed an initial theory for a two-light scene and then generalized this theory to N-lights. We also provided an analysis of the computational complexity as well as efficient and hardware amenable ways to calculate each lights contribution based on the Symmetric Lighting by leveraging the symmetric nature of our technique. In addition to Symmetric Lighting, we also introduced several other contributions that resulted from our Symmetric Lighting capture method, such as an efficient and versatile data structure for storing the lighting contributions called the β map, novel shadow detection algorithms, and a gradient-based method for manipulating the distribution of photons for each light individually.

Our relighting method allows for previsualization and editing of on-set lighting in real-time, which cannot be accomplished using previous methods. This will enable directors and cinematographers to virtually preview and edit on-set lighting for greater artistic control and faster decisions on physical lighting changes, resulting in higher quality lighting in movies, fewer reshoots, and less on-set time. Although our method is less complex and costly to other relighting methods described in the Previous Work Section 6.2.5 in the next chapter, it is not without limitations. As previously described our method is based on the Lambertian reflectance model, which is a gross approximation to real world reflectance. Therefore our method, at this time, cannot be used to capture and manipulate more complex reflectance phenomena, such as subsurface scattering, transparent and emissive surfaces, and

Global Illumination. Specular reflectance can be captured, but at a cost of doubling the number of image acquisitions. This means that certain scenarios where our method might not excel is in capturing the lighting within scenes that contain humans skin or faces as they have been shown to require capturing sub-surface scattering. Additionally, scenes that contain no movement are ideal for the current version of our method; therefore scenes with fast moving objects or cameras will cause image blurring and produce large errors in terms of Symmetric Lighting.

Finally, a drawback of the β map is that the values represented at each pixel of the map contains the relative proportions of each lights contribution but not absolute amount of light. This becomes problematic when actual amount of light represented by a pixel in the β map is small. This can in techniques such as the shadow detection method described in this chapter when differentiating which pixel in the β map belongs to which shadow. This is due to the fact that shadows tend to have little or no light contribution from the light source. A method that represents the absolute value of the light contribution instead of the relative contribution may be beneficial in determining error with the β map as well as providing additional information GI contributions.

Chapter 6

Previzualization and Evaluations

6.1 Overview

In this chapter we describe the previs relighting application based on the Symmetric Lighting that was developed for this dissertation. As part this description, we explore the features of the relighting application as well as provide evaluations on the performance of the relighting technique. In Section 6.2 we describe the different relighting operations and how it can be used for different types of scenes, such as those with two and three-lights, High Dynamic Range lighting, and non-photorealistic scenes. In Section 6.3 we describe a separate application for estimating the color of unknown light sources in a novel way. Our light color estimation technique also leverages Symmetric Lighting theory and does not require expensive hardware or restrictive assumption used in previous techniques. The next technique developed for this dissertation is a novel multi-illuminant white-balance method, which is described in Section 6.4. In Section 6.5 we describe the three user studies we performed to validate the design and implementation of 1) our relighting method, 2) its potential viability in the movie industry as a virtual on-set relighting tool, and 3) expert

review of our programmable camera.

The main contributions in this chapter are :

- Novel View-dependent relighting method that can be utilized for real, physically-based synthetic, and Non-photorealistic scenes that uses a simple hardware setup, is fast, and provides many options for manipulating the lighting.
- Novel technique to solving the light color estimation problem
- Novel technique for calculating the white-balance of scenes that contain an arbitrary number light sources.
- Simple extensions to the Symmetric Lighting technique that can provide spatially-varying, near-field, High-dynamic Range features.

6.2 Scene Relighting

Relighting is the process of inferring the pre-existing lighting in a scene, undoing its effects, and applying new lighting. In a virtual scene, this can be achieved naïvely by editing the lighting and performing a full re-render of the scene. Since rendering a scene can be time-consuming, it is not practical to iteratively test minor changes in the scene. Therefore, more efficient methods for iteratively testing changes in scenes without full renders have been devised, which take advantage of a priori knowledge such as in virtual scene relighting which includes scene geometry, reflectance, and lighting information. Image-based relighting on the other hand, attempts to achieve the same goal of general relighting, but with the constraint of utilizing only image-based methods to estimate scene information.

If we consider the light traveling through a scene that is eventually sensed by some observer or camera, the lights path can be described by a function with seven

degrees of freedom called the Plenoptic [AB91] function. Furthermore, if you consider that the light that emitted that ray also has a similar function, then the total dimensionality of light radiated and irradiated can be considered a function of fourteen degrees of freedom (parameters of the Plenoptic function are described in more detail in the Appendix C). To fully capture this function with dense sampling is daunting, therefore most attempts to do so by ignoring certain dimensions, sparse sampling, or otherwise restricting the degrees of freedom. This idea of sampling the Plenoptic function gives way to applications such as Relighting, which aims to record some approximate form of this function in order to re-integrate this function virtually.

For relighting, view-dependence is a common way that lighting design software reduce the degrees of freedom, as lighting artists are often concerned with changing the lighting and not other parameters related to the camera, scene object, or materials [PHB08, HPB06b]. Using the Symmetric Lighting technique developed in Chapter 5, we can perform relighting of scenes if we maintain the assumptions that the camera and scenes are static during the capture process. If the scene, camera, or lights move, then the light capture process needs to be performed again. In our experiments the light capture process can take a few seconds to a few minutes as our system is in the prototype stage. A fully realized capture system could be developed for commercial purpose that could perform the capture process in real-time alleviating the assumption that the scene does not move. This is discussed briefly in the implementation but left as an extension for future work.

In Figures 6.1 and 6.2 show two scene examples relit using our method. These scenes consist of two lights, several objects with unknown geometry, reflectance, and appearance (texture). The only information known about the scene is the color of the two lights used to illuminate the scene (Figure 6.1: RGB values for



Figure 6.1: Images of a dish of candy relit with the application developed in this work. The candy was originally lit with blue light and red light (left image). The light color is then removed from the image using our relighting application and is re-rendered with a new lighting consisting of a green and purple light colors (right image).

$l_1=[226,95,120], l_2=[77,80,131]$ and Figure 6.2: RGB value for $l_1=[36, 128, 36], l_2=[128, 36, 36]$). The capture process consists of acquiring two images from which a β map is generated. The β map is then used in the relighting pipeline to perform real-time modification of the lights at over 100 frames per second (FPS) on a modest GPU¹. Our relighting application also has the capability of reducing or removing the influence or specific lights within the scene, which differentiates our relighting method from image re-coloring [RAGS01]. Figure 6.2 shows three images, on the left a scene with two lights (as originally designed), the center image shows the scene with one light removed, and third image with both lights removed. Evidence that the lights were actually manipulated is obvious when looking at the shadows below the spheres, the left images has two shadows generated from the two lights and the center has only one shadow generated from the single light. The lights were removed completely using only our relighting technique. Additionally, we can also increase the intensity of the lights to produce virtual High-Dynamic Range (HDR) images of the scene. We discuss our HDR technique as well as other lighting configurations, such as three-point lighting and non-photorealistic rendering (NPR) relighting in

¹In this context we mean modest to assume a consume level gaming card manufactured within the last five years which has a minimum shader level that includes fragment programs

the next subsections.

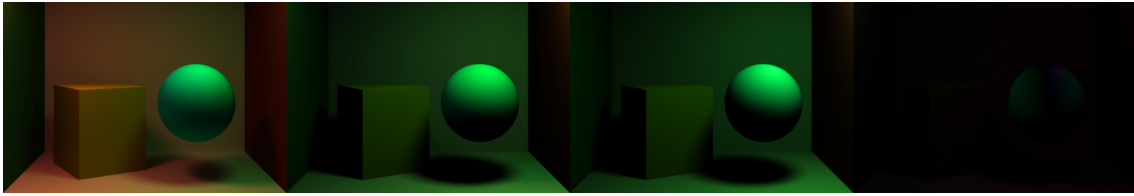


Figure 6.2: With relighting it is possible to remove the lights as well; left is the full scene with all lights (2 lights), left center is the same scene with one of the lights removed with our technique. The light was removed completely in the relighting application without additional processing. The center right image is the same scene re-rendered with only a single light for comparison. It is easy to see that the two center images contain only one shadow for the sphere compared to two shadows for the sphere in the left image. The image on far right shows an absolute difference of the pixel values of the two center images (difference was multiplied by a factor of two in order to make the difference perceptible).

6.2.1 Three Point Relighting

In our early relighting examples, most of our scenes show a lighting setup which is illuminate by two lights. A more common method of lighting used extensively in the Entertainment industry, is a lighting method that utilizes three lights [Bir06]. This method, aptly called three-point lighting², uses three lights to eliminate some of the problems that can be encountered when using only two lights, mainly to provide background separation, removing hard shadows, and accentuate shape definition better [Bir06]. The first light in this method, called the Key light, is responsible for providing the dominant lighting of the subject or scene. The second light, called the Fill light, provides soft lighting, which is used to reduce hard shadow lines and self-shadowing. The Fill light is often used to simulate indirect light by adding a softer and more diffuse light source from a different direction from the Key light. The third light, called the Back light, is primarily responsible for visually separating

²This is a misnomer as generally point lights are not used to illuminate scenes, but rather area lights as they provide more desirable lighting.

the subject from the background of the scene. The back light is generally placed behind the subject, casting light in the direction of the subject and the camera while not being visible in the camera image. Figure 6.3 shows an example setup using the three-point lighting method.

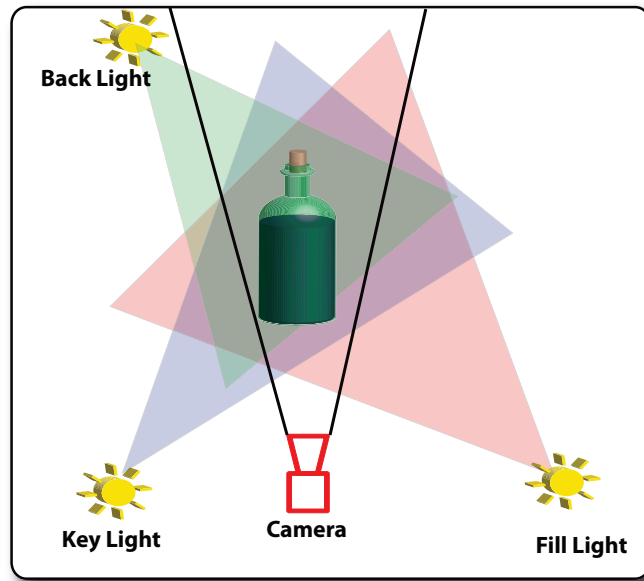


Figure 6.3: 3 Point lighting example.

One of the main problems solved with three-point lighting is separating the subject from the background in the scene. This is particularly true when the subject and background are similar visually or when color cannot be used to distinguish them (black and white film or low-light situations), which causes them to blend together. To show the benefits of three point lighting, we have added a third light to our standard relighting example scene depicted, which is depicted in Figure 6.4. The three-point light scene is in the center image of the figure has two lights at opposite sides (left and right) and a back light behind the sphere. To compare the three-point light method, this figure also has an image of the two-light setup from previous examples (left image). It is easy to see how the back light adds the perception of additional depth to the image that is not as apparent in the scene with only two

lights, thus providing a better delineation between the foreground and background. Artistically, this feature is an important tool for directors as it allows the director to focus the viewer's attention on the subject [Bir06]. As previously shown on the right image of Figure 6.2, we can also virtually remove any of three lights in this method through relighting providing the director the ability to blend the subject into the background if that is desirable.

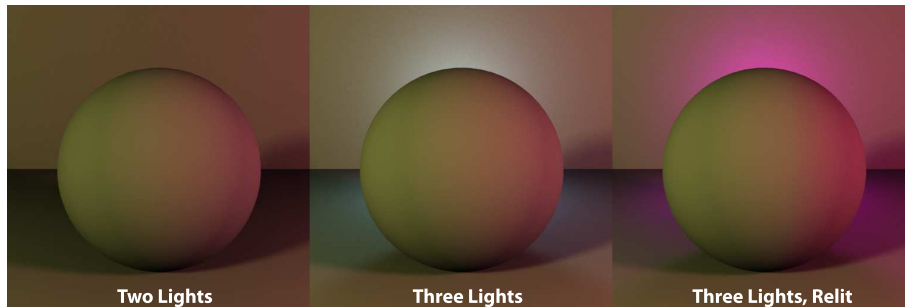


Figure 6.4: Two (left) and three point lighting (center and right) of the familiar sphere scene. In the center image two of the lights are to the left (Key) and right (Fill) of the sphere off-camera and the third light is behind the sphere providing back-lighting. Notice how the center image provides a perceptually deeper scene when compared to the two-light scene. Additionally, when the back light is modified, parts of the foreground where there is no influence of from the backlight is unaffected by the light change. Simple recoloring would not be able to precisely differentiate the different areas of the image.

6.2.2 High-Dynamic Range Relighting

In this subsection, we show that we can extend the range of the source luminance values beyond the typical 8-bit integer range. An important assumption of our capture method thus far, is that the luminance values of the lighting stay within the display range of the camera in order to appropriately estimate each light's contribution, or β value, to illuminating a pixel. Once the β values have been calculated, we can manipulate these proportions, as we have already seen with the dimming or removal of lights in the previous subsection. We can also use the β value to increase a light's contribution to a camera pixel to increase its intensity, thus making part of an image more bright. Furthermore, because the intensity values are represented virtually in the application, we can increase the luminance values beyond the physical capabilities of the original capture hardware creating an arbitrarily high-dynamic range image.

To achieve HDR rendering of the images we convert pixel values from 8-bit integers to floating point values and utilize a logarithmic scale for multiplying the luminance values of the lights, which was developed by Debevec et al [Deb98b] for luminance scaling. This provides a significant increase in the range of the lighting values which can multiply with the reflectance to produce the higher luminance images. This simulates the appearance of light with much stronger bulbs in the real scene and adds a separate dimension for editing the lights beyond just color. In Figure 6.5 we show an example image relit with our relighting application. On the left, the image with the green dragon creature (Yoshi) has significantly fewer high-luminance pixels compared to the same scene relit with higher intensity luminance. Below each image is a luminance histogram showing the actual luminance values for all pixels in the image. As is evident from the histograms, the image on the right has more pixels with higher luminance values.

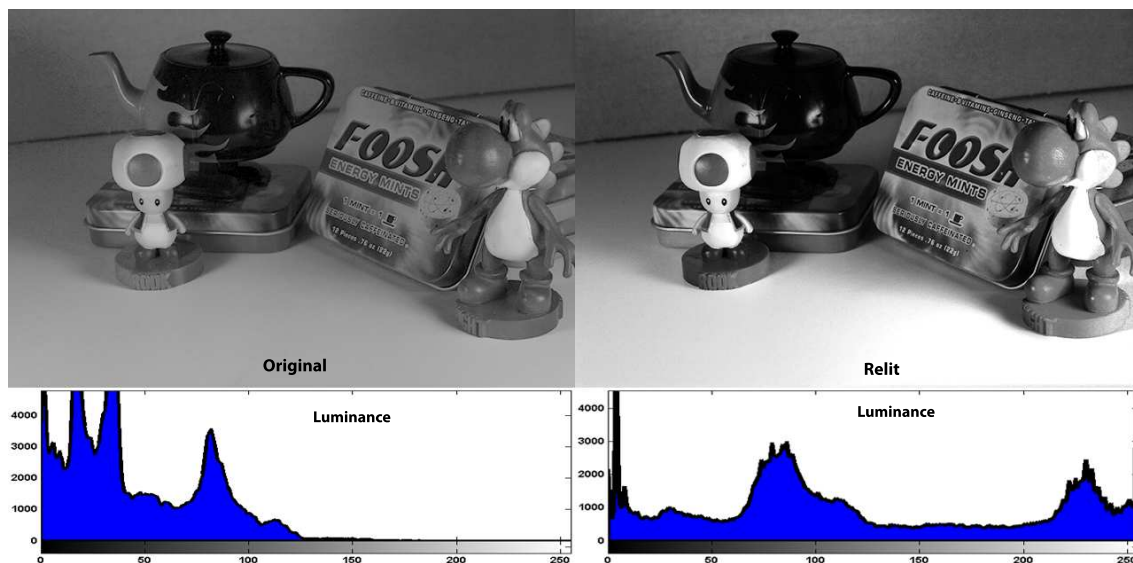


Figure 6.5: Left is the original image, and right is a relighting of the scene with HDR lighting added. Notice the dragon creature (Yoshi) has a significant increase in luminance with no color change, which is not present in the original image. These images are being presented as luminance images to convey the difference in luminance values better. Below each image is its respective luminance histogram. As you can see, the image on the right has significantly more values in the higher luminance range (values distributed to the right of the graph) than the left image. The histogram scale ranges from 0-255, because tone mapping is performed in order to map the image to a displayable range for 8-bit color channels.

6.2.3 Non-photorealistic Relighting

Figure 6.6 shows that relighting can be performed using Non-Photorealistic Rendering (NPR) of images as an alternative to photorealistic images. In the image in Figure 6.6, we apply a series of image processing filters to convert a photorealistic image into a NPR image using the techniques described in [WOG06] called Image Abstraction. This NPR method modifies the contrast of several image features, such as luminance and color opponency by reducing regions with lower contrast and artificially increasing regions of high contrast. A β map is then generated for the NPR image and subsequently relit. Since the light colors used to generate the illuminance of the image are known, the NPR effect does not hinder the relighting as the Symmetric Lighting calculations minimize the difference between pixels regardless of how the scene is lit.



Figure 6.6: Left is the original NPR image and right image is the original image relit with our method. The lighting can be modified as part of the NPR effect rendering or as a separate process. The NPR effect modifies the contrast in regions of low and high contrast (low contrast regions in the original image are subsequently reduced in contrast where higher contrast regions are increased). Relighting is performed after the NPR effect is applied, since Symmetric Lighting is not influenced by the NPR effect. The relit image provides a dramatic change in color.

6.2.4 Light Editing

In this section we demonstrate how light editing can be performed in conjunction with our relighting method. Given that a β map describes the proportion of light each light source contributes to the light impinging on the location in the scene, editing the β map is tantamount to changing this proportion. In essence, modifications to the β map changes the way lights distribute photons within the scene. Typically, light modifications are made in terms of color, intensity, and sometimes location. The β map allows for light editing changes that are fundamentally different than those typically associated light changes, such as light painting or light stenciling as described in the remainder of this section.

There many different ways to edit the β map which effect the look of an image during relighting. Figure 6.7 illustrates three examples of such methods which can be described as, 1) modifying regions of interest, 2) global modifications, and 3) artistically driven. The types of light modification demonstrated in this section are performed after the Symmetric light capture therefore editing the dispersion of light photons in the scene in this manner results in changes that can violate the physically accuracy of the lighting. For example, the image on the far right of Figure 6.7 demonstrates an artistically driven light painting in the shape of WPI, which adds additional light to the scene that previously did not exist. Additional methods for editing β map could include other image-based operations such as masks, per pixel operations, blending with patterns and gradients. These operations are often used in many other image editing contexts outside of the one described here and are therefore available in popular image manipulation programs such as Photoshop or GIMP. Since we store each individual lights influence in a different color channel of the β map, the aforementioned operations can easily be applied to individual lights within the scene by simply applying these operations to the appropriate channel in

the β map. Additional examples of image-based operations used to manipulate the β map are provided in the Appendix D.



Figure 6.7: In this example we show how editing the β map essentially edits the lights, their distribution of photons, and their influence on parts of the scene. Left is the original bath scene relit. The other images show several edits to the β map; left-center image has had a specular high-light changed from cast by a red light to a blue by selecting the β map area in an image editing tool and reversing the values (see red arrow), whereas the right-center image has had the influence of the blue light reduced by editing spread of β values corresponding to the blue light using the curves tool in an image editing tool. The image on the far right has a less nuanced change, where we’ve placed the acronym ”WPI” in blue light cast from blue light. Each of the edits can be done procedurally, by hand in a separate tool such as Photoshop, or by operations provided in our relighting tool.

In addition to editing lights, adding lights to an existing scene is also possible. Because the β map provides separate channels for each light, adding another channel to the β map is as trivial as adding an additional channel to the β map image. Typical images have up to four channels (RGBA), therefore β maps with the number of lights greater than 4 can just utilize a separate image. This could provide additional scenarios, such as removing real lights and adding synthetic lights to real scenes.

6.2.5 Previous work on Relighting

Capturing Lighting & Illumination Environments: Another approach to relighting a scene other than capturing the reflectance information is by capturing the illumi-

nation of a scene through inverse rendering. This process is thoroughly described in a doctoral dissertation by Stephen Marchner [MD98]. In his work he focuses on images captured on camera and using linearity properties of light to develop a formulation of inverse measurements of various properties of a scene including textures, light, and BRDFs. Later, Yu et al. [YDMH99] showed how this model could also be applied to inversely capture the global illumination of a scene by performing inverse Radiosity of a known scene.

In another direction, Debevec et al. [DWT⁺02], assume the lighting environment from some other location has been captured *a priori* and uses this lighting data to relight actors in order to composite them into the original scene footage for seamless video. The problem to solve here is how to position a real lighting environment to match the illumination properties of another captured environment. They solve this by building a light stage, which is a dome with 156 colored lights that can be activated to mimic the original lighting environment with the captured data. This was followed up by Wenger et al. [WGT⁺05] who used the same light dome in conjunction with very high-speed video cameras to capture the subject illuminated with basis illumination, which can be recombined for relighting. They additionally showed how to combine motion compensation to alleviate motion blur of the actor’s movement that occurred despite capturing at 2160 frame per second (FPS). In this work, we opted for a more simple, although view-dependent, approach of capturing only two images in the natural lighting environment. This eliminates the need for a light dome and compositing after the fact, while drastically reducing the complexity of the system presented here. Although our method reduces the cost and complexity needed compared to the systems developed by Debevec and Wegner, our method sacrifices view-independence and motion compensation.

As mentioned previously, another technique used for Relighting applications is

employing the use of captured real-world lighting such as Environment Maps [HS98, FLW02] or Light Probes [Deb98a]. These techniques assume the user utilizes the captured lighting as the desired lighting environment and at best only provides basic editing capabilities such as rotating the environment maps. They do however accurately capture the spatially-varying aspects of the whole lighting environment but provide little in the way of editing the lighting. Pellacini [Pel10] developed an interface called EnvyLight for editing the environment maps and provide clear lighting parameters for designers to edit the environment maps and light probes. Despite EnvyLight’s ability to edit the lighting maps, the ability to edit individual light sources does not exist. This is in contrast to the work presented here, in which the capability to edit the properties of the individual lights is a primary feature. It would be interesting to integrate a hybrid approach where the user can choose to either edit the environment map or the individuals lights in a global versus local fashion, but this will be left for future work.

Finally, a webpage titled Synthetic Lighting for Photography [Hae92]³ describes a simple technique for exploiting the linearity of light. This work is similar to the Symmetric Lighting technique presented here but lacks any academic theory, exposition, or results other than an image. To the best of our knowledge, this work has never been academically published but never the less warrants mentioning. The Symmetric Lighting technique presented in this dissertation exploits the linearity of light as well but is expanded beyond two lights, provides error analysis, and a formulation and representation for relative light contributions. Additionally, capturing lights individually does not take advantage of the reduce Signal-to-Noise Ratio when utilizing the ”multiplex advantage” [SNB07]. In other words, generally in photography insufficient lighting can result in noisy images making inferring infor-

³We mention this idea for completeness only as it applies only to related work.

mation about the scene more difficult. By photographing more lights simultaneously instead of individually, the images generated will be taken under more light producing a better signal-to-noise ratio when processing the images. This is referred to the "multiplex advantage" of photographing multiple lights.

