Image-Based Remodeling: A Framework for Creating, Visualizing, and Editing Image-Based Models

R. Alex Colburn

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

Doctor of Philosophy

University of Washington

2014

Reading Committee:

Brian Curless, Chair

Michael F. Cohen, Chair

Dieter Fox

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Image-Based Remodeling: A Framework for Creating, Visualizing, and Editing Image-Based Models

R. Alex Colburn

Co-Chairs of the Supervisory Committee:
Professor Brian Curless
University of Washington

Affiliate Professor Michael F. Cohen
Microsoft Research

Image-based models are geometric models created from photographs and textured with the photographs for realistic rendering. In recent years, it has become increasingly easy to capture many photographs of an object and use computer vision techniques to model the object with image-based geometry. However, it can be difficult to display and interact with these models in a manner that reproduces the visual fidelity of the original photographs. It is even more difficult to support the interactive manipulation and editing that one might expect from other types of geometric models.

The overarching goal of this thesis is to create a framework for building, visualizing, and editing image-based models. Toward this goal, I first describe the related literature and outline the associated research challenges related to geometric representation, navigation, creation, and editing of image-based geometry.

One of the challenges in visualizing image-based geometry is discovering camera viewpoints and navigation paths that are both visually pleasing and enable story telling. To address this challenge, I describe a GPU-accelerated, image-based rendering algorithm that enables the creation of such visualizations in the form of camera paths and cinematic effects commonly used by cinematographers. In order to avoid objectionable rendering artifacts such as occlusion holes, the fast rendering algorithm is used to quickly sample the parame-

ter space of camera viewpoints and define a viable region of camera parameters. This region is subsequently used to constrain an optimization for a camera path that maximizes parallax and conforms to cinematic conventions. This rendering algorithm also allows users to create more complex camera paths interactively, while experimenting with effects such as focal length, depth of field, and selective, depth-based desaturation or brightening.

A greater challenge is editing image-based geometry. To move image-based models away from the limitations of view-only geometry to the broader class of editable geometry, I describe a new approach for creating and visualizing editable image-based architectural models in a photorealistic manner. Using this approach, I present an interactive system that enables modeling, and remodeling of image-based geometry in the context of home interior architecture. This system supports creation of concise, parameterized, and constrained geometry, as well as remodeling directly from within the photographs. Real-time texturing of modified geometry is made possible by precomputing view-dependent textures for all of the faces that are potentially visible to each original camera viewpoint, blending multiple viewpoints and hole-filling where necessary. The resulting textures are stored and accessed efficiently, enabling intuitive, real-time, realistic visualization, modeling, and editing of the building interior.

Finally, I demonstrate how the image-based remodeling system enables lighting effects. Using a combination of the texture created in the image-based remodeling system and radiosity form-factor calculations, we estimate the irradiance at any location in the model. We can utilize this irradiance estimate to further refine light source estimates, and calculate surface reflectance properties. Given the additional estimates, edited models can be re-rendered to reflect lighting changes due to edited geometry, changing light properties, and the addition of synthetic objects.

TABLE OF CONTENTS

GLOSSARY

ALBEDO: the ratio of diffusely reflected energy (light, shortwave radiation) from the surface to incident energy upon it. See equation A.9 for a more formal definition.

HIGH DYNAMIC RANGE IMAGING (HDRI OR HDR): a set of methods used in imaging and photography used to produce images with greater luminance value dynamic range (and luminance resolution) than standard photographic methods. HDR images can accurately the range of luminance values found in real scenes.

IMAGE-BASED RENDERING (IBR): a collection of techniques and representations used to visualize 3D scenes and objects in a realistic way without full 3D model reconstruction. IBR uses images as the primary substrate.

LAMBERTIAN: A Lambertian surface that is assumed to be a diffuse reflector and uniformly reflects light with no directional dependence for the viewer, e.g., a matte, a very surface such as cardboard. This is in contrast to a specular surface that behaves more like a mirror and reflects light only along specific directions.

LIGHT FIELD OR LUMIGRAPH: a 4D subset of the plenoptic function. The light field is typically parameterized using the intersections of rays with two planes. A common setup consists of a parallel camera plane and focal plane.

LIGHT PROBE: an omnidirectional, high dynamic range image that records the incident illumination conditions at a particular point in space.