Full Scene Description, Light Fields and Virtual Scenes: Relighting can also be considered in the context of virtual scenes in addition to real scenes. Virtual scene relighting presents a special case of the relighting application in that generally speaking all scene information is directly available or can be calculated with relative ease. The availability of full scene information means that virtual scene relighting is expected to perform as well or better than lighting modifications that would physically occur on a real set from modifying the lighting. As observed by Gershbein and Hanrahan [GH00] both virtual and real scene contain visually complex scene with equally complex characteristics, but until recently the visual results of modifications to the lighting environment was immediately available only in the real scenes. Several academic works have been proposed to not only increase the rendering capabilities including speed of virtual relighting, but also the properties of light. Gershbein and Hanrahan [GH00] developed an OpenGL based virtual lighting system that used a deep framebuffer on the graphics hardware and was reported to speed up relighting by 2500 times for interactive rates of 20Hz. This technique used factored BRDFs and lighting and calculated their product in texture hardware.

Utilizing faster graphics hardware, Hasan et al. [HPB06a] was the first to incorporate indirect lighting into the relighting applications for virtual scenes. They work under the assumption of fixed views but with highly complex geometric and material details. Their key observation was to linearly transform samples of direct lighting within the framebuffer to indirect lighting. Similarly, Obert [OKP⁺08] provides relighting control for indirect lighting but adds a more comprehensive artistic

control of indirect lighting. Then Ragan-Kelley et al. [RKKS⁺07] added the ability to integrate transparency, motion blur, and antialiasing. In contrast to the technique present in this work, which was primarily developed to be used on real scenes and does not have access to the full scene information, the aforementioned works provide many more features. Although the presented work shares the use of accelerated graphics hardware as the main computational platform, indirect lighting for our technique has been left as future work.

Precomputed Radiance Techniques: Relighting is frequently performed on compressed representation or basis for the lighting. In real-time rendering, the use of Spherical Harmonics (SH) (low-frequency) and Wavelets (All-frequency) has been shown to be an effective set of bases for representing lighting environments. Kautz et al. [KSS02] showed that relighting can be performed simply via dot product of the precomputed SH coefficients for the Lighting with coefficients for the BRDF. Similarly, Sloan et al. [SKS02] showed this technique could include global effects as well.

Ng et al. [NRH04] showed that a variant of the precomputed radiance could utilize a wavelet basis instead of SH with the added benefit of not being restricted to low-frequency lighting. This allowed them to perform a triple integral represented as a full six-dimensional space of materials, visibility, cosine term and lighting. This technique then extended to include time-varying lighting and enhanced shadows [NRH03]. Finally, Cheslack-Postava et al. [CPGN⁺07] proposed a technique to reduce the storage via a compression representing 4D wavelets for light-transport. All these and related radiance pre-computation techniques require precise geometry information as part of the factored light transport is stored per vertex and thus limited to virtual scenes. This differentiates the work presented here from their technique in that Symmetric Lighting can be utilized for both virtual and real

scenes. Precomputation techniques are also computationally expensive and time consuming to perform as the factored light transport is performed for all views due to the interactive nature of the geometry for those scene. This is in contrast to the technique presented here which can be done in real-time but has the limitation of being view-dependent.

Active Illumination Techniques: Illumination can not only be sensed from a scene but it can be projected as well for the purposes of relighting. This can be in the form of a projector illuminating a scene, flash photography, or even using lighting outside the visible spectrum. Eisemann & Durand [ED04] used pairs of images with and without flash to enhance images in low light situations using an image with ambient light and with flash to reduce noise, enhanced color, and relight images with information from both. Similarly, Wang et al. [WDC⁺08] used Near-infrared (750nm-1.4 micro meters) lighting to mitigate low light situations by capturing the illumination using video sequences. Another novel flash based technique used to relight images was that of Kim and Kautz [KK09], which captured an illumination estimation of the correlated color temperature and designed a chromatic flash that would automatically match the color temperature. These techniques generally introduce artificial lighting into the real scene using patterns or coded light. Our technique is similar to these techniques and is considered an Active illumination technique due to manipulation of the scene lights. Instead of introducing patterns or artificial light that is not part of the scene as the aforementioned techniques do, we manipulate only the light that already exists as part of the scene. In other words, we utilize only the natural lighting that would occur within the scene, manipulate it in a timed sequence for our calculations.

Different basis have been used to perform relighting, capturing dense sampling of both reflectance and illumination. Using a proprietary light dome, Wegner et

al. [WGT⁺05] capture a spherical set of images using time-multiplexing representing lighting from all angles using high-speed video and discrete LED light sources. Moreno-Noguer et al. [MNNB05] use a similar technique using video and lighting to capture a basis representation of the scene illumination but show that a minimal optimal basis can be obtained through calibration. Finally, Sen et al. [SCG⁺05] exploit the Helmholtz reciprocity of light for capturing the light transport of a scene by interchanging the light and camera within a scene. Their technique like the one presented here requires no knowledge of the reflectance or geometry present in the scene. The work presented here is similar in spirit to the work of Sen et al. but does not exploit reciprocity but instead the symmetry of the lighting within the scene. Furthermore, the work presented here is much less complicated than that of Wenger et al and Noguer et al, in that we do not require or even need a specialized basis for representation, or to capture hundreds of images per second. Instead the technique presented here is simple and only requires a single image per light.

6.2.6 Relighting Implementation

In the entertainment industry, there is strong motivation for performing relighting in order to create images that convey the creator’s vision, as well as to allow for images to have been captured in different locations but brought together to appear as if they were shot in the same location. In terms of the movie making timeline, the relighting process is performed in post-production. This can be problematic for directors since they have to make lighting decisions during principle photography (i.e., before post-production) based on lighting changes that may be made later on in the movie making process [PTMD07]. Therefore it would be advantageous for directors to view how a scene might look in post-production, while actually shooting the scene. This work achieves this goal, by allowing for relighting of real

and synthetic scenes using PCam, on-set during filming. This will give directors more control and freedom for lighting during principle photography and reduces the need for reshoots, saving time and money.

The method for relighting used in this work was developed in Section 5.2.2 called Symmetric Lighting. This involves taking two or more photographs of the scene with the lighting colors transposed as described in Section 5.2.3.1. As illustrated in Figure 6.8, the previs relighting process consists of acquiring images from a camera as input to the relighting application. The relighting application then calculates a β map (Step 1), which depends on the two input images and light color values l_1 and l_2 . One of the original images is then processed to remove all lighting, which results in a lighting-free image that has only reflectance and shading information (Step 2). Then during relighting (Step 3) a new set of light colors chosen by the user l'_1 and l'_2 are multiplied with the β map to produce a new lighting map. The new lighting map can then be multiplied element-wise with the lighting-free image to produce a relit and rendered image. The user can change the lighting values l'_1 and l'_2 , which are subsequently incorporated into the next render update. This allows the relighting application to remake the lighting map in real-time, thus performing real-time relighting simply by iterating over step three, as is shown in Figure 6.8.

As can be seen in the code Listings 6.1 and 6.2, this workflow can easily be translated into a program for performing these operations. In these two listings we present GLSL code that was used to implement a Camera Shader for use on PCam. The camera populated two texture maps with the two captured images (in this example we assume two lights). Then for each pixel⁴ in the two images we calculate the β value by searching through all possible β values to find the value that is minimal in terms of the least squares (i.e., L_2). On a first generation Motorola

⁴in our configuration we assigned a single texel to each pixel in order to avoid any imaging artifacts.

Relighting Application Pipeline

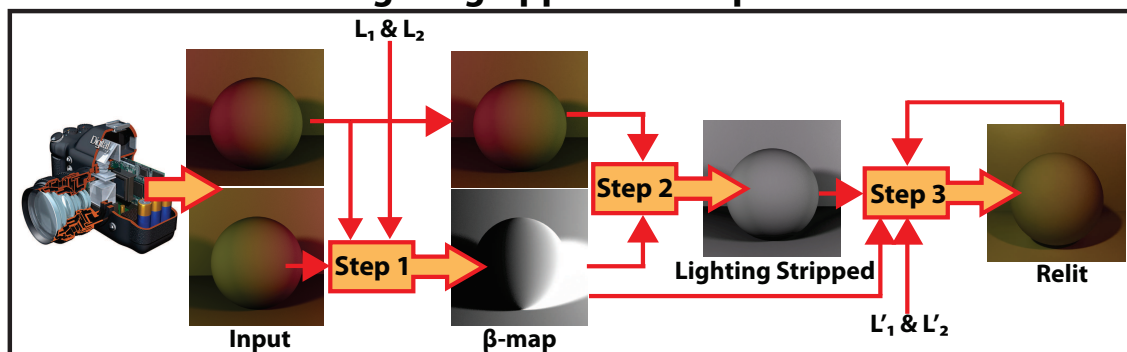


Figure 6.8: Diagram illustrating the relighting workflow; images from the camera are and light color information are input into the relighting application. The application generates a β map from the input. Using the original image and β map, the original image has the original lighting removed. Then new light colors can be multiplied with the β map and the lighting-free image to generate a relit image.

On a Droid phone this operation can be performed at 24 FPS with the un-optimized version shown here. We have also listed in the Appendix B a Matlab version of the code for exposition purposes.

Listing 6.1: A brute force implementation of the Symmetric Lighting capture technique performing calculations outlined in Equations 5.6 and 5.7. This GLSL code shows a simple way to perform Step 1 from Figure 6.8 which generates a β map to be used as input to Steps 2 and 3. For exposition purposes we omitted any optimizations.

```
/*Symmetric Lighting
* Description: For exposition purposes this method performs
* a brute force approach to finding the Beta value for each
* pixel in the input images. The light values are predetermined
* from either the synthetic rendering system or from the lighting
* gels used to illuminate the scene. Assumes the images are linear
* in terms of their color response (i.e. gamma=1).
* Output: A Beta map calculated from the light values l1 & l2 and the two
* images rgb1 and rgb2 generated from the scene.
```

```

* Written by: Clifford Lindsay, Feburay 26, 2011.
* Department of Computer Science , WPI */
uniform sampler2D src_tex_unit0; //Image 1 texture
uniform sampler2D src_tex_unit1; //Image 2 texture
uniform vec3 l1; //Light colors used in the scene
uniform vec3 l2;

void main(){
    float finalBeta    = 0.0;
    float finalLength = 100.0;
    // get image 1 and image 2
    vec3 c1 = texture2D(src_tex_unit0 , gl_TexCoord [0].st).rgb;
    vec3 c2 = texture2D(src_tex_unit1 , gl_TexCoord [0].st).rgb;
    vec3 I1;
    vec3 I2;
    // Iterate through Beta vals.
    for (float beta=0.0; beta < 1.0; beta=beta + .001){
        I1 = beta*l1 + (1.0-beta)*l2;
        I2 = beta*l2 + (1.0-beta)*l1;
        //Calculate Euclidian length
        if(length(c1*I2 - c2*I1)< finalLength){
            finalLength = len;
            finalBeta    = beta;}
    }

    // return Beta
    gl_FragColor = vec4(finalBeta , finalBeta , (1-finalBeta ), 1.0);
}

```

The relighting is performed in Steps 2 and Steps 3 as previously mentioned. In Listing 6.2 we have combined these steps into a single GLSL fragment shader used to implement a Camera Shader in PCam and the relighting application used in the user studies. In this example, we have an image from a scene lit by two lights and a β map produced from Step 1. Four light values (two previous values l_1 and l_2 and two new values l'_1 and l'_2) are used to remove the lighting from the image and apply new lighting. The resulting image is rendered using a multi-pass method where Step 1 is performed in the first pass and Steps 2 and 3 are performed in the second pass.

To scale this to more than two lights the same structure of the code is virtually the same except for the additional light values. We have also listed in the Appendix [B](#) a Matlab version of the code for exposition purposes.

Listing 6.2: A brute force implementation of the relighting technique performing calculations outlined in Equations [5.6](#) and [5.7](#). This GLSL code shows a simple way to perform Step 2 and Step 3 from Figure [6.8](#) which produces an image of the scene relit using different lighting values. For exposition purposes we omitted any optimizations.

```
/*Relighting
 * Description: For exposition purposes this method performs
 * a relighting of an image by removing the old light values
 * from each pixel in the input image. The new lighting is
 * added by multiplying the Beta map time the new lights.
 * Assumes the images are linear in terms of their
 * color response (i.e. gamma=1).
 * Ouput: A fragment that corresponds to the input image
 * with new lighting values.
 * Written by: Clifford Lindsay, Feburay 26, 2011.
 * Department of Computer Science, WPI
 */
uniform sampler2D src_tex_unit0;
uniform sampler2D src_tex_unit1;
uniform vec3 l1;//old light 1
uniform vec3 l2;//old light 2
uniform vec3 l3;//new light 1
uniform vec3 l4;//new light 2

void main(){
    //retrieve original image.
    vec3 c1 = texture2D(src_tex_unit0, gl_TexCoord[0].st).rgb;
    //retrieve Beta map image.
    vec3 beta = texture2D(src_tex_unit1, gl_TexCoord[0].st).rgb;
    //calculate the proportions of the old lights for each pixel
```

```

vec3 I      = beta.r*l1 + beta.b*l2;
//calculate the proportions of the new lights for each pixel.
vec3 new_l1 = beta.r*l3;
vec3 new_l2 = beta.b*l4;
//remove old light to generate lighting free image.
vec3 wb = c1 * (1.0/(I+.01));
//apply new lighting.
vec3 r1 = (wb*new_l1)+(wb*new_l2);
//return relit image.
gl_FragColor = vec4(r1, 1.0);
}

```

Our technique assumes that, apart from the light color rotation, everything else within the scene remains stationary during the capture. If the scene is most composed of diffuse objects, then only two images are required. If the scene contains a significant number of specular objects then four images would be captured, two for Symmetric Lighting and two extra to compensate for the specularity as described in Section 5.2.3.4. Compared to other prior techniques, one or two images per light source as input is significantly lower than the number of inputs required in all other techniques, which can require hundreds of images [PTMD07] with the caveat is that our technique is view-dependent. Being view-dependent is not a limitation as it may be unnecessary to fully enumerate all camera positions to capture all potential variations as they may never be used. Additionally, when performing relighting it has been shown that relighting artists focus on lighting parameters such as color or intensity and not camera locations [PHB08, HPB06b]. Additionally, if camera positions do change, recalculating the β map in real-time to accommodate new views can be achieved with modest improvements to our Symmetric Lighting method.

The lighting setup we used to calculate the β map described in Chapter 5.2 consists of a number of lights with predetermined light colors. We assume that the light colors are known to the relighting application in order to provide a constrained sys-

tem of variables to solve Equation (5.7) to perform the inverse lighting calculations. Our original lighting setup used two lights with bulbs that emit light with Correlated Color Temperature (CCT) of 6500k or over cast daylight illumination which is sometimes referred to as CIE D65. Each light was filtered with a different RoscoLux polycarbonate gels made by Rosco [Lab11] in order to alter the color of the light. Each filter gel has published specifications for transmittance and wavelengths that are filtered including the dominant wavelength transmitted through the filter when used in conjunction with a specified source. Because the light source and filter gels have manufacturer provided specifications, the light source color is actually provided when using the gels according to their specifications. Although not recommended, If light bulbs with color different than what is recommended by gel manufacture are used, formulas for determining light color of the light plus the gels can be found in [GW00]. LED light panels with mono-chromatic color are commonly used for set lighting. Alternatively, these panels could be augmented with tri-color LEDs, which could provide set lighting that can be electronically altered to other colors for performing Symmetric Lighting calculations. This type of lighting setup would provide two equally viable scenarios for performing Symmetric Lighting calculations, 1) two lights with their color set to a typical white light (e.g., CIE D65, which is midday sun) would be used nominally, which are then momentarily switched to different light colors (red and green) in order to capture the Symmetric Lighting images, or 2) use two different light colors permanently on set (again red and green) and only rotate the colors in order to capture the Symmetric Lighting images. Consequently, scenario 1) is what we performed when using the Rosco gels, but scenario two is also feasible with the gels as well.

Color gel filters are often used for entertainment industry, such as television, movie production, and stop-motion animation. For previs relighting as previously,

the colors of the lights used with the Symmetric Light capture setup do not have to match those used nominally on set (they should be in the same location). In other words, typical white light can be used on set until Symmetric Lighting calculations are performed for previs. Then gels would be placed over the lights then removed once the Symmetric Lighting calculations were finished. Since this is an active illumination technique, the light colors used in the Symmetric Lighting are chosen to be far enough apart as to provide a clear separation of the lights. In other words, to avoid a degenerate lighting setup (i.e., having light colors that are too close to one another, which reduces the Symmetric Lighting dimensionality describe in Section 5.2.3.2), the light colors are chosen to be a significant distance from one other in terms of the sRGB color space. In our setup we use shades of red (Rosco #4660, 60-red:R=218,G=131,B=102), green (Rosco #89, Moss Green:R=42,G=165,B=95), yellow (Rosco #12, Straw Yellow:R=255,G=203,B=0), and blue (Rosco #59, Indigo Blue:R=0,G=38,B=96). The lighting system acts more like a camera flash so the lights from the capture system are only on briefly long enough to perform the capture (few seconds) and perform the relighting for the previs, then the typical light colors could be restored. Figure 6.9 show an image of our lighting setup in the Lego scene with two lights filtered using the Roscolux gels.

For this project there were three different cameras used to capture images or real scenes, the first was a Point Grey Research FireFly MV Firewire camera (Model FFMV-03M2MC) with resolution of 640x480 pixels with a Nikon machine vision lenses. The second camera was a Canon Rebel DLSR 500D with Canon lens, with the capability to capture images up to 5184 x 3456 pixels (much larger than necessary for previs). The third was a stock camera from a Motorola Droid Smart phone. In addition to images capture from a real scene, synthetic scenes were also used for relighting. The Canon DSLR and Firefly cameras we used to compare the

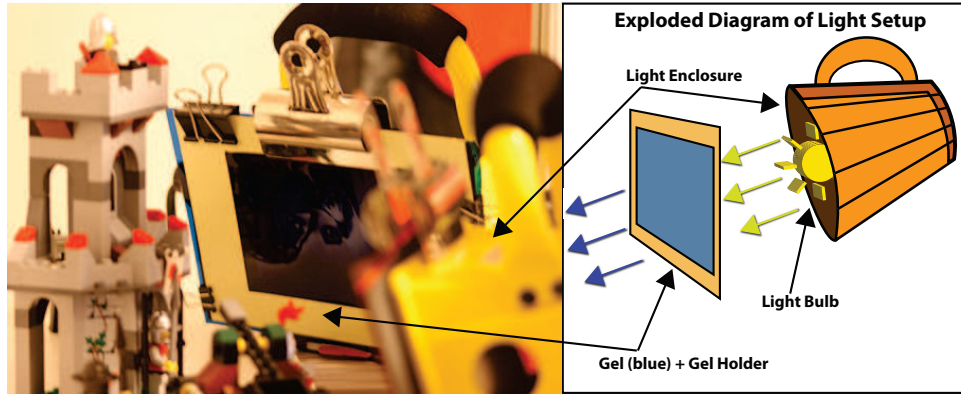


Figure 6.9: Left side of the figure is a picture of the Lego scene with the lighting setup described in this work. This scene has two lights (yellow enclosures) with colored gels placed in front of the two stationary light sources. The right side of the figure shows a diagram of the lighting setup. Placed in front of the light enclosure is a rectangular gel holder (cardboard frame) with a colored Rosco gel inserted inside. The gel and gel holder completely cover the opening of the light enclosure so that only filtered light from the gel is emitted. The enclosure houses a CIE D65 light bulb and is lined with foil to reflect the light outwards.

noise profiles of the camera, while the Smart phone camera was chosen because it contained on-board processing capabilities. The Canon DSLR produced much less noise than the other two cameras, resulting in Symmetric Lighting calculations with fewer noise anomalies (i.e., wrong β values). The synthetic scenes were rendered using a physically-based rendering system called V-Ray [Gro11]. Once captured, the images are transferred to the relighting application in order to calculate the β map. For the Firefly camera images can be acquired and copied directly to memory on a laptop or desktop computer, where the β map can be generated on-the-fly. For the Smartphone with built-in camera, the images can be acquired directly on the phone and a β map can be accessed directly within the Smartphone's internal memory instead of needing to be physically copied. Because the camera for the Smartphone is built-in to the camera and the phones are generally handheld which can cause blurring during image capture, it is necessary to use a tripod or some other device to keep the phone from moving during image capture. For the synthetic scenes,

the images can take several minutes to hours to generate in V-Ray depending on the complexity of the scene and the sampling rates of the ray tracer. Then the images from the rendering system are hand copied over to the relighting application. Similarly for the Canon 550D, the images have to be hand copied to the relighting application, due to the camera not being able to directly copy the files over without proprietary software.

6.2.7 Evaluations

In this section we provide quantitative photometric evaluation for assessing the performance of the relighting operations described in Section 6.2 at the beginning of this chapter as well as the β map calculation from Chapter 5. At the end of this section we also provide additional results to show our relighting method being used with different scenes and different lighting setups. In Section 5.2.3.3 we described what happens when scenes violate our Lambertian reflectance assumption by containing specular highlights. Therefore, our first evaluation in this section provides a quantitative assessment of error associated with specular highlights when calculating the β map. Our second evaluation assesses the performance of our relighting method by comparing several relit scenes to ground truth images using a perceptual color difference metric. Our third evaluation compares the luminance differences in scenes that have been augmented during relighting with a HDR multiplier to that of similarly rendered scenes with increased luminance.

The evaluations were performed using three different types of metrics, a metric for determining color differences, an absolute difference metric, and metrics for determining luminance differences. For color difference, the Delta E (ΔE_{00}) metric [LCR01] will be used as it is a standardized method developed by the CIE group for evaluating color. This metric is commonly used to assess perceptual difference

images that exhibit small color differences. Because we expect our relighting method to manifest errors in small differences in color, we use the ΔE_{00} to assess our relighting. The absolute image difference or L_1 norm metric is a pixel-wise subtraction of two images, where the absolute difference in pixel values shows how they deviate from each other [Rus11]. Because the β map has no human visual system analogue, we chose an absolute measure to gauge the differences in error for generating β maps with scenes that contain specularities.

In Section 5.2.3.3 we described the error associated with scenes that violate our Lambertian reflectance assumption. We previously hypothesized that scenes that contain more specular reflections would produce more error. In this section we quantify that error for five different reflection models that contain various levels of specular reflection. If we assume that the ground truth β map is one that is calculated using a scene which contains only objects with Lambertian reflectance, then we can estimate the error by comparing any other scene’s β map with the β map from our ground truth. Because there is no analogue for β maps in the Human Visual System, there is no need to use a perceptual metric for gauging error. The metric we use is the absolute image difference, which we calculate using the *imabsdiff* difference method provided by Matlab (Matlab code provided in Appendix B). The absolute difference is calculated per pixel and normalized, allowing us to calculate exploratory statistics regarding our error. Figure 6.10 provides a set of Box plots describing our error statistics.

For the five different reflectance models containing specularity, we have ordered their plots in increasing values of specularity from left to right. It is easy to see that the error increases with the amount of specularity in the reflectance model, since no other properties of the scene have been changed. Parts of every scene may exhibit negligible specularity as seen in Figure 6.11, therefore all error ranges start at zero

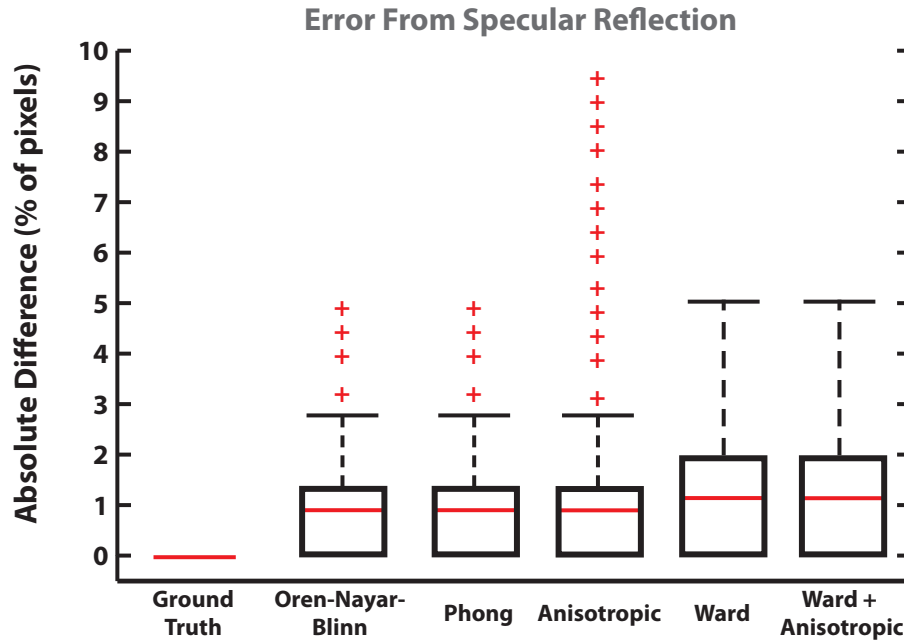


Figure 6.10: Since our model is based on a Lambertian reflectance, then we can assume that the ground truth error related to specularity is a pure Lambertian reflectance model and all other models can be compared to this one. We plot the absolute difference in terms of pixel values for the β maps for a subset of different reflection models that exhibit specularity. All other variables of the scene remain constant and only the reflectance changes.

and can produce error as high as 5% for any pixel and an average error around 1%.

Figure 6.11 below shows visually the error graphed in Figure 6.10.

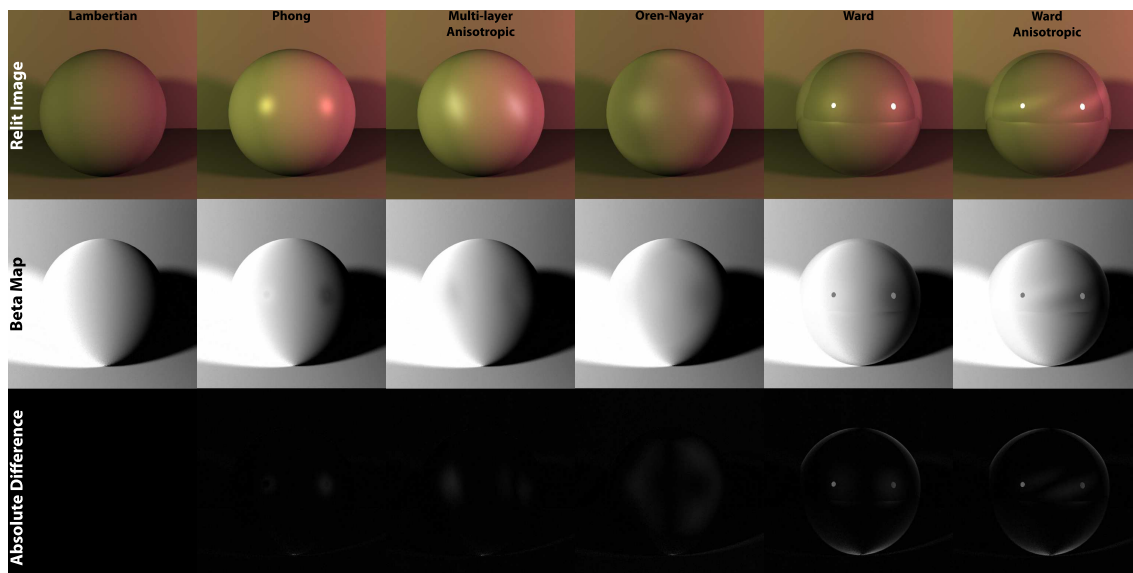


Figure 6.11: Visual comparison of the error associated with specular reflection. Top row are the relit images with various reflectance models. The second row shows their β maps. Third row shows the absolute difference.

To assess the performance of our relighting method we used the ΔE_{00} metric. The ΔE_{00} metric provides a measure of the differences in color based on the Euclidian distance within a specified color gamut. Since 1976, there have been several revisions of the current ΔE metric that use a different color space and slightly different equations for calculating color differences, which resolve issues related to perceptual uniformity as well as providing color differences for use in different contexts, such textile manufacturing, digital application, and print media.

$$\Delta E_{00}^* = \sqrt{\left(\frac{\Delta L'}{S_L}\right)^2 + \left(\frac{\Delta C'}{S_C}\right)^2 + \left(\frac{\Delta H'}{S_H}\right)^2 + R_T \frac{\Delta C'}{S_C} \frac{\Delta H'}{S_H}} \quad (6.1)$$

In the case for CIE ΔE_{00} the primaries are specified in the L*C*h color space [LCR01], where S specifies the compensation coefficient and R_T is the hue rotation term. The LCh color space is three dimensional and specifies the color in terms of lightness or luminance (L), chromaticity (C), and hue (h). This method is commonly used for evaluation of Color Appearance Models (CAMs) as well as other color related experiments. As long as the color are transformed to the same whitepoint or recorded under the same illumination, then differences in color that are imperceptible can be adjusted using a global tolerance. This tolerance affects the hue of LCh color space by allowing a range within the compensation coefficient weighting of the hue (S_h). Essentially, any color difference that does not cross the just-noticeable-difference (JND) threshold based on this tolerance is considered the same color. That threshold depends on the environment the viewer is in, and context in which the differences are used, but in general it is not possible to view color differences below $\Delta E_{00} < 1$ [Fai05]. Additionally, it is common in the textile and print industries that two sample colors having a $\Delta E_{00} < 5$ are considered equivalent [BBS00]. We adopt this threshold for our evaluations because currently a standard threshold for our context does not yet exist and this threshold seems reasonable

given the stringtness of the print industry.

In our relighting experiments we used several different scenes for which we relit with predetermined light values. Using the same predetermined light values we rendered the same scenes using the physically-based rendering system V-Ray to produce ground truth images of the scenes to compare against our relit images. For five different lighting values, we compared the relit scenes with the rendered ground truth images using the ΔE_{00} metric previously described. Matlab code for performing the color difference statistics is located in Section 8.

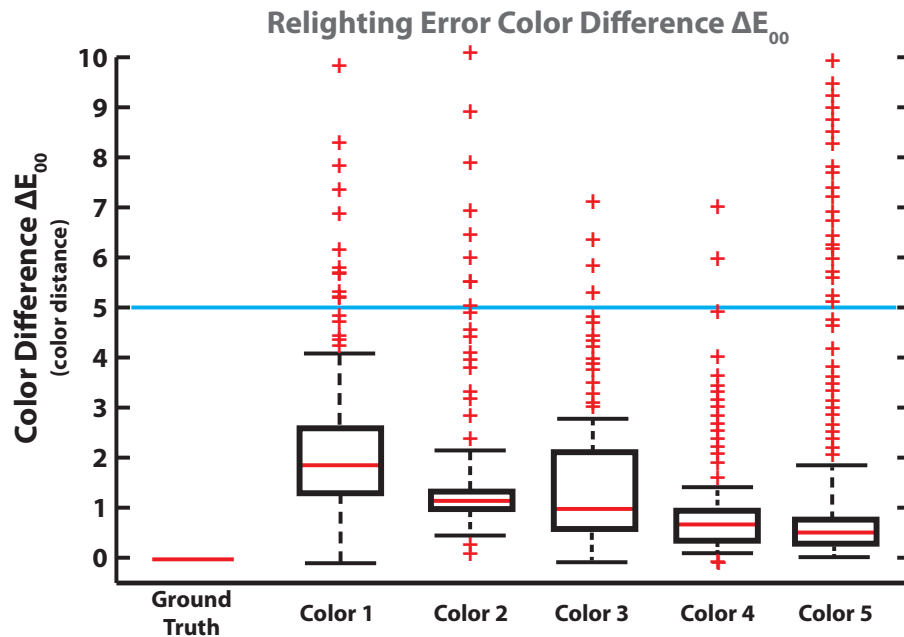


Figure 6.12: This figure show the relationship between the error of a set of relit image and their corresponding ground truth images. As previously stated we have chosen to use the ΔE_{00} threshold from the print industry to gauge perceptible differences in images. An image containing pixel differences below a threshold of five is considered the same. As you can see most of the pixels for each set of images fall below the threshold with the exception of a few outlier pixels, which we attribute to noise in the images.

Figure 6.12 shows a set of box plots describing the error between the relit images and the ground truth. Each box plot describes the error associated with different lighting values used to relight and re-render the scenes. As can be seen by the

figure, all color difference values for relighting fall well below the $\Delta E_{00} = 5$ line which we stated as an acceptable difference. The images from the scenes do contain small amounts of specular reflection, at least a portion of the outliers (red crosses) in the figure can be attributed to that while others can be attributed to noise in the images. We should point out that the number of outliers is actually quite small relative to the total number of pixels used to calculate the statistics in this figure. Additionally, the scale for the ΔE_{00} plots in Figure 6.13 is transformed to a scale from one to six to allow the values for viewing purposes, as the original scale of one to twenty made the images too dark to see the differences as they are quite small.

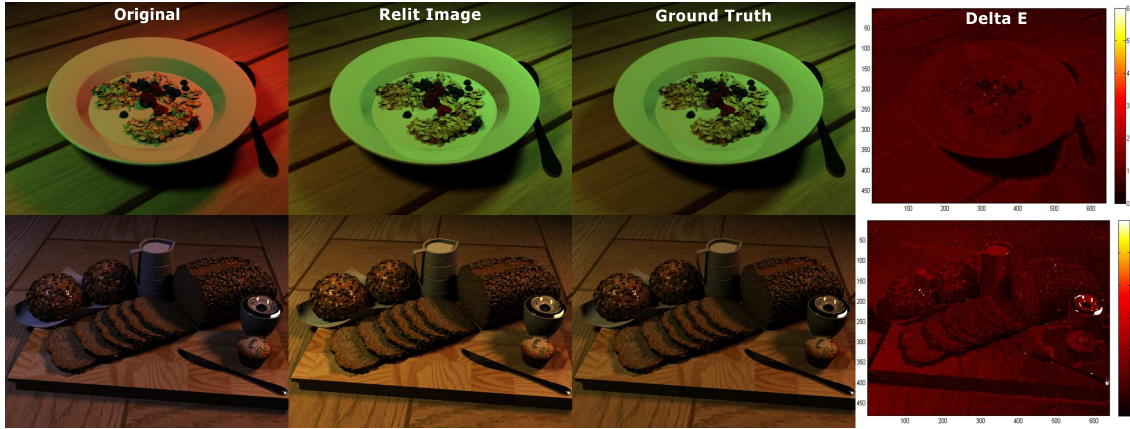


Figure 6.13: Visual comparison of the error associated with our relighting application. First column is the original images before relighting. The second column shows a relighting of the original image. Third column is the ground truth images to compared the to the relit images of the second column. Fourth column is a figure showing the color difference at each pixel using the ΔE_{00} values plotted with Matlab.

6.2.7.1 Additional Results

This section shows some additional results of our relighting application. The results are mixed virtual scenes rendered with a physically-based rendering system and the rest are real scenes. For each scene the light source colors are known and the β map for each scene image is calculated "on-the-fly" using hardware acceleration. Each real scene use colored lighting generated from polycarbonate gels filters placed in front of a light source with CIE D65 color temperature.

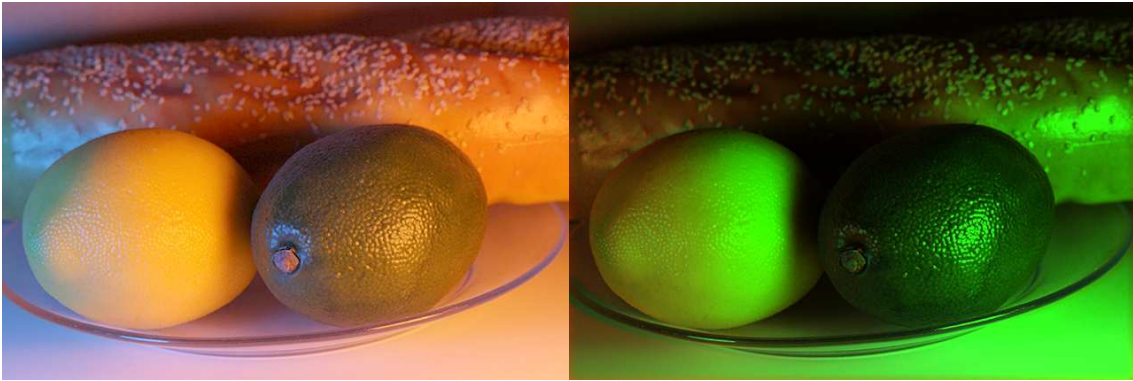


Figure 6.14: Left is the original image, and right is a render of a real scene relit with scene lighting.

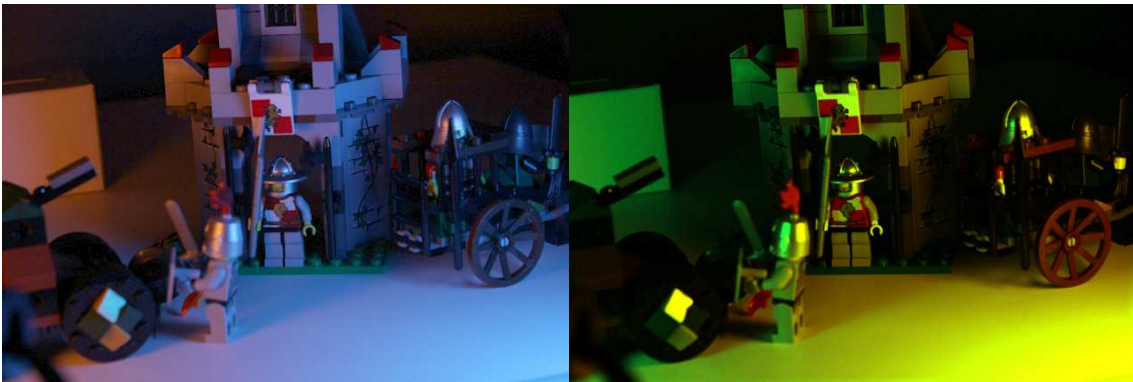


Figure 6.15: Left is the original image, and right is a render of a real scene relit with scene lighting. Despite having a significant amount of specularities within this image, this relighting works well.

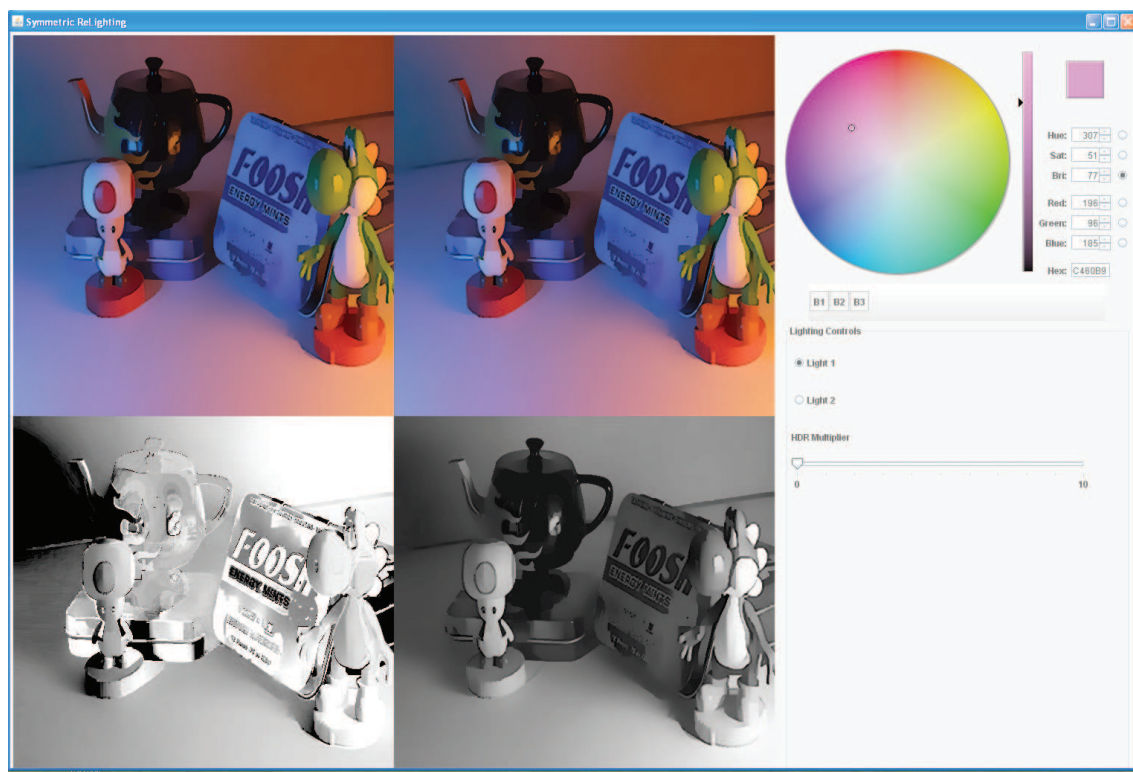


Figure 6.16: A screen shot of the desktop application developed for relighting.

6.2.7.2 Limitations

In general, the method presented here for relighting is faster, amenable to GPU acceleration, less complicated, and less data intensive than the other methods previously proposed. But it does have several limitations that the other methods do not have. The first limitation is that Symmetric Lighting is view-dependent. The consequence of view-dependence is that the camera or the lights must not move, otherwise a new β map must be calculated. Since the application performs its computation on the GPU, this can be done relatively fast on mobile device with speeds approaching real-time and real-time on more powerful GPUs. In other words, all operations, including creating a new β map is done for every frame with render times in 30 frame per second on a desktop with modest GPU (Nvidia 9600 GT).

The second limitation is that the relighting calculation assumes a discrete set of light sources, not a continuous environment, such as an outdoor scene. This implies that it cannot handle spatially varying light sources as input to the relighting algorithm. Indeed this is true, as the initial testing setup used several multi-color LED lamps. The lamps would produce light colors as a mixture of red, green, and blue LEDs. But the low-quality manufacture of the lamps made them inconsistent spatially. In other words, a lamp that was emitting a yellow light, would have some areas that exhibited red light that green or vice versa, causing large errors in the β map calculation. This necessitated the change from low-quality LED lights, to movie quality Rosco gel lighting.

6.3 Light Color Estimation and Calibration

There are many situations where estimating the illumination within a scene can be important. On movie sets, estimating the illumination is often used to aide in relighting, rendering virtual objects in consistent lighting as real objects, and in color processing. Light color estimation is the task of estimating the color of a particular light source which is different from the task of estimating the illumination of a scene. The difference is that light source color estimation refers to explicitly estimating the photometric or radiometric properties from a full range of possible color values that produce the sensed color, whereas illumination estimation tries to classify an a lighting environment within restricted group of standard color values called blackbody radiators [Fai05]. Blackbody radiators are group of evenly spaced color estimates that follow a specialized locus called the Planckian or black body Locus [WS67], which is single path on the CIE 1931 chromaticity chart corresponding to correlated color temperatures [CIE87] as seen in Figure 6.17. The black body radiators, measured in degrees Kelvin (K), have a series of corresponding standard illuminants which are defined by the CIE and emit light colors on this locus. Examples of such illuminants include Tungsten A (2856K), B (4874K), C (6774K), D55 (5500K), D60 (6000K), D65 (6500K), E (5454K) and F1-F12 (fluorescent light range 6430K - 3000K).

As can be seen from Figure 6.17, estimating the illuminant is a matter of searching the restricted space of the Planckian Locus, but finding the color of a light source is a much broader search that could span the entire chromaticity chart. Most techniques for determining the color of an illuminant estimate the illumination of a scene and therefore search the restricted space rather than the light color [CFB02, DWA04, HMP⁺08, LM71, BCF02, FHH01]. This is generally due to the

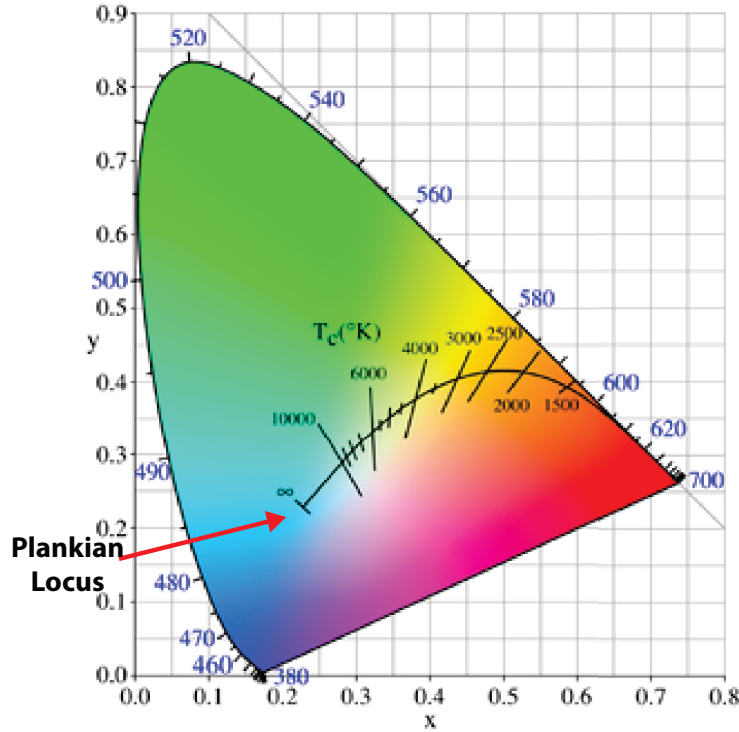


Figure 6.17: CIE 1931 Chromaticity diagram with Planckian Illuminant Locus.

fact that most common light sources fall within the Planckian Locus, including most Tungsten, Halogen, fluorescent, other filament based light bulbs, and various positions of the Sun. This is not always true however for LED-based lighting or lighting that is covered by a color spectral filter, which is commonly used in the entertainment industry, therefore the previous methods will incorrectly estimate these light colors. Therefore we have developed a light color estimation method that will determine the color of scene illuminants without the restriction of belonging to the group of blackbody radiators.

The most reliable and commonly used method for measuring the full range of color of a light source is via a spectrophotometer, which use diffraction gratings to separate the light sample into spectrum bands and measuring the light intensity at specific wavelengths [DeC97]. Spectrophotometers are highly specialized devices

and can cost several hundreds to thousands of dollars making them cost-prohibitive for most people. We present our alternative method, which is relatively simple and inexpensive using the Symmetric Lighting technique developed for this dissertation. Restating the Symmetric Lighting Equation 5.7 with the modification of globally minimizing the light color (e.g., L_2) for all pixels within a captured image, yields Equation 6.2 below:

$$\operatorname{argmin}_{L_2} \|[C_1 * (L_2 * \beta + (1 - \beta) * L_1)] - [C_2 * (L_1 * \beta + (1 - \beta) * L_2)]\| \quad (6.2)$$

Then if we assume that we know the color of light L_1 and calculated the β map for another previously known light source, then the process of re-acquiring two new images of the same scene with a new and unknown light source is needed. After acquiring the new images, the new light color estimate is found by minimizing the objective function above in Equation 6.2. This technique also alleviates the need to convert to the sRGB [IE99] color space from full spectrum values, which requires estimating in-gamut tri-stimulus values from those that may arise from a spectral conversion.