MULTI-VIEW STEREO (MVS): a method of recovering 3D positional information of matching features in image sequences sets for which camera parameters and poses are known

[51].

PHOTOGRAMMETRY: the practice of determining the geometric properties of objects from photographic images.

PLENOPTIC FUNCTION: Agarwala [2] describes the plenoptic function as follows:

> Light can be modeled as rays that travel through three-dimensional space without interfering with each other. Imagine a parameterized function of all the rays of light that exist throughout three-dimensional space and time, which describes the entire visual world and contains enough information to create any possible photograph or video ever taken. Adelson and Bergen [1] showed that this function, which they call the plenoptic function, can be parameterized by seven parameters: a three-dimensional location in space (Vx, Vy, Vz), time (t), the wavelength of the light (λ), and the direction of the light entering the location (parameterized by spherical coordinates θ, φ). Given these seven parameters, the plenoptic function returns the intensity of light (radiance). Thus, the plenoptic function provides a more precise notion of the visual world surrounding us that we wish to depict.

STRUCTURE FROM MOTION (SfM): a method of recovering camera parameters, pose estimates, and sparse 3D scene geometry from matching features in image sets [51].

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to express a very deep and sincere gratitude to my advisors, Michael Cohen and Brian Curless, for their guidance, encouragement, and support. Thanks to the other members of my reading committee, Dieter Fox and Mehlika Inanici, and to my numerous collaborators including David Salesin, Aseem Agarwala, Aaron Hertzman, Maneesh Agrawala, and Colin Zheng.

I would also like to thank the current and former UW students and postdoctoral fellows, faculty, and staff who have helped make my time at the UW such a pleasure. Very special thanks to Laura Effinger-Dean, Kristi Morton, and the rest of the UW CSE Band.

Finally, I would like to express my deepest gratitude to my wife Heather Colburn for her love and support.

DEDICATION

to my dear wife, Heather

Chapter 1

INTRODUCTION

Photographs and geometric models are powerful ways to visually communicate details of physical objects. Not only do photographs exhibit great detail, but they are also easy to capture, as well as easily shared, and edited. In contrast, geometric models are somewhat more difficult to create and display than photographs. However, some geometric models are configured to depict information about the object, by varying viewpoints and displaying intrinsic dimensions, that are not obvious in a photograph. Because geometric models are often manually created, they are typically designed to have the affordances for easy editing and manipulation.

Image-based models are geometric models that are created from photographs and use the texture from the photographs for realistic rendering. This synthesis of photography and geometry allows one to create photorealistic virtual environments from a few photographs.

Consider a home interior; a few well placed snapshots like those found on real estate websites (see Figure 1.1) help convey details about room usage and furnishings. However, it is difficult for a single photograph to depict complete rooms, floor plans, room adjacency information, or even scale. Further, photographs are not well suited for capturing complicated building interiors such as narrow hallways connecting rooms, small rooms like bathrooms, and stairways, for removing clutter such as furniture.

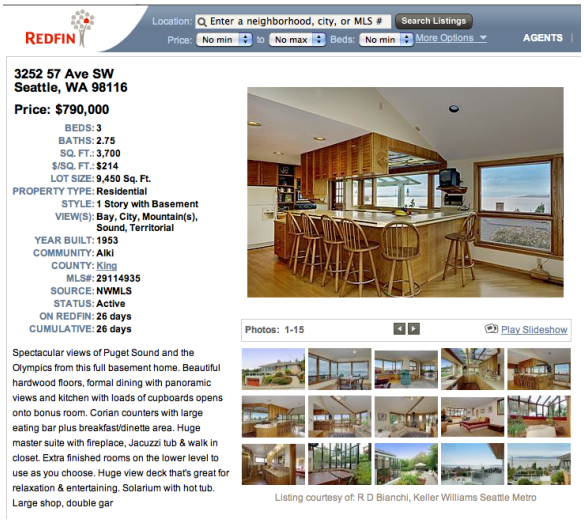


Figure 1.1: Photographs on a real estate website [102]. While the photographs convey information about a particular room and its furnishings, they fail to provide an overall sense of the home.