Results : Below in Table 6.1 is a sample of the values collected in our light color estimation tests. We used a synthetic scene rendered several time with the second light L_2 having different light color values. We then performed the calculations in Equation 6.2 to estimate the light value for L_2 in each image compared our estimate with the known ground truth. Best we calculate our light color estimate globally across all image pixels, the error associated with each image is averaged across the entire image. Areas within the image with high Signal-to-Noise (SNR) provided almost the exact answer, while other areas with lower SNR (i.e., high noise) provided

answers that varied significantly from the ground truth. As a means to quantify our comparison, we use an error metric based on Euclidian distance between the two color vectors, which is implemented in Matlab as $d = dist(X, Y)$. In our case, X is the ground truth color value for L_2 and Y is the estimated value and d is the error. Most academic work related to white balancing use the same error metric, in which values ≤ 10 are generally considered acceptable [Hor06]. The best white balance algorithms using highly restrictive assumptions routinely perform well with error values in the 5-10 range using proper training data sets [Hor06]. The average distance of the estimated color vector compared to the ground truth is provided in Table 6.1. It is not surprising that consistent across the images used in our tests, that values that showed significant deviations of light color estimate from the ground truth were those areas in the images that corresponded to higher levels of noise. If we assume that areas in the images with lower noise can estimate the light color correctly, then as an alternative to using the *average* value for the light estimate we could use the statistical *mode* as was done in [TEW99]. The mode values are located in Figure 6.2. We justify this by showing images with low to moderate noise levels produce the proper light color estimate in a majority of the pixels, allowing the statistical mode to be the prevailing and lowest error estimate. Future work on estimating the light color will focus on more varied scenes using real images. Real scenes were not used due to the high amount of noise, and the goal of this section was to test the light estimation algorithm for validity not noise analysis. These results show that this technique for light color estimation is a viable method.

Calibration : Image sensors generate values that are linearly proportional to the light sensed by the imaging pixels. The non-linearity of images is generally the result of the camera’s imaging system applying a nonlinear color transform to raw pixel values to compress saturated pixel values so they can be converted to a

Ground Truth (sRGB)	Estimate Color (sRGB)	Error (vector distance)
(91,138,240)	(93,130,242)	8.48
(160,160,50)	(168,172,51)	14.46
(115,50,110)	(113,55,118)	9.64
(175,50,70)	(173,48,64)	6.63
(175,210,212)	(177,202,212)	8.25
		Avg.

Table 6.1: Estimate of lighting color and the error using the average of the image pixel color rounded to the nearest integer.

Ground Truth (sRGB)	Estimate Color (sRGB)	Error (vector distance)
(91,138,240)	(95,141,236)	6.40
(160,160,50)	(165,154,53)	8.37
(115,50,110)	(118,52,112)	4.12
(175,50,70)	(172,51,73)	4.36
(175,210,212)	(176,208,210)	3.00
		Mode

Table 6.2: Estimate of lighting color and the error using the mode of the color values instead of the average.

standard RGB color space. This process is often referred to as Gamma Correction. To avoid applying linear operations to nonlinear pixel values that have been gamma corrected, it is necessary to apply an inverse gamma correction as seen in Figure 6.18 by multiplying the pixels values V_{in} by the power of the inverse of the gamma correction that was applied by the camera, in most cases a γ of 1/2.2 would be used in Equation 6.3. For display devices that expect a gamma correction, the original gamma power must be reapplied to the pixel values prior to being displayed by using a γ value of 2.2 for Equation 6.3.

$$V_{out} = V_{in}^{\gamma} \quad (6.3)$$

For all image capture and interpretation tasks many of the camera properties require calibration. Calibration involves solving for and correcting your cameras

intrinsic and extrinsic properties, lens aberrations, chromatic transforms, and tone response curves. The work presented in this chapter assumes that the workflow, the process of going from input image to rendered image, has a known mapping for color values and requires no other calibration steps. Often after camera calibration the images have an inverse gamma applied to the pixels values. The inverse gamma operation linearizes the pixels values making most subsequent operations also linear, which is commonly known as a linear workflow in many areas of image manipulation such as photography, image processing, computer vision, and rendering. Linearization is usually the first step prior to manipulating image or video in order to transform the image colors to an independent subspace of the original color space [RKAJ08]. This process usually entails performing an inverse gamma operation to not only images but also supplemental data such as look up tables, color values, and light estimates that will also be used in the image manipulation in order to maintain a linear workflow.

For this dissertation, any operation performed on images or image related data is assumed to be part of a linear workflow and therefore an inverse gamma correction has been applied prior to any processing. Unfortunately when acquiring images from cameras that have proprietary camera processing pipelines (most consumer level and professional level cameras), gamma correction is not the only non-linear operation applied to the image. In addition to nonlinear gamma, a camera's processing pipeline generally add several other operations to provide a more "satisfying" image for the consumers of the cameras. These operations map light values that enter the camera and convert them to non-linear pixel values, which is generally referred to the camera's response curve. The details of these operations are general not provided by the manufacturer of the cameras and are considered part of the trade secrets and distinguishing features of the cameras [RWPD05], but usually include some

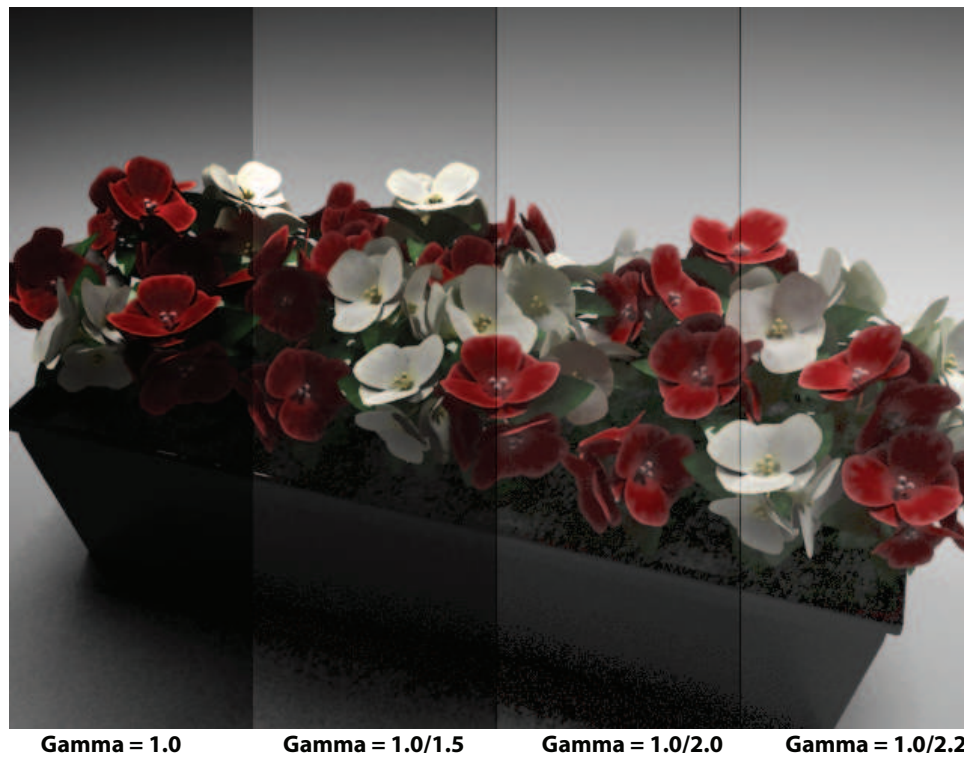


Figure 6.18: Image with dark regions that have had a Gamma correction applied. The values under the region specify how much correction was applied.

noise correction, white balancing, color correction, and color/gamut transforms. Because the operation details are not available and these nonlinear adjustments in the camera pipeline can introduce nonlinearities in an otherwise linear workflow, it is imperative for the response curve of the camera to be derived in order to account for any nonlinearity. The process of deriving a response curve has been the topic of several academic works [DM97, MN99] related to High Dynamic Range image capture techniques. The basic method for determining a camera’s response curve is to capture a set of images with a range of exposure settings then fit the luminance values to a known nonlinear curve. After fitting the luminance values to a known curve, the values can then be converted to a linear response analytically by inverting the nonlinear curve. Several different methods for deriving a camera’s response curve have been developed. For an overview of these methods the reader is referred to [RWPD05]. For the presented work, we use the technique developed by Debevec and Malik [DM97] for deriving the response curve.

6.4 Multi-Illuminant White Balance

6.4.1 Overview

Color constancy is the ability of the human visual system to mitigate the effects of different lighting conditions on an object’s reflectance such that the object’s perceived color is unchanged under different illumination. In photography, the process of color constancy is mimicked in a camera by applying a white balance method to a photograph after capture in order to compensate for colors cast by non-canonical lighting. For example, white balancing can remove the yellow tint that results from incandescent lighting or a blue cast caused by the early morning sun.

Most current research in white balance methods make the simplifying assumption

that the entire photographed scene is illuminated by a single light source. However, most real-life scenes contain multiple lights and the illumination recorded at each camera pixel is due to the combined influence of these light sources. Furthermore, most of today’s cameras and photo-editing software employ white balance techniques that make classic assumptions, such as the Grey World assumption [Buc80], which also assumes the scene being white balanced contains only a single light source. To solve the issue of multiple light source white balancing, it is common to segment the image and apply a traditional white balance technique to the different regions. In practice though, this can lead to hard transitions in white balanced images as neighboring regions can potentially estimate different light sources at region boundaries. Also, this solution does not account for partial lighting contributions from any light source thereby increasing the chance for incorrect white balancing.



Figure 6.19: Comparison of a scene with multiple light sources (yellow fill in the background and blue light in foreground). Left is the original image, center is our method, and right is the method of Ebner [Ebn03].

We provide a solution to white balancing images that contain two different colored light sources by utilizing the known color of each light and determining their appropriate contribution to each camera pixel by utilizing our Symmetric Lighting

method derived in Chapter 5. Our white balance technique has two main steps. First, we determine the β map for the image that is to be white balanced. We then use a simple projection technique to shift each illuminant color to a specified White point, such as CIE D50 or D65 as is commonly done when white balancing. Shifting the illuminant color is simply a series of vector multiplications. Because our algorithm uses mostly vector math, our technique is amenable to on-camera hardware implementation and real-time executions as is shown in the implementation in Section 6.4.3.

6.4.2 Related Work

Single light source white balance algorithms: Research into solving the color constancy and white balance problems has yielded many techniques. Classical color constancy techniques based on simple statistical methods include the Gray World [Buc80] and Max RGB [LM71] algorithms. These techniques make simplifying assumptions about scene or image statistics, which work only in special cases (i.e., scene reflectance averages to gray for Gray World assumption) but fail in more general application. Higher order statistical methods such as Generalized Grey World [FT04], the Grey-Edge algorithm [vdWGG07], and techniques employing natural image statistics [GG07] have been successful in solving illumination estimation problems. Several other classes of statistical methods that estimate scene illumination based on probabilistic assumptions such as Bayesian methods [FHH01], assumptions about the color spaces such as Gamut Mapping [For90] or color shifts [Ebn04], or using machine learning such as Neural Networks [CFB02]. The majority of the color constancy and white balance research, including the methods just mentioned, make the assumption that the scene is illuminated by a single light source.

Multiple light source white balance algorithms: A few methods have been pro-

posed for white balancing images that contain multiple light sources. However these techniques either require some user input or make restrictive assumptions that make them somewhat impractical to use. Hsu et al [HMP⁺08] proposed a method that determines the light mixture per pixel by modeling it as an alpha matting problem. They then use the matting Laplacian from [LLW06] to determine the mixture. Aside from being complicated and not amenable to hardware implementation, their technique also requires the user to know the light colors a priori as our technique does as well. Kawakami [KIT05] proposed a technique that determines the light color automatically for outdoor scenes only. They make the assumption that the scene consists only of lights whose color follow the Planckian black body radiator line (see Figure 6.17). In contrast to our technique, we do not restrict the range of possible light source colors to black body radiators but the full color gamut and our technique can be indoors as well as outdoors. The technique proposed by Ebner [Ebn03] is most similar to our method. This method can automatically determine the light color and because it employs only vector operations within pixel neighborhoods it is amenable to hardware implementation. However, unlike our method, this method does make the simplifying assumption that small pixel neighborhoods average to a gray color (also known as the Gray World assumption [Buc80]). Consequently, this method works well for scenes in which the gray world assumption is true but can produce incorrect white balancing when this assumption is not met, which is common. Our method makes no assumption about the average scene color locally or globally, does not require any user adjustments related to lighting, and does not require the user to know all the light colors a priori (if we assume the light color estimation method described in Section 6.3 is used to discover the single unknown light color).

6.4.2.1 Multiple Illuminants

Color constancy and white balance algorithms generally fall into two broad categories [Hor06]; the first class being techniques that determine the reflectance of objects within the scene in order to estimate illumination by reducing the complexity of inverting the light transport. The second class involves transforming an input image in order to discount the effects of non-canonical scene lighting. Our method falls within the latter classification and as such we are not concerned with recovering reflectance but instead in determining the contribution of each illuminant and discounting their effects.

If we assume for the moment that the scene is illuminated by two known illuminants that differ in color. Then using the interpolation method described in Equation 5.5 (rewritten below as Equation 6.4), we can assume that each pixel, located at (x,y) in image I , has a relative contribution of light from each illuminant. Equation 6.4 is the same formulation, generally known as a linear interpolation, used to describe the relative light contribution in our Symmetric Lighting method as well.

$$\hat{L} = \beta l_1 + (1 - \beta) l_2 \tag{6.4}$$

We note that the assumption relative contribution of each illuminant in Equation 6.4 is also made by Hsu et al. [HMP⁺08]. However there are four main differences between our work and Hsu et al.'s.; 1) our method differs in how we come to determine the value of β or the relative contributions of each illuminant l_1 and l_2 from that of Hsu et al.'s work (described in the related work Section 6.4.2), 2) unlike Hsu et al., we do not assume that the light colors are bound to the range of the black body radiators, 3) also we do not assume that all scenes contain at most two

lights as we previously described in Section 5.2.3.1, and 4) Hsu et al. categorizes specular reflection as errors and is therefore a limitation of their method, but we explicitly described in Chapter 5 the influence of specular reflection and the effect on estimating the β map and how to compensate. Similar to Hsu et al. though, we assume that the light colors are known but not their relative contribution.

Assuming the light contributions from l_1 & l_2 are conserved then we can say the $0 < \beta < 1$, and that all the light present in the scene comes from only these lights and not any other source (i.e., no florescence or emitters). $I_{x,y}$ is a result of the reflectance within the scene scaled by some factor G . The task of white balancing image I then becomes a two part task, a) determining the colors of l_1 and l_2 in RGB space, and then b) determining their relative contributions, β to pixel $I_{x,y}$. In the following section we develop techniques for accomplishing both these tasks.

6.4.3 Implementation

As described in Section 5.2.2, finding the shortest distance between the color line and the light line $\overline{l_1 l_2}$ requires minimizing a quadratic function per pixel. A faster approximation to minimizing the shortest distance is to project the color to the light line. This technique requires only a subtraction, addition, multiplication, and a dot product. Since these operations can be translated into simple hardware operations or GPU commands, they can be performed very fast. Furthermore, instead of using pixel neighborhoods as suggested in citation [HMP⁺08] we can perform this operation on a per-pixel basis with little or no additional performance cost. Figure 6.3 shows our implementation using pixel shader code from the GLSL shading language.

Listing 6.3: This GLSL code shows our method of performing white balance to an image with multiple illuminants. We assume that a β map exists for the image and the two light values are known. Then we determine the appropriate values for undoing the previous lighting and apply a new light with a D65 color.

```
/*Relighting
* Description: White balance an input image with
* multiple illuminants.
* Output: A fragment that corresponds to the input image
* with new lighting values.
* Written by: Clifford Lindsay, February 26, 2011.
* Department of Computer Science, WPI
*/
uniform sampler2D src_tex_unit0;
uniform sampler2D src_tex_unit1;
uniform vec3 l1;//old light 1
uniform vec3 l2;//old light 2

void main(){
    vec3 D65 = {1.0, 1.0, 1.0}; //approx noon sunlight
    //retrieve original image.
    vec3 c1 = texture2D(src_tex_unit0, gl_TexCoord[0].st).rgb;
    //retrieve Beta map image.
    vec3 beta = texture2D(src_tex_unit1, gl_TexCoord[0].st).rgb;
    //calculate the proportions of the old lights for each pixel
    vec3 I = beta.r*l1 + beta.b*l2;
    //shift light to CIE D65 then apply to image.
    vec3 wb = c1 * (1.0/I) * D65;
    gl_FragColor = vec4(wb, 1.0);
}
```

The description of our implementation assumes certain technical details, such as the image being white balanced is mapped to a quadrilateral so that each pixel/texel from the image is being passed into the pixel shader listed in Figure 6.3. Since the light and image colors are defined in a linear RGB color space, the operations are

performed in three dimensions thus are vectorized for efficiency and parallelism.

To demonstrate that our algorithm is amenable to real time execution, we implement it on an Android based Motorola Droid smart phone with OpenGL 2.0 rendering capabilities and a built-in camera. The smart phone has a Texas Instruments OMAP 3430 Arm Cortex A8 CPU running at 600 MHz with 256 MB of ram. The GPU on the smart phone is a PowerVR SGX 530 GPU. A customized camera application was developed that captured photographs from the built-in camera and transferred them to an OpenGL rendering application which performed the white balance operation and wrote the image to the phone's file system. The whole operation, from click to white balance preview, took less than 1 second with the bulk of the time spent on transferring the image from memory to the GPU (not the actual white balance render time).

6.4.4 Results

We now describe the results of testing our white balance method against the ground truth as well as other methods. Our test scenes were comprised of a variety of synthetic images with known illuminant colors. For the synthetic images, renderings were done in the V-Ray physically based renderer, which produced physically accurate direct and indirect lighting (global illumination). Every effort was made to use plausible light source colors in order to test the performance of our white balance algorithms on real-world illumination. We should point out that all prior work algorithms for white balance assumes that the color of the illuminants lie relatively close to the Planckian Locus illustrated in Figure 6.17. We significantly relax this assumption and assume that the light source color lie within the boundary of the sRGB gamut (although measurements can be taken if the values are provided in dominant wavelength form). In addition to expanded the colors of the light sources,

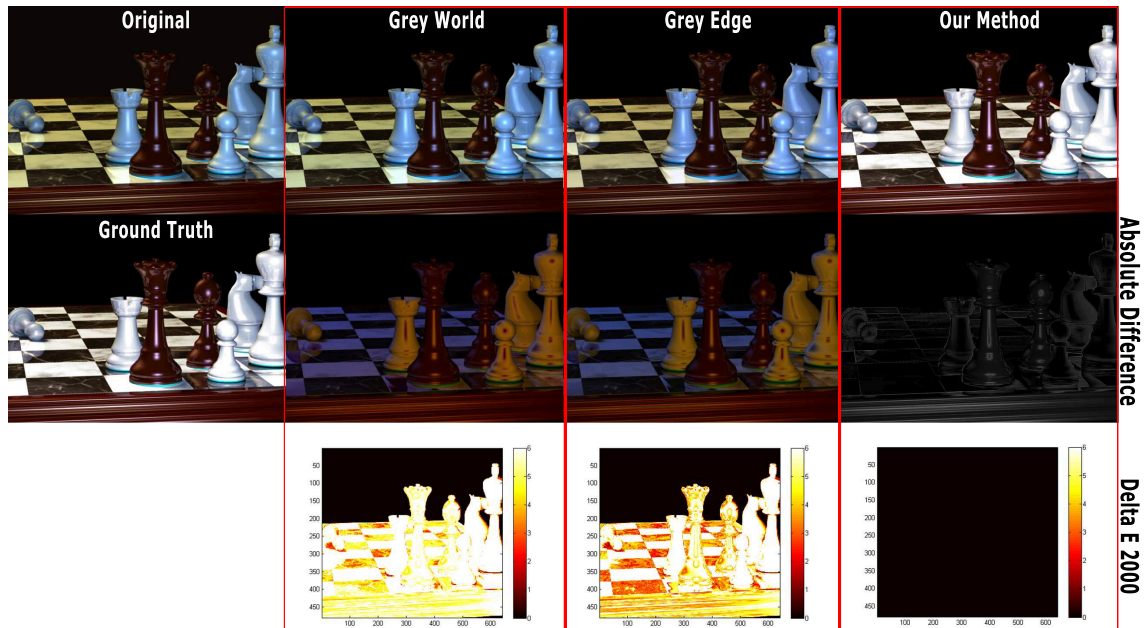


Figure 6.20: Results comparing our method, the Grey World method, and the Grey Edge method to the ground truth. The comparisons consists of a absolute difference images performed in Matlab using the *imabsdiff* function in the Imaging Toolbox and the CIE Delta E 2K function [LCR01]

we also produce good results with low-light images and with scenes that exhibit low-color complexity, on which white balance algorithms in general perform poorly with [Hor06]. Also, it is assumed that the values of the pixels within the image are properly exposed. In other words, that there does not exist any saturated pixels resulting in clamped pixel values. This is can cause errors to occur in the light color estimation as well as the relative light contribution.

In the presence of both specular and diffuse reflection our light contribution method works well as can be seen from Figure 6.20. The chess scene exhibits a high amount of specular highlights as well as diffuse reflection within areas lit by different lights. Notice also that the light colors in the input image for the chess scene are visually quite different. This scene is similar in setup to what might occur during a flash photograph under incandescent lighting. The foreground is lit by a cool blue light from the flash whereas the background is lit by a reddish-yellow

artificial light. This contrast in light color shows how robust our method is to colors outside the group of black body radiators, which is not possible with previous white balance algorithms. In Figure 6.20 we show the Chess scene with comparisons with the traditional and common used white balance methods gray world [Buc80] and the gray edge methods [vdWGG07].

6.4.5 Conclusion

In conclusion we have shown that our automatic white balance method can estimate dual lighting in complex scenes that exhibit both diffuse and specular highlights without the need for user assistance. We would like to point out that the prior technique of Symmetric Lighting developed in Chapter 5 makes what is generally a very difficult problem of white balancing multiple illuminant images relatively straightforward. Our white balance method relaxes many of the restrictive assumptions made in traditional white balance methods (i.e., illuminant color and number of lights) making our method the only method to our knowledge that can achieve white balancing scene with multiple lights of colors outside the black body radiators. In the future we would like to extend this method to three and possibly generalize it to N light sources. We believe that using the distance measure can also be applied to three light sources which is essentially a triangle representation in the linear RGB color cube. Also, we would like to make our light estimation technique to be more robust and applicable to images that contain both direct and indirect lighting.

6.5 User Studies

In this section we describe three user studies that were conducted to further evaluate the ideas implemented in this work. The first user study involved an evaluation of

the relighting theory and application in order to validate the use of Symmetric Lighting and as a relighting tool for previs (which we presented in Chapter 5). The second user study consisted of surveying experts in the field of Previsualization of their opinions on the need and desire for a previs virtual relighting tool that could be used on-set to perform set relighting (which we presented results for in Chapter 6). The third study was an Expert Review user study to validate the design and implementation of the our programmable camera back-end PCam (which we presented in Chapter 3). The WPI Institutional Review Board approved the procedures for all user studies. For reference, all user study questionnaires and scripts are listed in the Appendix Section C.

6.5.1 User Study #1: Relighting Evaluation

Recruitment : Eight WPI students, 6 males and two females between the ages of 18-32 years were recruited to participate in the user study. They performed relighting tasks in a library Tech Room and each study lasted about 30 minutes. We ran the study initially with only two people after which some issues with the software were discovered. So we delayed the remaining six participants until the software was fixed. Then subsequently, reran the study with same protocol with six additional participants for a total of eight. Our justification for using only eight participants comes from the claim that for usability testing, "five is enough" users is all that is required [NM90] for simple design iterations. For our application, this iteration only seeks to prove that our software provides at least a minimum level of intuitive usability. All of the participants in the study were students at WPI (2 graduate students, 6 undergraduates).

Study Design : This first user study was conducted with the goal of validating the design of our virtual relighting software. There were two primary objectives

associated with this study: 1) to determine if our software application met minimum usability engineering standards such as those described by [Nie94] and 2) determine if our software performed relighting in a realistic and qualitatively convincing way. Additionally, we surveyed the participant's knowledge about Previsualization in order to determine their understanding of the context for which the application was being developed.

To validate the usability of our software design the user was asked to perform a series of relighting tasks using the relighting application developed for this dissertation. After being consented at the very beginning of the user study, we asked "On a scale of 1-10, how familiar are you with movie Previsualization and relighting?" to gauge their understanding of previs. We did this to establish a baseline understanding of the concept of previs and relighting. We then proceeded to explain what previs is in general, then how we could incorporate relighting into previs. We then gave them a preview of the relighting application view video and walked the subjects through all the operations that could be performed using the application. We then asked them to independently perform a series of tasks related to relighting using our relighting application (see script in Appendix C.2.1). The participants were asked to manipulate the intensity, color, dispersion, and location of the lights. For changing the intensity, they were provided a slider which increased the intensity of the lights on a logarithmic scale. Several methods were provided to modify light colors including, a color wheel, a hue slider, and numerical values for RGB values (red, green, blue) and HSV values (hue, saturation, and values). For dispersion, the user could apply a paint brush like tool to the screen to adjust the lighting (i.e., light painting). Finally, the application allowed for small movements of the light via a translation slider (only one degree of freedom, translation in the x direction). They were allotted 10 minutes to perform the relighting tasks and were also given the

opportunity to experiment on their own with the application. The users were supervised while performing these tasks in case any difficulties were encountered with the software. Additionally, users were told that they could provide comments or ask questions during the time they performed the relighting tasks.

After performing the relighting tasks, the participants completed a twenty-item questionnaire regarding their experience with the relighting application. The first five questions of the questionnaire determined demographic information as well as computer familiarity and usage. The next set of questions focused on their understanding of computer graphics, photography, and pre-vis. Finally, the remaining questions were regarding the realism and capabilities of the relighting application. In particular, we asked their thoughts on how the application performed in manipulating the properties of the lights when relighting the scene.

Results : All of the participants were asked about their familiarity with computers and how much time they spend using them on a weekly basis, with the average being 34.5 hours per week (a potentially high average resulted because one person reported 108 hours per week). The rationale for asking this question was to assess their level of comfort with computers in general. Next questions were used to gauge their familiarity with computer graphics, movie lighting, and movie making in general. We asked them if they had either taken a computer graphics or photography classes and only three of them had (all three were computer graphics classes) and all but two rated their photography skills as amateur (lowest rank) and two rated themselves as intermediate (one above amateur). Also, we asked if they were familiar with virtual lighting concepts used in 3D rendering on a scale of 1-5 (1=none, 2=a little bit familiar, 3=fairly familiar, 4= good familiarity, and 5=excellent familiarity). Additionally, each subject had experience with either creating a 2D/3D rendering or movie. Figure 6.21 shows the responses to these questions from the

subjects.

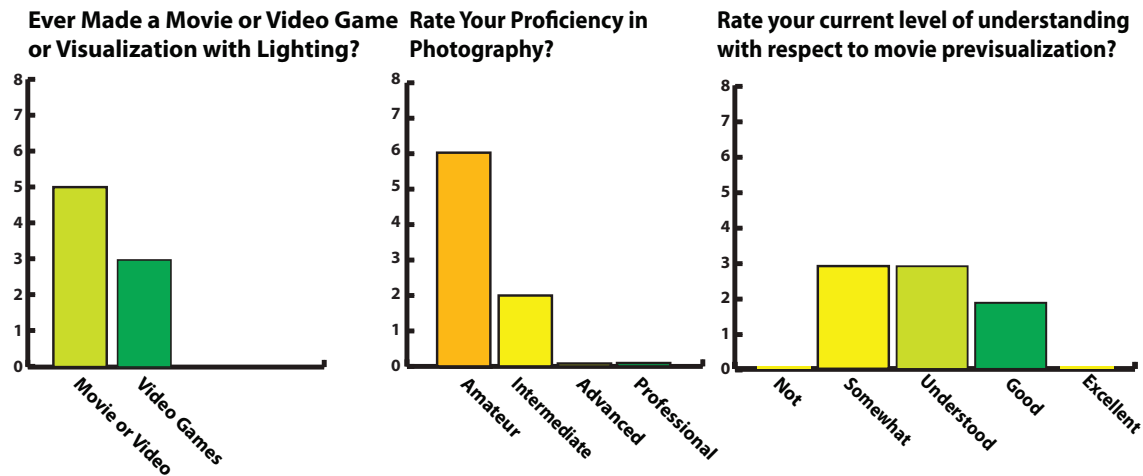


Figure 6.21: Demographic information and background information regarding the subjects in this user study.

We predicted that most participants would not have a full understanding of "movie previsualization relighting" but that once the concept was introduced that their level of understanding would increase rapidly. Everyone answered that they had little or no understanding of movie previs light at the beginning of the experiment but by the end of the experiment they expressed that their understanding had improved by indicating that they now have a good or somewhat of an understanding of movie lighting previs. Also, we predicted that as the participant becomes familiar with the goals of relighting, that the interface and the tools provided by our software would complement these goals making it easy to edit the lights appropriately as well as provide intuition about the light property they were changing. Our main reasoning for this is that relighting is just a method for manipulating how lights interact with an environment, which is similar to tasks people do on a daily basis when adjusting the lights in their environment. As indicated in Figure 6.22 most participants reported that it was extremely easy to change the light color. And when the participants were asked if the lighting property that was changed was intuitive

in the context of the relighting, most answered either fairly or very intuitive. In other words, when I changed a property of a light, such as color did the application and relit image produce the expected results. This indicates that the software application was designed in a manner that was consistent with participant’s intuition about light change. Also, when asked which property of the lighting changes were intuitive, most said changing color. Our design of the application appeals to this intuition by providing three-times as many UI widgets for manipulating light color than the other properties (3:1 ratio).

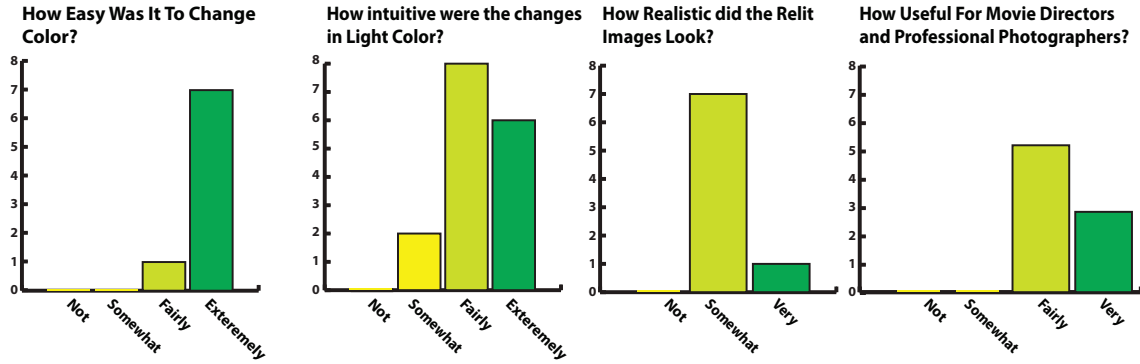


Figure 6.22: Qualitative questions regarding the use and realism of the virtual relighting software.

Additional questions were asked to determine how real the participants perceived the relighting to be and how useful the software might be for directors and professional photographers. When asked how real the relighting looked after a lighting change all indicated a moderate level of realism (see Figure 6.22). To gauge their understanding at this point we asked them to indicate what property they would manipulate in the interface if a director asked them to “*change the lighting to a cool morning*”, all indicated at least the color property was needed to be modified (which was correct). Our justification for asking a question comes from the notion that most people possess a certain level of Color or Lighting Vocabulary that we borrow from our perception of these colors in our environment [RKAJ08]. We also

asked for their opinion on how useful they thought this tool might be, when fully implemented, for directors or professional photographers. As shown in Figure 6.22, all participants indicated that it might be useful in the context of movie making or professional photography.

In the last questions we asked the participants for additional feedback or opinions that were not addressed by questionnaire, which they thought might be necessary to convey to us. Several participants suggested features to be added to the software such as more options for changing the lights and lighting presets. This feedback indicated to us that they were comfortable enough with the tool in the short amount of time that they wanted to go be given more advanced control over relighting. Their desire to have more controls also indicated to us that they had increased their level of understanding of relighting. The first participant discovered a bug in the software, which was subsequently fixed for the other users. Also, some of the participants indicated there were no changes in the shadows shape (as it was not featured in this version of the software). This may have contributed to a greater realism score if there had been more shadow manipulation. A couple of the participants praised the tool as being "cool", "very fun", and "pretty easy to understand". In conclusion, the data seems to suggest that the relighting tool was easy enough to understand in the limited amount of time given to learn the tool, provided all the functionality needed to perform the relighting tasks (although a couple of non-essential features were suggested), and achieved at least a moderate level of realism.

6.5.2 User Study #2: Previs Society Survey

Domain Expert User Study Design and Recruitment : This second user study was conducted with the goal of validating the utility of our virtual relighting software in the movie industry. Specifically, we wanted to gather the opinions of professionals

in the previs field regarding the usefulness of employing software for virtual relighting of movie sets during the previs stage of movie making. The Previsualization Society is an organization of professional movie makers and cinematographers who specialize in movie previs. We conducted a web-based survey that presented the questions from a questionnaire we developed (see Appendix [C.2.3](#)). We then posted a link to the web-based survey on a message board posting at the Previsualization Society's website (membership is required to post message) located at [\[Soc\]](#). The survey contained seventeen questions, which included demographics and questions regarding on-set lighting and relighting. The survey focused on questions about how lighting and relighting was performed on-set and their opinions about the utility of virtual relighting in improving on-set lighting previs.

Results of Domain Expert User Study : For our survey, we had responses from three professional movie makers. The first was a Previs Artist with three years of experience, the second was a Professor of Cinematography and a Director with 12 years of experience, and the third was a Technical Director with 7 years of experience (all males, ages: 50, 41, and 35). The opinion of these three professionals, varied widely on most matters, but all agreed that on-set lighting takes up a large portion of the on-set time to perform. When asked what percentage of on-set time is spent adjusting the lighting all three said greater than 75%, and given that the largest part of the movie making budget is spent for on-set, it would stand to reason that waiting for lighting to be modified adds considerable costs. They all stated that lighting is currently generally not a part of previs but all agree that it should be more prevalent in previs. All the subjects stated that the director is usually not directly responsible for lighting design and direction, but is more often delegated to other movie personnel such as cinematographers, technical directors (TD), or directors of photography. Also, the lights are physically modified by a lighting crew

and a person called a Gaffer who is generally the head electrician responsible for managing the lighting crew.

According to our survey participants, the director's central vision for a story does not usually revolve around a particular lighting setup. One participant of our survey offered his opinion that "[lighting] is usually not important because they don't understand lighting". This certainly cannot be true for all directors but does support the notion that directors may usually focus on other parts of the story, such as acting, choreography of action, and dialogue. This supports the previous statement that directors may delegate most of the lighting responsibility. In the context of light direction, all three participants suggested that a director would convey their ideas about the lighting or how they want the scene to have a particular look to other movie personnel. Then the lighting design and setup would be directed by that person and not the director of the movie. Additionally, the only people who seem to be concerned with previs of lighting at the moment are the visual effects crews (VFX) for the purpose of matching the current on-set lighting with that of the virtual lighting used to create visual effects in post-production. VFX crews often photograph the scene or take lighting measurements during filming [FZ09] and subsequently loaded into visual effects software for previewing.

Another participant stated that previs of on-set lighting may currently not occur because of a limitation of current technology. In response to the following question "is previs generally concerned with on-set lighting", the participant said "at the moment no as technology progresses, it may and should become more important especially with respect to digital workflows". This indicates a notion that there is currently a technology gap preventing relighting capabilities for on-set previs. This notion of a technology gap that will be filled in the near future seems to also be supported by the Visual Effects Handbook, which prophesizes that we should "expect

that lighting previs will have an increasing impact - and that cinematographers will play a role in that” [OZ10]. These statements seem to be in support of developing a tool for virtual relighting, such as the one developed for this dissertation.

In conclusion, according to our research the future of previs will include an on-set lighting component, which will probably be used by cinematographers and is already used by visual effects crews (in a limited fashion). As a cinematographer’s main tool consists of cameras, it makes sense to integrate these features into the camera or have a separate camera with these capabilities. As indicated by two of the participants, having the capability to preview lighting changes virtually would be useful, and furthermore having a camera with the ability to preview lighting changes would also be useful (one participant specifically said ”very useful”). This may also signify closer involvement from VFX crews with respect to lighting as they already perform such tasks, the gap between principle photography and some post-production activities will get shorter or even disappear as speculated in our motivating vision from Chapter 1. Since VFX crews already do lighting capture for post-production and cinematographers will in the future be using lighting previs to better plan lighting and produce better shots. The participants seem to indicate that having a programmable camera to run on-set lighting previs on would be valuable to the future of previs and filmmaking in general.

6.5.3 User Study #3: PCam Expert Review User Study

Expert User Study Design : In Chapter 4, we described a camera interface designed to complement our programmable camera architecture called PCam. To validate and refine our new camera user interface (UI) we performed a user study that tested the UI through a series of scripted tasks in order to discover usability issues. Instead of using random study participants to evaluate of camera UI, we performed an Expert

Review [Nie94]. Expert Reviews have been shown to uncover as many as three times the number of issues than general populations [ATT10, DR99] when testing is performed within the domain knowledge of the expert. Because we were developing a new type of camera interface which consisted of blending existing camera functions with a set of new features, we decided to utilize a professional photographer and movie maker as the expert for the study. Our expert was a twenty five year old professional photographer, artist, and independent filmmaker. She owns a small design firm specializing in television and web commercials as well as advertisement art. The main goal of our study was to ensure that our new UI preserved the usability of existing camera function⁵ while providing new functionality in an intuitive and usable way.

To achieve our goal, the study protocol focused on three areas for the evaluation: 1) testing "normal camera interface" of our UI, 2) testing the "advanced camera interface", 3) the study included a series of tasks for testing a separate desktop application called the Workbench. The "normal camera interface" incorporates the standard UI camera features that exist on most typical "Point and Shoot" cameras. The "advanced camera interface" includes new UI features for controlling and programming our PCam architecture described in Chapter 3. The Workbench application was developed for programming the low level features of PCam as well as developing new camera pipelines. In our testing protocol, we asked the expert to perform a series of predetermined camera operations that represented the typical camera usage, such as taking, reviewing, and editing photographs. This allowed us to focus the expert's feedback on the UI layout, how the UI was organized for performing typical camera tasks, and how well our UI preserved familiarity with existing camera UI and functionality. The evaluation was done on a camera system

⁵our normal camera functions were modeled after the typical Canon "Point and Shoot" camera interface design

simulated using a laptop with an embedded web camera. The study was recorded via video camera for review purposes.

For evaluating the "normal camera interface", the expert was instructed to perform a series of tasks related to normal camera functions, such as taking a picture, reviewing and adjusting captured images, and modifying the camera settings. In our study protocol, the expert was allowed to take as many pictures as was desirable. As the expert transitioned from one interface to another (i.e., picture capture, picture review, and camera adjust interfaces), we observed the experts actions and recording any usability issues. When the user completed all the tasks for this interface, a questionnaire was administered orally by the study administrator regarding the usability of the "normal camera interface". Although the evaluations were performed on a programmable camera, when evaluating the "normal camera interface", we restricted the camera to use only a traditional camera pipeline that produced photorealistic images.

The next focus area was the evaluation of the "advanced camera interface", in which the expert user was asked to perform a series of tasks relating to selecting and configuring alternative camera pipelines. At this point, the user was given access to the advanced interface and was given instruction on how to select and configure new camera pipelines. The "advanced camera interface" was pre-programmed with a set of alternative camera pipelines, such as Night Vision, edge detection, cartoon rendering, sepia tone mapping, and several others. The expert user was not asked to create or program any new pipelines at this time, but instead was asked to configure the parameters of the pre-programmed pipelines (descriptions of the implemented imaging algorithms and camera pipelines are located in the Appendix [B](#)). Once a new pipeline was selected and configured by the expert, the expert was asked to perform a series of picture acquisitions, picture reviews, and re-configuration of

the pipeline in the same fashion as was done the "normal camera interface". The user was allowed to acquire as many photographs as was desirable and when the expert completed taking pictures; a questionnaire was administered relating to the "advanced camera interface".

The third part of the user study involved the expert performing pipeline-creation tasks on the Workbench application, which involved building, previewing, and exporting new camera pipelines to our programmable camera. To create a new camera pipeline, the user must arrange a series of digital camera filters in sequential order within the Workbench application. Digital camera filters are small programs that are designed to modify the camera image, such as convert a color image to black and white, change the contrast of the image, or more complicated operations like converting the image to a cartoon rendering. The camera pipeline works by taking the input to the pipeline, which is an unmodified image, and passing it through a series of digital filters. After being modified by a filter, the modified image is then passed to the next filter until the end of the pipeline, producing the desired look of the image. The expert was asked to create a new pipeline using pre-programmed filters, which were available within the Toolbox area of the Workbench application. The expert user was not asked to perform any text-based programming of the digital filters. When the expert was sufficiently satisfied with how pipeline modified the input image, the expert was asked to export the pipeline to the camera. When the user completed evaluating the workbench application, a questionnaire related to the Workbench interface was administered. To conclude the study we asked several questions with respect to the interfaces as a whole.

Results of Expert User Study : The Expert User study provided informative feedback with respect to the design of all the UIs including the Workbench application. The expert felt that the organization and layout of the "normal camera interface"

was consistent with typical camera operations seen in commodity Point and Shoot cameras. The most prominent issue indicated by the expert was the lack of visual and audio feedback when taking pictures. For example, when taking a picture with film-based cameras, the mechanics of the shutter provided "click" sound and subtle vibrations. Digital cameras mimic this cue by providing a synthetic auditory "clicking" sound in conjunction with blacking out the preview screen momentarily to simulate a mechanical shutter action of closing and the re-opening. All other features of the "normal camera interface" performed in a manner consistent with other cameras.

For the advanced operations, such as selecting and tuning a new camera pipeline, the expert indicated that the additional functionality and layout did not over complicate the design and operation of the traditional camera functionality. Two issues were raised with respect to the interaction with the advanced functionality of the camera: 1) preserving the original image (traditional pipeline) was desirable, which was not part of the current design, and 2) making the normal interface aware of the advanced features. For the first issue, the expert indicated that she would prefer to view the result of the new pipeline in real-time as well as acquire the image of the new pipeline but also retain a copy of the unmodified photorealistic image as if it were rendered using the traditional camera pipeline. The second issue stemmed from the inability of the expert to change the camera pipelines from within the normal camera interface. Specifically, the expert suggested that users would want to have the capability to rapidly change the scene type as well as the camera pipeline, which is not possible with the current "normal camera interface". For example, currently the camera user has to first switch to the advanced camera interface, select a new pipeline, and then navigate back to the "normal camera interface", which can be time consuming. In other words, the expert suggested that we allow the "advanced

camera interface” to save new pipelines to the list of pre-existing scene types in the ”normal camera interface” as well as consumer cameras (e.g., snow scene, indoor, twilight, beach, fireworks, etc.). The expert indicated that it would ”seem natural that new camera pipelines would be available for selection within the normal camera interface under *Camera Settings* as a new *scene type*” (refer to Section 4.2 for more details about the interface features). This would integrate new camera pipelines into the category of new scenes, which is how the expert envisioned they would be perceived by users. This would alleviate any confusion about the technical details of camera pipelines to the average user by calling them ”new scene type”, to which most cameras user are somewhat familiar with. Additionally, this would also alleviate the need for additional steps for switching between different scenes and different camera pipelines, thus minimizing the number of steps needed to switch between camera pipelines and saving time when capturing time sensitive moments or events on-camera.

The majority of the constructive feedback and observations came from the expert when using the Workbench application. The expert was familiar with cameras, video editing, and various digital content creation tools but not programming, so was not able to create any new digital filters for creating pipelines. Creating new digital filters involved writing Camera Shaders (see Section 3.3 for more details) and although the expert was aware that these shaders could be written, the expert had no experience in doing so. Fortunately, the Workbench application provided several pre-programmed digital filters and within several seconds the expert was able to assemble a new pipeline as a few minutes of instructions on how to create the pipelines. When the expert was asked about the intuitiveness of the layout and design, the expert indicated it was ”easy to use and produced results rapidly”. One flaw in the layout was the fact that the user repeatedly tried to drag and

drop digital filters onto the preview window (instead of the pipeline widget) despite being instructed that the filters needed to be dropped on to the pipeline. When asked about the tendency to want to drop the filters on to the Preview Window, the expert indicated that the "it seemed like the logical place to drop the digital filter and I intuitively want to do that". This desire to want to drop the filters there probably stems from the visual layout and Direct Manipulation style of the application. The user also indicated the need for reversing or "undoing" actions, as was common in other content creation tools. Undo capability was indicated a missing feature that the expert thought would be highly desirable as part of the interface for creating new pipelines was primarily designed for users to rapidly modify the pipeline and view changes. The final constructive criticism from the expert was that it would be useful for the application to indicate the progressive changes each filter made by providing a thumbnail image for each filter indicating how the pipeline looked at various points in the pipeline. In other words, the expert would like to be able to view the result of each successive digital filter in the pipeline on the input image.

Chapter 7

Future Work

7.1 Expanded Scene Capture

Multi-view stereo capture techniques have improved to the point of accuracy that rivals that of laser scanners with sub-millimeter accuracy. Despite these advancements, Multi-view stereo capture can still suffer from the inability to properly match corresponding points due to edge discontinuities and occlusions [FTR⁺04, Fer06, FRC⁺05, GCS06, FRC⁺08]. Conversely, capturing edges only for the purpose of reconstructing geometry and surface appearance can be equally accurate as Multi-view stereo but result in large data sets (GBs), acquisition times in the hours, narrow vertical field-of-view (no viewpoint change in elevation), and additional capture hardware [CLS⁺06]. This project seeks to develop a 3D capture technique that is adequately accurate without generating large data sets and deals with point matching issues in Multi-view stereo methods.

In the future, we will develop a novel geometry capture technique that combines acquiring the Depth Map from Multi-view stereo with capturing Edge Maps from Multi-flash imaging (this concept is illustrated in Figure 7.1). Combining these

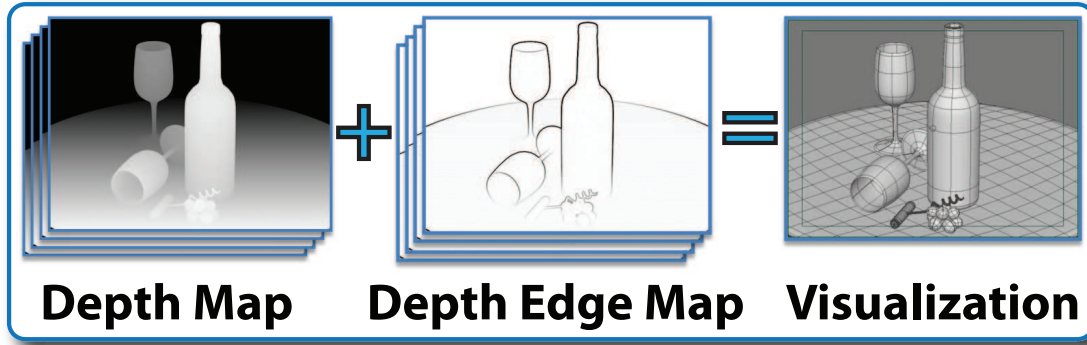


Figure 7.1: Approximate representation of the geometry from the depth map and depth edge map.

two approaches will allow for the capture of precise geometry without sacrificing the fidelity at depth edges, which is common problem with Stereo and Multi-view stereo capture techniques. Additionally, combining these approaches will decrease the acquisition time and sampling density needed for geometry capture exhibited in previous Multi-flash capture methods [CLS⁺06] while simultaneously increasing the vertical sampling density for appearance capture. Feris et al. [FRC⁺08] showed that this marriage of Stereo with Multi-flash imaging significantly enhanced passive stereo matching near discontinuities and specularities. They show that the error, using a root-mean-squared error (RMS) measure, decreases with window size without adding any significant processing time to the overall method. The rationale is that since Stereo and Multi-view stereo suffer from the same occlusion problems that extending this technique to Multi-view Stereo will also improve capture.

As we just described how Multi-flash can be employed to improve Multi-view stereo capture, we can also leverage the Multi-view camera to improve the surface appearance capture of Multi-flash imaging. Previous Multi-flash imaging techniques that capture the appearance of an object [LSC⁺06], simply sample along an arch around the object. This leads to sufficient sampling horizontally around the object but neglects sampling vertically, resulting only in the capture of empirical reflectance

functions and no explicit normals. To improve on previous surface appearance capture using Multi-flash imaging, we can leverage the greater vertical resolution of the Multi-view camera. In particular, we propose using the non-parametric BRDF capture approach of [AZK08] and the Spherical gradient approach for capturing Specular and Diffuse Normal Maps as in [MHP⁺07]. In addition, the proposed technique of Multi-view stereo and Multi-flash imaging will not require a turntable or other non-portable hardware; therefore it can be readily used outside the laboratory and in real-world scenes.

7.2 Relighting With Complex Lighting Environments

Generating complex lighting environments that are independent of a scene is desirable for producing realistic relighting. The fundamental quantity for lighting is the Light Field [Ger39], which is a vector field that quantifies light originating from all direction within a volume and can parameterized as a 4D function. Most techniques for games use environment maps which produce visibly plausible lighting only at the location the environment map was recorded. This is equivalent to only recording a single light vector [Arv94] within Light Field [Ger39], and not accurately capturing spatial variations or changes in lighting as objects move. For Previsualization and production rendering a full light field is desirable in order to aid in Global Illumination techniques such as Ray Tracing, Photon mapping, and Radiosity, which can accurately capture spatial variation in lighting. Capturing a "real-world" Light Field generally requires sophisticated light setups such a geodesic light emitting dome called the Light Stage [DHT⁺00a] and densely sampled lighting environments with precise measurements. These requirements make such techniques

impractical to outside of a laboratory in real-world lighting environments. Alternatively, lighting environments can be estimated with Light probes [Deb98b] or other photographic and Catadioptric camera [Nay97] techniques but are essentially equivalent to Environment maps. So the capture of many Light probes is necessary for estimation of complex lighting environments or Light fields. The major challenge is how to capture complex lighting environments that exhibit the real-world features such as spatially-varying, temporally-varying, and near and far field illumination in a feasible way without requiring complex light capture setups.

For future work, we propose to develop a method for approximating Incident Light Fields from a collection of online photographs. The proposed solution will aim to recreate complex lighting environments from collections of photographs by inferring the 3D location, direction, color, and intensity ranges of objects emitting or reflecting light within that scene. Additionally, the proposed technique will also develop a data structure and rendering algorithm that is GPU amenable to facilitate its use for relighting and 3D Previsualization. The main contribution of this proposed technique will have two parts. As far as we know, this will be the first technique that will attempt to recreate a lighting environment from unorganized photographs from a Community Photo Collection, allowing for the generation of complex lighting environments without the use of complicated recording devices and light reproduction techniques such as a Light Stage which was not previously possible. Secondly, by developing a new GPU amenable data structure and rendering algorithm, this technique will improve upon the existing environment mapping techniques by allowing for spatial and angular light variation which is not possible with existing real-time rendering techniques.

7.3 Surface Reconstruction Using Geometry Maps

The first step of visualizing the capture data is converting the raw data into a format that is amenable to rendering on graphics hardware. The reasoning behind this idea is that most of the processing and visualization will be accelerated using graphics hardware. Therefore, the raw data that we capture will be converted into a series of maps which will be the standard data structure for visualization. As shown in Figure 7.2 the objects within the scene will be represented by a data structure called a Geometry map. The surface details, such as normals, tangents, and bi-tangents will be stored in a normal map. The reflectance will be stored in a BRDF map and the textures will be stored in a texture map.

Geometry maps (also known as Geometry Images [GGH02]) store an approximation of a three-dimensional mesh (vertices, edge connectivity, faces) that has been sampled and reparameterized into two-dimensional representation. Most of the benefits of representing meshes as Geometry maps over a traditional irregular linear basis representation (triangles/quads) stems from the resampling of the meshes to create a completely regular structure within a simple $N \times N$ array of values [GGH02]. As a result, we can represent geometry as images and fully utilize the texture domain of graphics hardware in a similar fashion to that of Texture mapping and Normal mapping. Moreover the geometry representation can benefit from the same types of hardware acceleration that are utilized for texturing, such as compression, image processing and anti-aliasing may be useful for mesh related operations (see Figure 7.2 for a visual illustration of creating a Geometry map).

Traditionally, Multi-view capture techniques, like those being proposed, generally represent a reconstructed scene in one of four ways; voxels, level-sets, polygon meshes, or depth maps. For reasons of compatibility with Multi-flash imaging,

depth maps are our representation of choice. Often these representations can be sufficient for rendering as is or the representations can be converted to an alternate data structure such as a polygon mesh (if it is not already in this format. Polygon mesh construction can be accomplished through a series of computationally intensive steps; sampling, normal integration, surface reconstruction, vertex coloring, and smoothing the mesh. Since the desired final format of the geometry is the Geometry map, further sampling and reparameterization is required to convert a polygon mesh to Geometry map. Both of these steps (depth map to polygon mesh, and polygon mesh to Geometry map) can be computationally expensive and add significant time to the preprocessing stage. There is also redundant work that is performed by both techniques, specifically the sampling required by each technique. Additionally, errors that originate in the polygon mesh creation process will propagate through the Geometry mapping process.

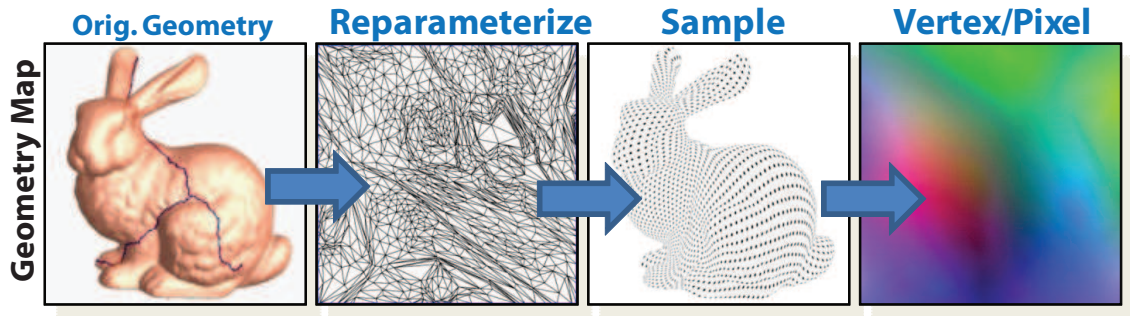


Figure 7.2: The Geometry map creation process outlined in [GGH02]

For future work we plan to alleviate the unnecessary sampling and redundant reconstruction, a regular sampling and subsequent parameterization of the depth and edge data will be performed directly on the depth map to create a Geometry map. In essence we will short circuit the previously outlined steps by creating the Geometry map straight from the captured data. To my knowledge this is the first time such a process has been conceived using depth and edge data to create

Geometry maps. In addition to a potential preprocessing speedup, the Geometry map representation will allow for smoothing, noise reduction, and some rendering operations that would normally be done in mesh space to be done in image space on the graphics hardware. In particular, ambient occlusion operations have been shown to be feasible in image or screen space [SA07, BS09]. This technique would give an approximate solution to global illumination without the need to do costly ray tracing, such as soft and self shadowing.

7.4 PCam Version 3.0

There are two limitations with the current version of PCam, first is the bottleneck of transferring image frames from image sensor to GPU and the second is the lack of high-quality optics and lenses. At the moment, the major bottleneck for the GPU computing is actually loading data into GPU memory from host or CPU memory, which is 10-100 x times slower than loading data from GPU memory to kernel register memory. This assumes that the desired data is in CPU memory, which in terms of camera processing; the data has to travel across a buss first before residing in host memory, making it more of a bottleneck. It would be desirable for the next version of PCam to avoid these bottlenecks by writing image frames directly from the image sensor to GPU memory. Currently there are no architectures which support such an operation. There are some good prospects; mainly Nvidia has a product line that is designed for video capture which uses a complimentary GPU board which can write directly to GPU memory. This system though still requires the original frame to be transferred over analog line (RCA or BNC connections) to the complimentary card, and then subsequently written to the GPU memory. This setup is mainly geared toward the encoding of video frames to compressed format.

This system is available for desktops computers only and is not available to the general public and is probably cost prohibitive. Another limitation with current version of PCam is the lack of high-quality optics for cameras that have GPUs (and vice-versa). A desirable improvement for the next version of PCam is to include higher-quality optics onboard of the new PCam version. Currently, the version of PCam is implemented on a mobile phone system which tends to have flat little or no optics beyond a simple flat lens. The system lacks the ability to provide real focus capabilities and zoom is only provided digitally. The improvement of optics on mobile phones or the use of an amenable platform with higher quality optics can improve the image quality of current capabilities of PCam.

Chapter 8

Conclusions

This dissertation presented a new method for interactive previs, which automatically captures set lighting and provide interactive manipulation of cinematic elements to facilitate movie maker’s artistic expression, validate cinematic choices, and provide guidance to production crews. This dissertation describes a new hardware and software previs framework that enables interactive visualization of on-set relighting and post-production elements. In this dissertation we describe a three-tiered framework which is our main contribution; 1) a novel programmable camera architecture that provides programmability to low-level features and a visual programming interface, 2) a series of new algorithms that analyze and decompose the scene photometrically, and 3) a previs component that leverages the previous to perform interactive rendering and manipulation of the photometric and computer generated elements.

The architecture of modern digital cameras can be decomposed into three stages; while the technology in each camera stage has evolved the core design of digital cameras has not changed. To address this issue we designed a programmable camera backend, called PCam, which can implement many of the state-of-the-art computational photography (CP) algorithms on-camera. PCam allows users to redefine,

re-order, and modify the camera pipeline. User developed programs for use on our programmable camera, which we call Camera Shaders, are executed on-camera during image capture. Since PCam can be easily reprogrammed for different specific tasks, a single camera can mimic the capabilities multiple specialized cameras through software programming instead of hardware.

Additionally, we designed a new User Interface (UI) for use with programmable cameras that addresses the shortcomings of both the previously mentioned development approaches for applying CP techniques. The our UI provides a visual programming interface that abstracts all the technical details of programmable camera pipelines to simply arranging a sequence of visual blocks. Each visual block, which we call a digital filter (also called Camera Shaders), is automatically mapped to an atomic operation and its underlying code that performs the operation in the camera pipeline. Our UI provides an extensive library of pre-programmed filters, which the user can easily add to existing or newly created camera pipelines, without requiring programming knowledge or an understanding of the technical details of the inner workings of the camera. Additionally, our interface provides immediate WYSIWYG feedback of filter and pipeline changes, thereby alleviating the shortcomings of the previous CP development approaches by allowing quicker turn-around time for creating and editing programmed camera pipelines (seconds vs. hours).

In terms of Photometrics, this dissertation presented two techniques for analyzing and manipulating the properties of light within a captured scene. Each new technique focuses on a different property of light, intensity (color) and distribution of photons. Capturing a light's influence on a scene is generally an ill-posed problem and often difficult to do in the presence of an unknown geometry, reflectance, and appearance for each the object within the scene. The first technique estimates the proportion of light each light source contributes to illuminating a point in the

scene using an active illumination technique called Symmetric Lighting. The second technique builds upon the first by incorporating second-order statistics to provide the capability to manipulate the distribution of photons for each light.

Lastly, for Relighting, an interface was designed to bring all the parts previously mentioned into one application. It has also been shown the Photometric work presented here can also be utilized for several outstanding problems, such as multiple-illuminant white balance, and light source color estimation.

In conclusion, this dissertation provides an end-to-end solution for performing relighting of scenes with multiple, real-world lights utilizing our new programmable camera architecture, a novel Symmetric Lighting technique, and a user interface for performing the relighting. We have shown that Symmetric Lighting can be reformulated to solve other lighting and image-based problems, such as light color determination and multi-illuminant white balancing. Symmetric Lighting is a simple but effective technique that can be quickly implemented and utilized for many relighting applications but provides a level of sophistications that proves useful in many areas.

Bibliography

- [AB91] Edward H. Adelson and James R. Bergen. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, pages 3–20. MIT Press, 1991. 135
- [Ado09] Adobe. Adobe flex 3. <http://www.adobe.com/products/flex/>, 2009. 55
- [AMH02] Tomas Akenine-Moller and Eric Haines. *Real-Time Rendering*. A. K. Peters, 2 edition, 2002. 44
- [ARC06] Amit Agrawal, Ramesh Raskar, and Rama Chellappa. Edge suppression by gradient field transformation using cross-projection tensors. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2301–2308, Washington, DC, USA, 2006. IEEE Computer Society. 123, 124, 127, 128
- [ARNL05] Amit Agrawal, Ramesh Raskar, Shree K. Nayar, and Yuanzhen Li. Removing photography artifacts using gradient projection and flash-exposure sampling. *ACM Transactions on Graphics*, 24(3):828–835, August 2005. 124, 128
- [Arv94] James Arvo. The irradiance jacobian for partially occluded polyhedral sources. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 343–350, New York, NY, USA, 1994. ACM. 203
- [ATP⁺10] Andrew Adams, Eino-Ville Talvala, Sung Hee Park, David E. Jacobs, Boris Ajdin, Natasha Gelfand, Jennifer Dolson, Daniel Vaquero, Jongmin Baek, Marius Tico, Hendrik P. A. Lensch, Wojciech Matusik, Kari Pulli, Mark Horowitz, and Marc Levoy. The frankencamera: an experimental platform for computational photography. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 29:1–29:12, New York, NY, USA, 2010. ACM. 27, 36, 66, 67

- [ATT10] W. Albert, T. Tullis, and D. Tedesco. *Beyond the Usability Lab: Conducting Large-Scale User Experience Studies*. Morgan Kaufmann. Elsevier Science, 2010. 195
- [AW90] Gregory D. Abram and Turner Whitted. Building block shaders. *SIGGRAPH Comput. Graph.*, 24:283–288, September 1990. 76
- [AZK08] N. Alldrin, T. Zickler, and D. Kriegman. Photometric stereo with non-parametric and spatially-varying reflectance. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008. 203
- [Bay76] Bryce E. Bayer, 1976. US Patent Number 3,971,065, Assigned to Eastman Kodak Company by the US Patent Office. 44
- [BBS00] R.S. Berns, F.W. Billmeyer, and M. Saltzman. *Billmeyer and Saltzman’s principles of color technology*. Wiley-Interscience publication. Wiley, 2000. 163
- [BCF02] K. Barnard, V. Cardei, and B. Funt. A comparison of computational color constancy algorithms. I: Methodology and experiments with synthesized data. *IEEE Transactions on Image Processing*, 11:972–984, September 2002. 169
- [Bea09] Beagleboard. The beagleboard is an ultra-low cost, high performance, low power omap3 based platform designed by beagleboard.org community members. <http://beagleboard.org/>, 2009. 53
- [bEP11] Warner brothers, HBO Entertainment, and Fox Searchlight Pictures. Movie collage., April 2011. 17
- [BFH⁺04] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for gpus: stream computing on graphics hardware. *ACM Trans. Graph.*, 23(3):777–786, 2004. 50
- [Bir06] J. Birn. *Digital lighting and rendering*. [digital] Series. New Riders, 2006. 113, 137, 139
- [BL79] S. Beucher and C. Lantuejoul. Use of watersheds in contour detection. In *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation, Rennes, France.*, September 1979. 119
- [BN76] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, 1976. 88

- [Boe07] Steve Boelhouwer. Machinima gets a day job - the emerging use of game technology in feature films. web article on cinema without borders, Feb. 2007. [7](#)
- [Bro99] Warner Brothers. Matrix storyboards. www.whatisthematrix.com, 1999. [4](#), [7](#)
- [BS09] Louis Bavoil and Miguel Sainz. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09: SIGGRAPH 2009: Talks*, pages 1–1, New York, NY, USA, 2009. ACM. [207](#)
- [BT78] H.G. Barrow and J.M. Tenenbaum. Recovering intrinsic scene characteristics from images. *Computer Vision System*, 1978. [116](#), [122](#)
- [Buc80] G. Buchsbaum. A spatial processor model for object colour perception. *Journal of the Franklin Institute*, 310(1):1 – 26, 1980. [93](#), [177](#), [178](#), [179](#), [185](#)
- [BZCC10] Pravin Bhat, C. Lawrence Zitnick, Michael Cohen, and Brian Curless. Gradientshop: A gradient-domain optimization framework for image and video filtering. *ACM Transactions on Graphics*, 29(2):10:1–10:14, March 2010. [28](#), [124](#)
- [CFB02] Vlad C. Cardei, Brian Funt, and Kobus Barnard. Estimating the scene illumination chromaticity by using a neural network. *J. Opt. Soc. Am. A*, 19(12):2374–2386, 2002. [169](#), [178](#)
- [CIE87] CIE. International lighting vocabulary, 1987. [169](#)
- [CLL05] Sek M. Chai and Abelardo Lopez-Lagunas. Streaming i/o for imaging applications. In *CAMP '05: Proceedings of the Seventh International Workshop on Computer Architecture for Machine Perception*, pages 178–183, Washington, DC, USA, 2005. IEEE Computer Society. [35](#)
- [CLS⁺06] Daniel Crispell, Douglas Lanman, Peter G. Sibley, Yong Zhao, and Gabriel Taubin. Beyond silhouettes: Surface reconstruction using multi-flash photography. In *3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 405–412, Washington, DC, USA, 2006. IEEE Computer Society. [201](#), [202](#)
- [CPGN⁺07] Ewen Cheslack-Postava, Nolan Goodnight, Ren Ng, Ravi Ramamoorthi, and Greg Humphreys. 4d compression and relighting with high-resolution light transport matrices. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 81–88, New York, NY, USA, 2007. ACM. [148](#)

- [DBB02] Philip Dutre, Kavita Bala, and Philippe Bekaert. *Advanced Global Illumination*. A. K. Peters, Ltd., Natick, MA, USA, 2002. 90
- [Deb98a] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 189–198, New York, NY, USA, 1998. ACM. 22, 121, 146
- [Deb98b] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA, 1998. ACM. 88, 140, 204
- [DeC97] Casimer DeCusatis. *Handbook of Applied Photometry*. Aip Press, 1997. 85, 86, 170
- [Des03] Bill Desowitz. The previs gospel according to mcdowell and frankel. *Animation World Network*, September 2003. 5
- [DHT⁺00a] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the reflectance field of a human face. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 145–156, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 87, 93, 110, 203
- [DHT⁺00b] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the reflectance field of a human face. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 145–156, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 111
- [DM97] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 369–378, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 176
- [Doy02] Audrey Doyle. The power of previz. *Computer Graphics World*, 25(7):5, July 2002. 9

- [DR99] J.S. Dumas and J. Redish. *A practical guide to usability testing*. Lives of Great Explorers Series. Intellect Books, 1999. 195
- [DvGNK99] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Trans. Graph.*, 18:1–34, January 1999. 87
- [DWA04] Ron O. Dror, Alan S. Willsky, and Edward H. Adelson. Statistical characterization of real-world illumination. *Journal of Vision*, 4(9), 2004. 90, 169
- [DWT⁺02] Paul Debevec, Andreas Wenger, Chris Tchou, Andrew Gardner, Jamie Waese, and Tim Hawkins. A lighting reproduction approach to live-action compositing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 547–556, New York, NY, USA, 2002. ACM. 22, 85, 86, 145
- [Dye01] Charles R. Dyer. Volumetric scene reconstruction from multiple views. In *Foundations of Image Understanding*, pages 469–489. Kluwer, 2001. 87
- [Ebn03] Marc Ebner. Combining white-patch retinex and the gray world assumption to achieve color constancy for multiple illuminants. In *Pattern Recognition*, volume 2781 of *Lecture Notes in Computer Science*, pages 60–67. Springer Berlin / Heidelberg, 2003. 177, 179
- [Ebn04] Marc Ebner. Color constancy using local color shifts. In *ECCV (3)*, pages 276–287, 2004. 178
- [Ebn08] Marc Ebner. Gpu color constancy. *journal of graphics, gpu, and game tools*, 13(4):35–51, 2008. 92
- [ECJ⁺06] Per Einarsson, Charles-Felix Chabert, Andrew Jones, Wan-Chun Ma, Bruce Lamond, Tim Hawkins, Mark Bolas, Sebastian Sylwan, and Paul Debevec. Relighting human locomotion with flowed reflectance fields. In *Rendering Techniques 2006: 17th Eurographics Workshop on Rendering*, pages 183–194, June 2006. 11
- [ED04] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM Trans. Graph.*, 23:673–678, August 2004. 116, 149
- [ERDC95] Jr. Edward R. Dowski and W. Thomas Cathey. Extended depth of field through wave-front coding. *Appl. Opt.*, 34(11):1859–1866, 1995. 33

- [Fai05] M D Fairchild. *Color Appearance Models*, volume 8. Addison Wesley Longman, Inc., 2005. 163, 169
- [Fer98] Bill Ferster. Idea editing: Previsualization for feature films. *POST Magazine*, 04, 1998. 3
- [Fer06] Rogerio Schmidt Feris. *Detection and modeling of depth discontinuities with lighting and viewpoint variation*. PhD thesis, Santa Barbara, CA, USA, 2006. Adviser-Turk, Matthew. 24, 116, 201
- [FH09] Jiří Filip and Michal Haindl. Bidirectional texture function modeling: A state of the art survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31:1921–1940, November 2009. 87
- [FHH01] Graham D. Finlayson, Steven D. Hordley, and Paul M. Hubel. Color by correlation: A simple, unifying framework for color constancy. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1209–1221, 2001. 169, 178
- [FLW02] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 249–256, New York, NY, USA, 2002. ACM. 22, 121, 122, 146
- [For90] D. A. Forsyth. A novel algorithm for color constancy. *Int. J. Comput. Vision*, 5(1):5–36, 1990. 178
- [FRC⁺05] Rogerio Feris, Ramesh Raskar, Longbin Chen, Kar-Han Tan, and Matthew Turk. Discontinuity preserving stereo with small baseline multi-flash illumination. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 412–419, Washington, DC, USA, 2005. IEEE Computer Society. 87, 201
- [FRC⁺08] Rogerio Feris, Ramesh Raskar, Longbin Chen, Karhan Tan, and Matthew Turk. Multiflash stereopsis: Depth-edge-preserving stereo with small baseline illumination. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(1):147–159, 2008. 87, 201, 202
- [FT04] Graham D. Finlayson and Elisabetta Trezzi. Shades of gray and colour constancy. In *Color Imaging Conference*, pages 37–41, 2004. 93, 178
- [FTR⁺04] Rogerio Feris, Matthew Turk, Ramesh Raskar, Karhan Tan, and Gotsuke Ohashi. Exploiting depth discontinuities for vision-based fingerspelling recognition. In *CVPRW '04: Proceedings of the 2004*

- Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 10*, page 155, Washington, DC, USA, 2004. IEEE Computer Society. 201
- [FZ09] C.L. Finance and S. Zwerman. *The Visual Effects Producer: Understanding the Art and Business of VFX*. Focal Press. Elsevier Science, 2009. 10, 193
- [GCHS05] Dan B. Goldman, Brian Curless, Aaron Hertzmann, and Steven M. Seitz. Shape and spatially-varying brdfs from photometric stereo. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 341–348, Washington, DC, USA, 2005. IEEE Computer Society. 87, 93
- [GCS06] Michael Goesele, Brian Curless, and Steven M. Seitz. Multi-view stereo revisited. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2402–2409, Washington, DC, USA, 2006. IEEE Computer Society. 201
- [Ger39] A. Gershun. The light field. *Journal of Mathematics and Physics*, Vol. XVIII:51–151, 1939. 203
- [GG07] Arjan Gijsenij and Theo Gevers. Color constancy using natural image statistics. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2007. 93, 178
- [GGH02] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. *ACM Trans. Graph.*, 21(3):355–361, 2002. 205, 206
- [GH00] Reid Gershbein and Pat Hanrahan. A fast relighting engine for interactive cinematic lighting design. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 353–358, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 22, 147
- [Gla09] Glade. Glade - a user interface designer. <http://glade.gnome.org/>, 2009. 55
- [Gno09] Gnome. Gtk+ is a highly usable, feature rich toolkit for creating graphical user interfaces which boasts cross platform compatibility and an easy to use api. <http://www.gtk.org/>, 2009. 55
- [Goo10] Google. Android operating system. <http://www.android.com/>, May 2010. 62

- [Gro08] Khronos Group. OpenGL ES specification - the standard for embedded accelerated 3d graphics. <http://www.khronos.org/registry/gles/specs/>, August 2008. 62
- [Gro11] Chaos Group. V-ray for 3ds max and maya. <http://www.chaosgroup.com/en/2/vray.html>, April 2011. ver 1. 158
- [GSHG98] Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. The irradiance volume. *IEEE Comput. Graph. Appl.*, 18(2):32–43, 1998. 88
- [GW00] W. S. Stiles Gnther Wyszecki. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley, 2nd edition edition, 2000. 156
- [Hae92] Paul Haeberli. Synthetic lighting for photography. web, January 1992. <http://www.graficaobscura.com/synth/index.html>. 146
- [HMP⁺08] Eugene Hsu, Tom Mertens, Sylvain Paris, Shai Avidan, and Frédo Durand. Light mixture estimation for spatially varying white balance. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–7, New York, NY, USA, 2008. ACM. 92, 169, 179, 180, 181
- [Hor06] Steven D. Hordley. Scene illuminant estimation: Past, present, and future. *Color Research & Application*, 31(4):303–314, 2006. 89, 92, 93, 172, 180, 184
- [HPB06a] Miloš Hašan, Fabio Pellacini, and Kavita Bala. Direct-to-indirect transfer for cinematic relighting. *ACM Trans. Graph.*, 25(3):1089–1097, July 2006. 22, 147
- [HPB06b] Miloš Hašan, Fabio Pellacini, and Kavita Bala. Direct-to-indirect transfer for cinematic relighting. *ACM Trans. Graph.*, 25:1089–1097, July 2006. 135, 155
- [HS98] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '98, pages 39–ff., New York, NY, USA, 1998. ACM. 22, 121, 146
- [IBM01] IBM. Eclipse. <http://www.eclipse.org/>, November 2001. 62
- [(IE99] International Electrotechnical Commission (IEC). Iec # 61966-2-1:1999 # srgb # specifications. IEC Specification, 2 1999. 171

- [Jen09] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2009. 85
- [JLMM05] R. Johansson, L. Lindgren, J. Melander, and B. Moller. A multi-resolution 100 gops 4 gpixels/s programmable cmos image sensor for machine vision. *IEEE Journal of Solid-State Circuits*, 40(6):1350–1359, 6 2005. 33
- [JO06] Jin H. Jung and Dianne P. O’Leary. Cholesky Decomposition and Linear Programming on a GPU. Master’s thesis, University of Maryland, 2006. 106
- [JOH07] DAVID M. JOHNSON. ‘pre-viz’ carves out niches on set. *Variety Magazine*, Online:1, April 2007. 4, 5, 8
- [JW92] C. Jiang and M.O. Ward. Shadow identification. *CVPR*, 92:606–612, 92. 119
- [Kaj86] James T. Kajiya. The rendering equation. In *SIGGRAPH ’86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM. 90, 114
- [Kat05a] Steve D. Katz. Charting the stars v.3. *Millimeter*, 04, 2005. 6
- [Kat05b] Steven D. Katz. Is realtime real? *Millimeter*, April, 2005. 6
- [KBD07] Jan Kautz, Solomon Boulos, and Frédo Durand. Interactive editing and modeling of bidirectional texture functions. *ACM Trans. Graph.*, 26, July 2007. 87
- [KDR⁺02] Ujval Kapasi, William J. Dally, Scott Rixner, John D. Owens, and Brucek Khailany. The Imagine stream processor. In *Proceedings 2002 IEEE International Conference on Computer Design*, pages 282–288, September 2002. 41, 42
- [KIT05] Rei Kawakami, Katsushi Ikeuchi, and Robby T. Tan. Consistent surface color for texturing large objects in outdoor scenes. In *ICCV ’05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1200–1207, Washington, DC, USA, 2005. IEEE Computer Society. 92, 179
- [KK09] Min H. Kim and Jan Kautz. Consistent scene illumination using a chromatic flash. In *Proc. Eurographics Workshop on Computational Aesthetics (CAe 2009)*, pages 83–89, British Columbia, Canada, 2009. Eurographics. 149

- [KRD⁺03] Ujval J. Kapasi, Scott Rixner, William J. Dally, Bruce Kailany, Jung Ho Ahn, Peter Mattson, and John D. Owens. Programmable stream processors. *IEEE Computer*, pages 54–62, August 2003. 41
- [KSK92] Gudrun J. Klinker, Steven A. Shafer, and Takeo Kanade. Color. chapter The measurement of highlights in color images, pages 309–334. Jones and Bartlett Publishers, Inc., , USA, 1992. 111
- [KSS02] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 291–296, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association. 148
- [Lab11] Rosco Laboratories. 52 harbor view, stamford, ct usa, 06902. <http://www.rosco.com/>, 2011. 156
- [LAW09] Clifford Lindsay, Emmanuel Agu, and Fan Wu. P-cam:a programmable architecture for digital camera back-ends. Technical Report WPI-CS-TR-09-11, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609-2280, October 2009. 24
- [LCR01] M. R. Luo, G. Cui, and B. Rigg. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Research & Application*, 26(5):340–350, 2001. 159, 163, 184
- [Lev10] M. Levoy. Experimental platforms for computational photography. *Computer Graphics and Applications, IEEE*, 30(5):81–87, September 2010. 64
- [LFDF07] Anat Levin, Rob Fergus, Frédo Durand, and William T. Freeman. Image and depth from a conventional camera with a coded aperture. *ACM Trans. Graph.*, 26(3):70, 2007. 32
- [Lin10] Clifford Lindsay. Personal communications with paul debevec, academy award winner and associate director of usc institute for creative technologies, los angeles, ca., 2010. 18, 19, 86
- [LKG⁺03] Hendrik P. A. Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Image-based reconstruction of spatial appearance and geometric detail. *ACM Trans. Graph.*, 22(2):234–257, 2003. 87, 93
- [LLW06] Anat Levin, Dani Lischinski, and Yair Weiss. A closed form solution to natural image matting. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern*

- Recognition*, pages 61–68, Washington, DC, USA, 2006. IEEE Computer Society. 179
- [LLW⁺08] Chia-Kai Liang, Tai-Hsu Lin, Bing-Yi Wong, Chi Liu, and Homer Chen. Programmable aperture photography: Multiplexed light field acquisition. *ACM Transactions on Graphics*, 27(3):55:1–55:10, 2008. 32
- [LM71] EDWIN H. LAND and JOHN J. McCANN. Lightness and retinex theory. *J. Opt. Soc. Am.*, 61(1):1–11, 1971. 93, 169, 178
- [LPD07] Bruce Lamond, Pieter Peers, and Paul Debevec. Fast image-based separation of diffuse and specular reflections. In *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. 110, 111
- [LSC⁺06] Douglas Lanman, Peter G. Sibley, Daniel Crispell, Yong Zhao, and Gabriel Taubin. Multi-flash 3d photography: capturing shape and appearance. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Research posters*, page 99, New York, NY, USA, 2006. ACM. 202
- [MD98] Stephen Robert Marschner and Ph. D. *Inverse Rendering for Computer Graphics*. PhD thesis, 1998. 145
- [MG97] Stephen R. Marschner and Donald P. Greenberg. Inverse lighting for photography. In *IN FIFTH COLOR IMAGING CONFERENCE*, pages 262–265, 1997. 93
- [MH84] Gene S. Miller and C. Robert Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *SIGGRAPH 84 Course Notes for Advanced Computer Graphics Animation*, July 1984. 88
- [MHP⁺07] Wan-Chun Ma, Tim Hawkins, Pieter Peers, Charles-Felix Chabert, Malte Weiss, and Paul Debevec. Rapid acquisition of specular and diffuse normal maps from polarized spherical gradient illumination. In *Rendering Techniques 2007: 18th Eurographics Workshop on Rendering*, pages 183–194, June 2007. 110, 111, 203
- [MN99] T. Mitsunaga and S.K. Nayar. Radiometric Self Calibration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 374–380, Jun 1999. 176
- [MNBN07] F. Moreno-Noguer, P.N. Belhumeur, and S.K. Nayar. Active Refocusing of Images and Videos. *ACM Trans. on Graphics (also Proc. of ACM SIGGRAPH)*, Aug 2007. 32

- [MNNB05] F. Moreno-Noguer, S.K. Nayar, and P.N. Belhumeur. Optimal Illumination for Image and Video Relighting. In *IEEE European Conference on Visual Media Production (CVMP)*, pages 199–208, Dec 2005. 150
- [Mot09] Motorola. Droid smartphone. <http://en.wikipedia.org/wiki/Motorola>, October 2009. 62
- [MWL⁺99] Stephen R. Marschner, Stephen H. Westin, Eric P. F. Lafortune, Kenneth E. Torrance, and Donald P. Greenberg. Image-based brdf measurement including human skin. In *Eurographics Workshop on Rendering*, 1999. 87, 93
- [Nay97] S.K. Nayar. Catadioptric Omnidirectional Camera. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 482–488, Jun 1997. 204
- [Nay06] Shree Nayar. Computational cameras: Redefining the image. *IEEE Computer Magazine*, Aug.:30–38, 2006. 32, 33, 37
- [NB03] S.K. Nayar and V. Branzoi. Adaptive Dynamic Range Imaging: Optical Control of Pixel Exposures over Space and Time. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1168–1175, Oct 2003. 33
- [NBB04] S.K. Nayar, V. Branzoi, and T. Boulton. Programmable Imaging using a Digital Micromirror Array. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume I, pages 436–443, Jun 2004. 33
- [NFB97] Shree K. Nayar, Xi-Sheng Fang, and Terrance Boulton. Separation of reflection components using color and polarization. *Int. J. Comput. Vision*, 21(3):163–186, 1997. 111
- [Nie94] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann Series in Interactive Technologies. AP Professional, 1994. 187, 195
- [NIK11] Lesley Northam, Joe Istead, and Craig Kaplan. Rtx: On-set previs with unrealengine3. In Junia Anacleto, Sidney Fels, Nicholas Graham, Bill Kapralos, Magy Saif El-Nasr, and Kevin Stanley, editors, *Entertainment Computing ICEC 2011*, volume 6972 of *Lecture Notes in Computer Science*, pages 432–435. Springer Berlin / Heidelberg, 2011. 4
- [NKGR06] Shree K. Nayar, Gurunandan Krishnan, Michael D. Grossberg, and Ramesh Raskar. Fast separation of direct and global components of a scene using high frequency illumination. In *SIGGRAPH '06: ACM*

- SIGGRAPH 2006 Papers*, pages 935–944, New York, NY, USA, 2006. ACM. 112
- [NM90] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, CHI '90, pages 249–256, New York, NY, USA, 1990. ACM. 186
- [Nok09] Nokia. Qt a cross-platform application and ui framework. <http://qt.nokia.com/>, 2009. 53, 55
- [NRH03] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph.*, 22:376–381, July 2003. 148
- [NRH04] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graph.*, 23:477–487, August 2004. 148
- [OKP⁺08] Juraj Obert, Jaroslav Krivánek, Fabio Pellacini, Daniel Šýkora, and Sumanta N. Pattanaik. iCheat: A representation for artistic control of indirect cinematic lighting. *Computer Graphics Forum*, 27(4):1217–1223, 2008. 22, 147
- [Owe02] John D. Owens. *Computer Graphics on a Stream Architecture*. PhD thesis, Stanford University, November 2002. 45
- [OZ10] J.A. Okun and S. Zwerman. *The VES Handbook of Visual Effects: Industry Standard VFX Practices and Procedures*. Focal Press. Elsevier Science, 2010. 2, 3, 5, 6, 10, 194
- [PAH⁺04] Georg Petschnigg, Maneesh Agrawala, Hugues Hoppe, Richard Szeliski, Michael Cohen, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)*, 2004. 32
- [Pel10] Fabio Pellacini. envylight: an interface for editing natural illumination. *ACM Trans. Graph.*, 29:34:1–34:8, July 2010. 146
- [PHB08] Fabio Pellacini, Milo Haan, and Kavita Bala. Interactive cinematic relighting with global illumination. In Hubert Nguyen, editor, *GPU Gems 3*, pages 183–202. Addison-Wesley, 2008. 135, 155
- [Poh10] Brian Pohl. Previs on low to micro budgets. <http://www.previsociety.com/society-qa/post/1056286>, August 2010. 8

- [Poo10] D. Poole. *Linear Algebra: A Modern Introduction*. Brooks/Cole, 2010. 105
- [Pre92] W.H. Press. *Numerical Recipes in C: The Art of Scientific Computing*. Number bk. 4. Cambridge University Press, 1992. 106
- [Pro09] OpenHand Project. Clutter. <http://clutter-project.org/>, 2009. 55
- [PTMD07] Pieter Peers, Naoki Tamura, Wojciech Matusik, and Paul Debevec. Post-production facial performance relighting using reflectance transfer. *ACM Trans. Graph.*, 26, July 2007. 22, 150, 155
- [RAGS01] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Comput. Graph. Appl.*, 21(5):34–41, September 2001. 136
- [RAT06] Ramesh Raskar, Amit Agrawal, and Jack Tumblin. Coded exposure photography: motion deblurring using fluttered shutter. *ACM Trans. Graph.*, 25(3):795–804, 2006. 33
- [RBH08] Alexa I. Ruppertsberg, Marina Bloj, and Anya Hurlbert. Sensitivity to luminance and chromaticity gradients in a complex scene. *Journal of Vision*, 8(9), 2008. 28, 124
- [Rix01] Scott Rixner. *Stream Processor Architecture*. Number 0792375459 in Academic. Springer, Kluwer Academic Publishers, Boston, MA, 1st edition, October 2001. 41
- [RKAJ08] Erik Reinhard, Erum Arif Khan, Ahmet Oguz Akyz, and Garrett M. Johnson. *Color Imaging: Fundamentals and Applications*. A. K. Peters, Ltd., Natick, MA, USA, 2008. 94, 174, 190
- [RKKS⁺07] Jonathan Ragan-Kelley, Charlie Kilpatrick, Brian W. Smith, Doug Epps, Paul Green, Christophe Hery, and Frédo Durand. The light-speed automatic interactive lighting preview system. *ACM Trans. Graph.*, 26, July 2007. 22, 148
- [RSYD05] R. Ramanath, W.E. Snyder, Y. Yoo, and M.S. Drew. Color image processing pipeline in digital still cameras. *IEEE Signal Processing Magazine Special Issue on Color Image Processing*, 22:34–43, 2005. 68, 74
- [RTF⁺05] Ramesh Raskar, Kar-Han Tan, Rogerio Feris, Jingyi Yu, and Matthew Turk. Non-photorealistic camera: depth edge detection and stylized rendering using multi-flash imaging. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 2, New York, NY, USA, 2005. Merl, ACM. 32

- [RTM⁺06] Ramesh Raskar, Jack Tumblin, Ankit Mohan, Amit Agrawal, and Yuanzen Li. Computational photography. *ACM / EG Computer Graphics Forum*, Vol 25(3):1–20, 2006. 24, 37, 92
- [Rus11] J.C. Russ. *The Image Processing Handbook*. CRC Press, 2011. 160
- [RWPD05] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting*. Morgan Kaufmann Publishers, December 2005. 174, 176
- [SA07] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 73–80, New York, NY, USA, 2007. ACM. 207
- [SAM05] Jacob Ström and Tomas Akenine-Möller. ipackman: high-quality, low-complexity texture compression for mobile phones. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 63–70, New York, NY, USA, 2005. ACM. 47
- [SCD⁺06] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 519–528, Washington, DC, USA, 2006. IEEE Computer Society. 21, 87
- [SCG⁺05] Pradeep Sen, Billy Chen, Gaurav Garg, Stephen R. Marschner, Mark Horowitz, Marc Levoy, and Hendrik P. A. Lensch. Dual photography. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 745–755, New York, NY, USA, 2005. ACM. 150
- [SCMS01] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafe. A survey of methods for volumetric scene reconstruction from photographs. In *International Workshop on Volume Graphics*, 2001. 87
- [SGI09] SGI. Open graphic language. <http://www.opengl.org/>, 2009. 55
- [SH98] P. H. Suen and G. Healey. Analyzing the bidirectional texture function. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '98, pages 753–, Washington, DC, USA, 1998. IEEE Computer Society. 87
- [SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph.*, 21:527–536, July 2002. 148

- [SNB07] Y. Y. Schechner, S. K. Nayar, and P. N. Belhumeur. Multiplexing for Optimal Lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1339–1354, Aug 2007. 22, 146
- [Soc] Previsualization Soceity. Previsualization soceity website. <http://www.previsociety.com/>. 192
- [STST00] BUNTAROU SHIZUKI, MASASHI TOYODA, ETSUYA SHIBAYAMA, and SHIN TAKAHASHI. Smart browsing among multiple aspects of data-flow visual program execution, using visual patterns and multi-focus fisheye views. *Journal of Visual Languages & Computing*, 11(5):529 – 548, 2000. 72, 74
- [Sza95] Joseph Szadkowski. Desktops set lucas plan. *Washington times*, evening:2, May 1995. 8
- [TB97] L.N. Trefethen and D. Bau. *Numerical linear algebra*. Miscellaneous Bks. Society for Industrial and Applied Mathematics, 1997. 105, 106
- [TEW99] Shoji Tominaga, Satoru Ebisui, and Brian A. Wandell. Color temperature estimation of scene illumination. In *Color Imaging Conference'99*, pages 42–47, 1999. 172
- [UGY06] Jonas Unger, Stefan Gustavson, and Anders Ynnerman. Densely sampled light probe sequences for spatially variant image based lighting. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 341–347, New York, NY, USA, 2006. ACM. 88
- [vdWGG07] J. van de Weijer, T. Gevers, and A. Gijsenij. Edge-based color constancy. *Image Processing, IEEE Transactions on*, 16(9):2207 –2214, sept. 2007. 93, 178, 185
- [Vig04] J. A. Stephen Viggiano. Comparison of the accuracy of different white-balancing options as quantified by their color constancy. volume 5301, pages 323–333. SPIE, 2004. 89
- [VRA⁺07] Ashok Veeraraghavan, Ramesh Raskar, Amit Agrawal, Ankit Mohan, and Jack Tumblin. Dappled photography: mask enhanced cameras for heterodyned light fields and coded aperture refocusing. *ACM Trans. Graph.*, 26(3):69, 2007. 32
- [WDC⁺08] Oliver Wang, James Davis, Erika Chuang, Ian Rickard, Krystle de Mesa, and Chirag Dave. Video relighting using infrared illumination. *Comput. Graph. Forum*, 27(2):271–279, 2008. 149

- [Wei97] Joachim Weickert. A review of nonlinear diffusion filtering. In *Proceedings of the First International Conference on Scale-Space Theory in Computer Vision*, SCALE-SPACE '97, pages 3–28, London, UK, 1997. Springer-Verlag. 128
- [WGT⁺05] Andreas Wenger, Andrew Gardner, Chris Tchou, Jonas Unger, Tim Hawkins, and Paul Debevec. Performance relighting and reflectance transformation with time-multiplexed illumination. *ACM Trans. Graph.*, 24:756–764, July 2005. 22, 145, 150
- [wik12] Digital light meter image. Wikipedia Creative Commons, 2 2012. 86
- [WLL⁺09] Tim Weyrich, Jason Lawrence, Hendrik P. A. Lensch, Szymon Rusinkiewicz, and Todd Zickler. Principles of appearance acquisition and representation. *Foundation and Trends Computer Graphics and Vision*, 4, 2009. 87, 110
- [WOG06] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 1221–1226, New York, NY, USA, 2006. ACM. 142
- [WRWHL07] R. Peter Weistroffer, Kristen R. Walcott, Greg Humphreys, and Jason Lawrence. Efficient basis decomposition for scattered reflectance data. In *EGSR07: Proceedings of the Eurographics Symposium on Rendering*, Grenoble, France, June 2007. 87, 93
- [WS67] Gunter. Wyszecki and W. S. Stiles. *Color science : concepts and methods, quantitative data and formulas / Gunter Wyszecki & W. S. Stiles*. Wiley, New York :, 1967. 169
- [YDMH99] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse global illumination: recovering reflectance models of real scenes from photographs. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 215–224, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 145
- [ZN06a] L. Zhang and S. K. Nayar. Projection Defocus Analysis for Scene Capture and Image Display. *ACM Trans. on Graphics (also Proc. of ACM SIGGRAPH)*, Jul 2006. 32
- [ZN06b] A. Zomet and S.K. Nayar. Lensless Imaging with a Controllable Aperture. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2006. 32

- [ZREB06] T. Zickler, R. Ramamoorthi, S. Enrique, and P.N. Belhumeur. Reflectance sharing: predicting appearance from a sparse set of images of a known shape. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(8):1287–1302, Aug. 2006. 87, 93

A - Glossary of Terms and Definitions

- *Point & Shoot* : Refers to the type of commodity digital camera that has been designed for ease of use. Most of the camera settings are automatic, allowing novice users to focus on shooting the scene. These cameras tend to be less expensive than higher-end cameras with lenses and sensors that are less flexible and precise.
- *Light-field* : This concept refers to the notion that light can be considered an 8-dimensional ray space (4D incoming & 4D outgoing, neglecting time & color).
- *Traditional camera processing* : Is the process by which camera manufacturers adjust the raw pixel values in accordance with human visual perception. The adjustments applied to the raw pixels compensate for various environmental factors in order to improve the visual look of the rendered image to that of the human observer (see Figure 3.1).
- *Reconfigurable Camera* : Is the idea that a camera that possesses a programmable back-end can be programmed to perform the functionality of a completely different camera.
- *Natural Photography* : Natural photography is the taking of photographs with image adjustments only. In other words, there is no additional post-processing done to the images. The purpose of natural photographs is to capture the scene as it is seen by the viewer.
- *Active Illumination* : Are a set of computer vision techniques that control light in a specified way in order to infer or measure properties of a scene.
- *White Point* : Is a set of tristimulus values or chromaticity coordinates that serve to define the color "white" in image capture, encoding, or reproduction.
- *White Balance* : Adjusting the color or lighting in an image for preserving the neutral colors of the image as if the scene were illuminated by a white light.
- *Regular Simplex* : Is a generalization of a triangle.
- *Regular Polytope* : A polytope whose symmetry is transitive.
- *Convex Hull* : A set of points whose surface is minimal and completely envelopes the surface of another set of points convexly.
- *Convex Set* : In Euclidean space, is the set of all points on a straight line.

- *Correlated Color Temperature* : The color associated with the temperature of a black body radiator.
- *CIE D65 Illuminant* : A standard illuminant color proposed by the CIE which is commonly noted as noon day sun.
- *Shader (program)* : A small program used to modify part of a rendering pipeline.

B - Programming Code

We present alternate exposition of generating a β map using the pseudo-code.

Listing 1: Psuedo-code for calculating the β map for a camera image and scene with two lights.

```
C1 % image 1
C2 % image 2
l1 % color of light 1
l2 % color of light 2

for i=1 to I do      % I dimension
    for j=1 to J do  % J dimension
         $\beta_{map}(i,j) = \text{find\_min}(C_1(i,j) * (l_2 * \beta + l_1 * (1 - \beta)) - C_2(i,j) * (l_1 * \beta + l_2 * (1 - \beta)), \beta)$  %  $\leftrightarrow$ 
            find min  $\beta$  value
    end for
end for
```

Listing 2: Matlab file written to perform shadow detection and segmentation in Section 5.3.1.

```
clear all;
alpha = imread('alpha.png');
rgb = imread('rgb.png');

```

Listing 3: Matlab written to calculate the absolute difference between two images from Section 6.2.7.

```
clc;
clear;
clear all;
```

```

cd './data/error_balls/ward_aniso/'

gt      = './ward_diff/beta.png'; % ground truth, rendered
compare = 'beta.png'; % generated from data
out      = 'diff.png'; % generated from relighting application
error    = 'error.txt'; % generated from relighting application

gt = im2double(imread(gt));
dt = im2double(imread(compare));

diff = imabsdiff(gt,dt);

%boxplot(reshape(diff(diff(:,:,1)<.1), [], 1))
save 'diff_file' diff;
imwrite(diff, out);
diff_out = reshape(diff(diff(:,:,1)<.1), [], 1);

fid = fopen(error, 'w');
fprintf(fid, 'Mean=%f\n', mean(diff_out));
fprintf(fid, 'Std=%f\n', std(diff_out));
fprintf(fid, 'Var=%f\n', var(diff_out));
fprintf(fid, 'Med=%f\n', median(diff_out));
fclose(fid);

cd '../..\..\..'

```

Listing 4: Matlab written to calculate the perceptual using δ_E difference between two images from Section 6.2.7.

```

% Based on the article:
% "The CIEDE2000 Color-Difference Formula: Implementation Notes,
% Supplementary Test Data, and Mathematical Observations," G. Sharma,
% W. Wu, E. N. Dalal, submitted to Color Research and Application,
% January 2004.
% available at http://www.ece.rochester.edu/~gsharma/ciede2000/

%% Written By: Clifford Lindsay, www.wpi.edu, 2011
clc;
clear;
clear all;

gt      = 'gt_diff_rgb.png'; % ground truth, rendered
compare = 'cereal.png'; % generated from relighting application
out      = 'diff.png'; % generated from relighting application
error    = 'error.txt'; % generated from relighting application
%% CIE Delta E Tolerances L, C, H
KLCH = [5 1 1]; % used in calculations; tolerate more lightness than color/hue ←
diff

%% read images, 1st image is ground truth, 2nd is sample
rgb1 = im2double(imread(gt));
rgb2 = im2double(imread(compare));

%% convert to lab (deltaE only works in lab)
C = makecform('srgb2lab');
lab_rgb1 = applycform(rgb1,C);
lab_rgb2 = applycform(rgb2,C);

%% size info and preallocate arrays
[xDim, yDim, zDim] = size(lab_rgb1);

```

```

l1 = zeros(xDim*yDim,zDim);
l2 = zeros(xDim*yDim,zDim);

%% convert to dE200 format kx3 matrices
l1(:,1) = reshape(lab_rgb1(:,:,1), [], 1);
l1(:,2) = reshape(lab_rgb1(:,:,2), [], 1);
l1(:,3) = reshape(lab_rgb1(:,:,3), [], 1);
l2(:,1) = reshape(lab_rgb2(:,:,1), [], 1);
l2(:,2) = reshape(lab_rgb2(:,:,2), [], 1);
l2(:,3) = reshape(lab_rgb2(:,:,3), [], 1);

%% perform CIE Delta E 2000 difference operation
dE00 = deltaE2000(l1, l2, KLCH);

%% Convert back to viewable format and display
final = reshape(dE00,[xDim yDim]);

figure;
imagesc(final);
colormap('hot');
caxis([0.0 6.0]);
colorbar;

figure;
boxplot(dE00);
imwrite(final, out);

%% print some stats
fid = fopen(error, 'w');
fprintf(fid, 'quantile=%f %f %f %f %f\n', quantile(dE00,[.025 .25 .50 .75 .975]));
fprintf(fid, 'irq=%f\n', iqr(dE00));
fprintf(fid, 'Mean=%f\n', mean(dE00));
fprintf(fid, 'Std=%f\n', std(dE00));
fprintf(fid, 'Var=%f\n', var(dE00));
fprintf(fid, 'Med=%f\n', median(dE00));
fclose(fid);

```

Listing 5: Camera Shader code in Section 5.3.1 converts camera image to gray scale.

```

/*grayscale effect
* Description: Converts all color values to a grayscale tonal representation.
* Output: A camera image converted to grayscale.
* Written by: Clifford Lindsay, February 26, 2011.
* Department of Computer Science, WPI */
precision mediump float;
varying vec2 vTextureCoord;
uniform sampler2D texture;
void main(void)
{
    vec4 col = texture2D(texture, vTextureCoord);
    float gray = dot(vec3(col[0], col[1], col[2]),
    vec3(0.3, 0.59, 0.11));
    gl_FragColor = vec4(gray,gray, gray, 1.0);
}

```

Listing 6: Camera Shader code in Section 5.3.1 that converts camera image to a posterize style.

```

/*Posterize Effect
* Description: Posterize of an image entails conversion of a continuous gradation
* of tone to several regions of fewer tones, with abrupt changes from one tone to
* another.
* Ouput: Camera image with posterize effect applied
* Written by: Clifford Lindsay, Feburay 26, 2011.
* Department of Computer Science, WPI */
uniform sampler2D texture;
uniform float threshold = 0.5;
void main(void)
{
    vec4 pel = texture2D(texture,gl_TexCoord[0].xy);
    vec4 res = vec4(0.0,0.0,0.0,0.0);

    // Now, for every value above threshold, put in the maximum:
    if (pel.r > threshold) { res.r = 1.0; };
    if (pel.g > threshold) { res.g = 1.0; };
    if (pel.b > threshold) { res.b = 1.0; };

    gl_FragColor = res;
}

```

Listing 7: Camera Shader code in Section 5.3.1 that converts a camera image to halftone style.

```

/*halftone effect
* Description: Halftone is the reprographic technique that simulates continuous ↔
tone imagery
* through the use of dots, varying either in size, in shape or in spacing.
* Ouput: A camera image converted to a halftone style.
* Written by: Clifford Lindsay, Feburay 26, 2011.
* Department of Computer Science, WPI */
uniform sampler2D texture;
uniform float steps = 32.0;

void main(void)
{
    float dotsize = 1.0 / steps ;
    float half_step = dotsize / 2.0;

    vec2 center = gl_TexCoord[0].xy - vec2(mod( gl_TexCoord[0].x, dotsize),mod( ↔
gl_TexCoord[0].y, dotsize)) + half_step;
    vec4 pel = texture2D( texture , center );
    float size = length(pel);

    if (dotsize*size/4.0 >= distance(gl_TexCoord[0].xy,center)) {
        gl_FragColor = pel;
    } else {
        gl_FragColor = vec4(0.0,0.0,0.0,0.0);
    }
}

```

Listing 8: Camera Shader code in Section 5.3.1 that converts a camera image to a negative.

```

/*negative effect
* Description: Convert color values to their negative.
* Ouput: A camera image converted to a negative style.

```

```
* Written by: Clifford Lindsay, Feburay 26, 2011.  
* Department of Computer Science, WPI */  
precision mediump float;  
varying vec2 vTextureCoord;  
uniform sampler2D texture;  
void main()  
{  
    vec3 col = texture2D(texture, vTextureCoord).rgb;  
    gl_FragColor.rgb = vec3(1.0, 1.0, 1.0) - col;  
}
```

C - Reference Material

C.1 Description of Plenoptic Function :

In order to measure the Plenoptic function, imagine placing an idealized camera at every possible location in space, say (X,Y,Z) . Then record the intensity of the light rays passing through the center of the camera at every possible angle (θ, ϕ) , for every wavelength, λ , at every time point t . It is simplest to have the camera always look in the same direction, so that the angles (θ, ϕ) are always computed with respect to an optic axis that is parallel to the Z axis. The resulting function then has the following form listed in Equation 1 :

$$P = P(\theta, \phi, \lambda, t, X, Y, Z) \quad (1)$$

This seven variable formulation is sometimes referred to a light field, with minor differences from various authors providing their own take on the Light Field definition. Furthermore, if we have a corresponding functions for all reflected light with the same formulation as Equation 1, then we can fully describe all light paths in a scene using these fourteen variables. The reflected paths are sometimes referred to as a Reflectance field.

C.2 User Study Documents

C.2.1 PCam Expert User Study Script

User Study Script:

This study looks at the usability of an advanced camera interface. This study will walk you through several steps with the goal of applying image processing effects to images on a new type of programmable camera. The steps below should be followed in order.

Step 1 – Generate a pipeline that does simple edge detection. You will use three shaders organized in a pipeline: Grayscale, Sobel Filter, and Negative.

Open the Pcam Workbench application. To generate a pipeline, you need to **drag and drop** shaders from the *Shader Toolbox* area to the *Rendering Pipeline* area (see figure 1). A preview of the how the pipeline will affect the rendering is visible in the Preview Window, which appears below the Rendering Pipeline area.

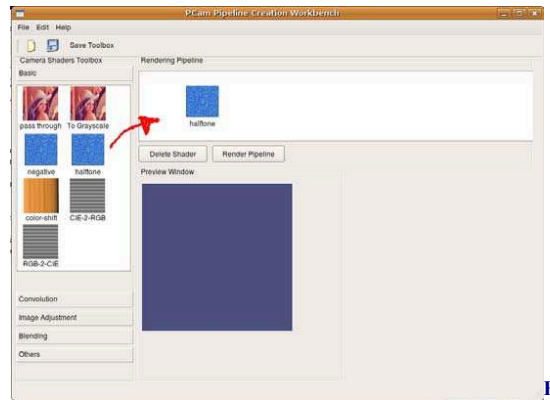


Figure 1: Workbench Application. To apply image filter to the preview window, drag and drop a specific filter from the Camera Shaders Toolbox to the Rendering Pipeline area.

Step 2 – Export the new pipeline to the Camera Interface.

Once you are satisfied with the look of the rendering for your new pipeline, you need to export the pipeline to the camera so the camera can use it to affect images captured by the camera. To do this, click on the **Save** icon in the application. This will bring up a file dialog for which you will have to name your pipeline. Type in a name for your new pipeline and click “**Save**”.

Step 3 – Apply the new pipeline on the camera.

Open the camera interface application and select the *Shaders Tab*. This will open the part of the camera interface where you can load and configure the pipeline you saved in the Pcam Workbench application. Click on your pipeline in the *Available Pipelines* area. This will select and load the pipeline, and the camera will begin rendering this pipeline's effects. Once you've selected your pipeline, you can then

User Study Script:

This study looks at the usability of an advanced camera interface. This study will walk you through several steps with the goal of applying image processing effects to images on a new type of programmable camera. The steps below should be followed in order.

Step 1 – Generate a pipeline that does simple edge detection. You will use three shaders organized in a pipeline: Grayscale, Sobel Filter, and Negative.

Open the Pcam Workbench application. To generate a pipeline, you need to **drag and drop** shaders from the *Shader Toolbox* area to the *Rendering Pipeline* area (see figure 1). A preview of the how the pipeline will affect the rendering is visible in the Preview Window, which appears below the Rendering Pipeline area.

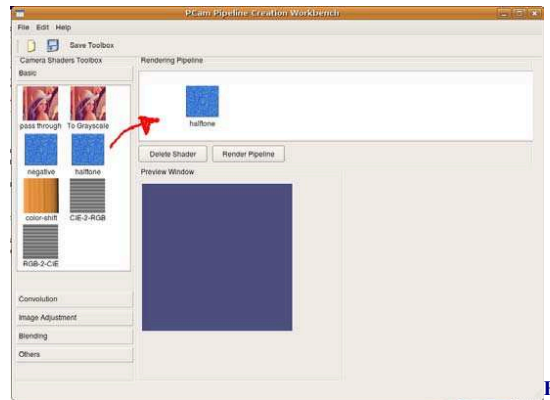


Figure 1: Workbench Application. To apply image filter to the preview window, drag and drop a specific filter from the Camera Shaders Toolbox to the Rendering Pipeline area.

Step 2 – Export the new pipeline to the Camera Interface.

Once you are satisfied with the look of the rendering for your new pipeline, you need to export the pipeline to the camera so the camera can use it to affect images captured by the camera. To do this, click on the **Save** icon in the application. This will bring up a file dialog for which you will have to name your pipeline. Type in a name for your new pipeline and click “**Save**”.

Step 3 – Apply the new pipeline on the camera.

Open the camera interface application and select the *Shaders Tab*. This will open the part of the camera interface where you can load and configure the pipeline you saved in the Pcam Workbench application. Click on your pipeline in the *Available Pipelines* area. This will select and load the pipeline, and the camera will begin rendering this pipeline's effects. Once you've selected your pipeline, you can then

preview of the picture you are about to take with the shaders applied is presented in the window. Here a negative shader is in effect.

Step 5 – Editing and saving photographs.

Photographs taken with the camera can be edited and saved using an interface similar to that of a traditional camera. To use this interface, click on the “**Picture Review**” tab at the top of the camera interface. Edit the previously taken photographs by modifying the brightness, saturation, and contrast using the provided Spinboxes.

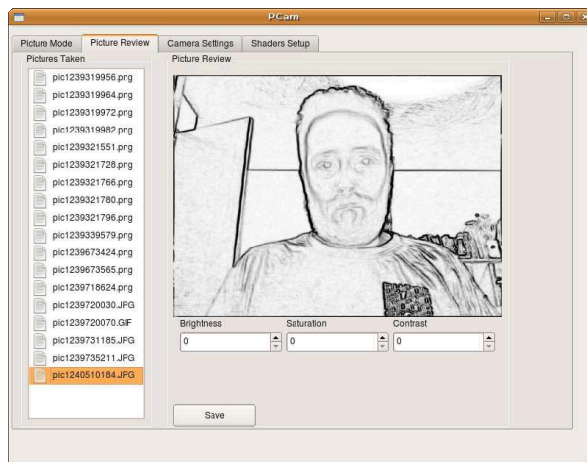


Figure 4: The camera interface's picture review tab. Once the picture has been taken, you can review the picture, adjust some of the basic properties, and save any changes.

Step 6 – Changing the camera settings.

To change the camera settings, click on the “**Camera Settings**” tab to select the camera setting window. To edit the base camera settings such as shutter speed, focus, and encoding, select the appropriate settings from the pull-down menus.

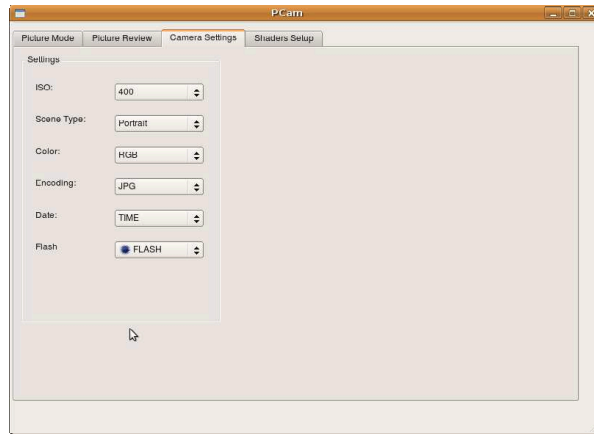


Figure 5: The Camera interface's Settings Tab. In this tab you can modify the camera settings.

C.2.2 Relighting User Study Questionnaire

Virtual Relighting Questionnaire

*Please mark your answer in the spaces indicated. If a question provides multiple choice answers, mark an **X** in the box indicating your answer.*

1. How old are you? _____ years
2. What is the highest education level you have achieved?
 - ☐ 8th grade
 - ☐ high school
 - ☐ GED
 - ☐ Some college
 - ☐ College degree
 - ☐ Some graduate school
 - ☐ Graduate degree
3. What is your gender?
 - ☐ Male
 - ☐ Female
4. How often do you use computers (work or leisure)?
 - ☐ 0-5 hours per week
 - ☐ 6-10 hours per week
 - ☐ 11-15 hours per week
 - ☐ 16-20 hours per week
 - ☐ More than 20 hours per week
5. If you use a computer regularly, how much time do you spend on a computer in a week? _____ hours/week
6. Have you taken a Computer Graphics course?
 - ☐ Yes
 - ☐ No

C.2.3 Previsualization Society Questionnaire

Previsualization Survey Questionnaire

- How old are you? __ __
- What is your gender? M__ F__
- What is your occupation? _____
- What is the highest level of education you have? H.S. __ 2-year College __ 4-year College __
Graduate School (Masters) __ Graduate School (Ph.D., MD. Etc.) __
- How many years have you been doing previsualization professionally? __ __
- In general, how is set lighting changed to accommodate a director's vision? (Physically on set, lighting isn't a big deal, in post, other (please explain) (_____)).
- What percentage of on-set time is spent changing the lighting? (< 25%, 25-50%, 50-75%, 75% >)
- Is previsualization generally concerned with on-set lighting? (yes, no, sometimes (how much, __% of the time)).
- Is previsualization generally concerned with lighting or relighting in post production (i.e. do directors want to see what the final lighting might look like during principle photography even though it may not be finalized until post production)? (yes, no, sometimes (how much, __% of the time)).
- If lighting is not generally part of previsualization or routinely ranked lower than other previsualization tasks, do you think that it should be more prevalent? (yes, no, adequately addressed now).
- If Lighting is previsualization on-set or in pre-production, please list the three top ways:
 1. _____
 2. _____
 3. _____
- If you could modify the set lighting virtually without physically altering the lights, how valuable would that be? (Not at all, somewhat, valuable, pretty valuable, extremely valuable).
- If virtual relighting of a real set was possible, what property of the lighting would be most valuable to modify interactively? (mark all that apply: location, color, intensity, shape of light distribution, all of the above)
- How much of an impedance is it to not be able to interactively see lighting changes on a real set? If interactive lighting and/or relighting is already being performed on set mark N/A instead of none (N/A, none, little, moderate, fair amount, extremely limiting).
- If you have any other comments about lighting, previsualization of lighting, virtual relighting, or this questionnaire please write them here:

D - Additional Renderings



Figure 1: Early preliminary renderings done with the first prototype hardware system.



Figure 2: Results using the laptop GPU with basic image processing. Each example is rendered on-camera in real-time instead of being offloaded to a computer for post processing. Left: Is a simulation heavy film grain. Left Center: Is a stylistic rendering. Right Center: Is a sepia tone filter applied to the image. Right: Edge detection used on an image of a house.

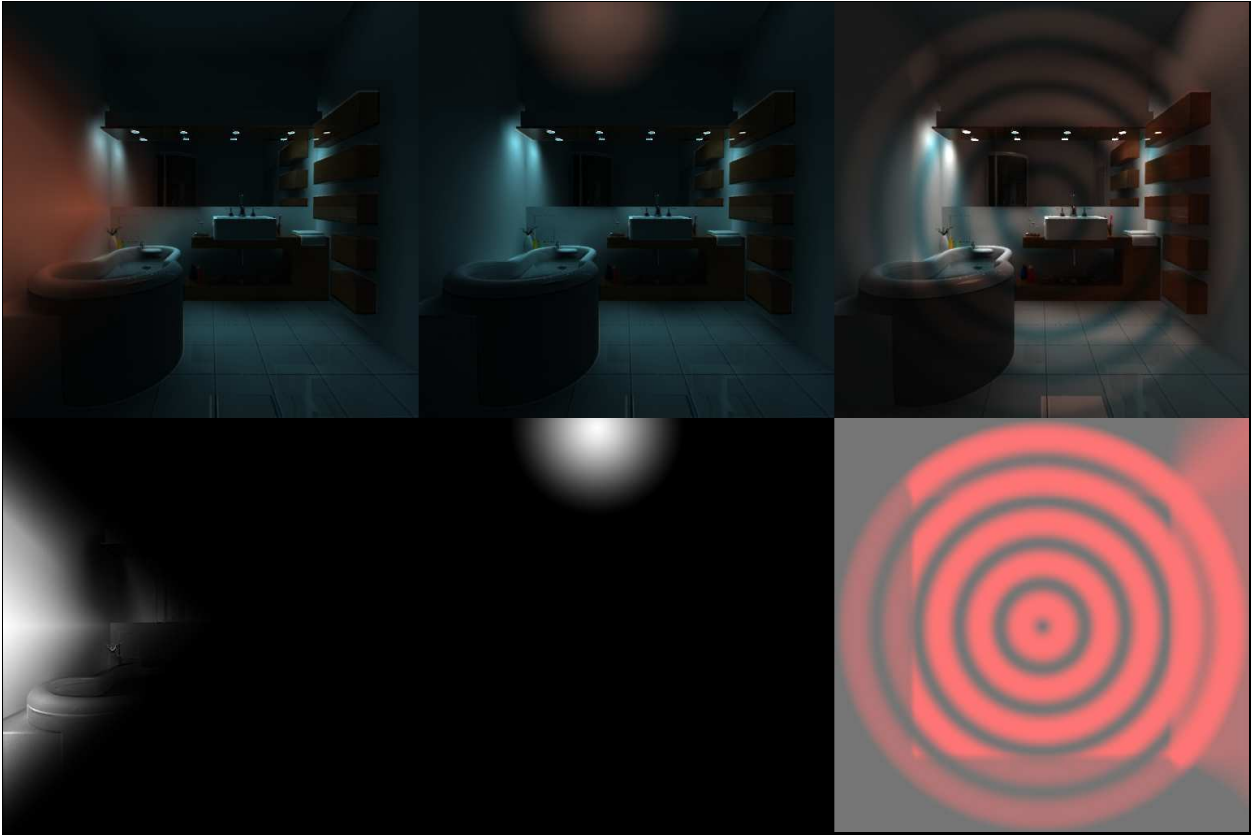


Figure 3: Additional β map manipulation images with their corresponding gradient masks. These were generated in Photoshop.



Figure 4: Pipeline images rendered using PCam; Left is the original image, left center has applied a color to gray scale conversion filter. Right center is a Gaussian blur kernel applied to the gray scale image. Left is an edge detection filter applied to the grey scale and blurred image.

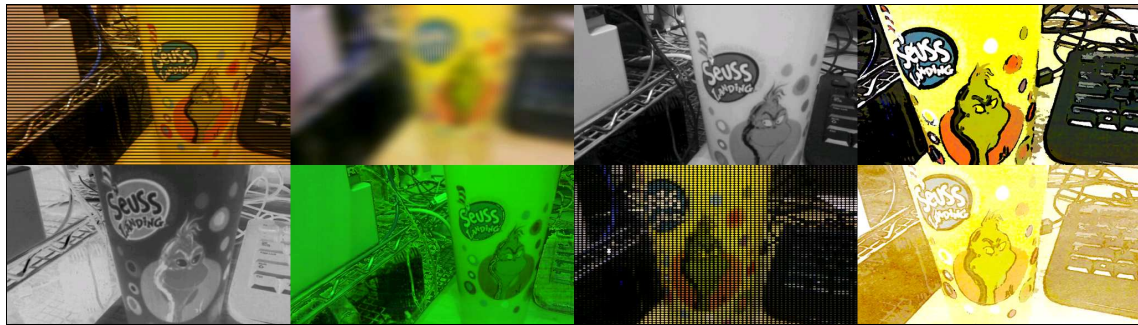


Figure 5: Pipeline images rendered using PCam; Top far left is a simulation of old scanlines, top center left is a Gaussian blur, top right center is gray scale conversion, and top far right is an effect called rotoscoping. Bottom far left is a negative effect, bottom center left is a night vision effect, bottom center right is a tile effect (groups of pixels converted to tiles), bottom far right is a watercolor painting simulation effect.