Purely geometric representations of an

object often provide a more informative view of the object than a photograph. In the case of a home interior, basic features, such as the floor plan, are explicitly defined. The model may be rendered from any viewpoint in a stylized manner. Optionally, geometric models may be rendered with occluders (such as ceilings or obstructing walls) selectively removed or made transparent to reveal important details behind the occluders (as in Figure 2.1). On the other hand, stylized rendering has its disadvantages. For example, stylized renderings usually lack sufficient geometric detail to produce realistic looking renderings. While it is certainly possible to create photorealistic renderings of an object or home interior, doing so typically requires a professional modeler with considerable skill.

Image-based models are geometric models created from photographs and textured with the photographs for realistic rendering. Image-based models have the potential to combine the advantages of both photographs and traditional geometry. Creating image-based models is fairly easy, and in some cases, can be fully automated using simple hardware such as a mobile phone [6]. When viewed, image-based models exhibit the detail of a photograph, even when the underlying geometry is coarse and lacks fine detail. Due to the depth imparted by the geometry, transitioning between viewpoints results in motion parallax between near and far parts of the scene. This motion parallax imparts added realism to the scene. Neither photographs nor geometry by themselves exhibit the level of detail and realism found in image-based geometry.

However, there are several problems with image-based models that make it difficult to use them to their fullest potential. In particular, and in contrast to well-designed traditional geometric models, image-based models are difficult, if not impossible, to edit. For example, it is very difficult to change lighting, change object colors, insert synthetic objects, and even perform basic editing operations, such as repositioning items within the model. Further, it is difficult to edit view-dependent texture. Unlike traditional geometry, there is no native mechanism to deal with occluded surfaces. If a surface is occluded from a certain viewpoint and a structure is removed, there is no mechanism to fill in texture for the newly revealed surface. Unlike a single photograph, simple edits have to be propagated to all viewpoints. Finally, as with photographs, there is no mechanism to reflect the shading changes or changes in lighting that might occur during an edit. While having a beautiful image-based virtual

environment for viewing a home interior or other object is a desirable goal, it is generally preferable to have the capability to edit the model.

In addition to the difficulty with editing, image-based models also present challenges to visualization. Sparse image data can make rendering and display problematic. Aside from the problem of discovering viewpoints that highlight interesting features or providing useful perspectives, it is often difficult to avoid transient viewpoints that exhibit objectionable artifacts produced during animation. Often the only visually pleasing viewpoints for an image-based model are near a viewpoint of an original photograph. This problem reduces functionality when navigating or finding a visual path within a model.

In their current state, image-based models are beautiful creations that clearly depict the objects they model; however, they are severely limited by the difficulties involved with viewing and even the most basic of editing operations.

The overarching goal of this thesis is to create a framework for building, visualizing, and editing image-based models. Toward this goal, in Chapter 2 I describe the related literature and outline the associated research challenges related to geometric representation, navigation, creation, and editing of image-based geometry.

One of the challenges in visualizing image-based geometry is discovering camera viewpoints and navigation paths that are both visually pleasing and enable story telling. To address this challenge, in Chapter 3 I describe a GPU-accelerated, image-based rendering algorithm that enables the creation of such visualizations in the form of camera paths and cinematic effects commonly used by cinematographers. In order to avoid objectionable rendering artifacts such as occlusion holes, the fast rendering algorithm is used to quickly sample the parameter space of camera viewpoints and define a viable region of camera parameters.

This region is subsequently used to constrain an optimization for a camera path that maximizes parallax and conforms to cinematic conventions. This rendering algorithm also allows users to create more complex camera paths interactively, while experimenting with effects such as focal length, depth of field, and selective, depth-based desaturation or brightening.

A greater challenge is editing image-based geometry. To move image-based models away

from the limitations of view-only geometry to the broader class of editable geometry, I describe a new approach for creating and visualizing editable image-based architectural models in a photorealistic manner. Using this approach, I present in Chapter 4 an interactive system that enables modeling, and remodeling of image-based geometry in the context of home interior architecture. This system supports creation of concise, parameterized, and constrained geometry, as well as remodeling directly from within the photographs. Real-time texturing of modified geometry is made possible by precomputing view-dependent textures for all of the faces that are potentially visible to each original camera viewpoint, blending multiple viewpoints and hole-filling where necessary. The resulting textures are stored and accessed efficiently, enabling intuitive, real-time, realistic visualization, modeling, and editing of the building interior.

Finally in Chapter 5, I demonstrate how the image-based remodeling system enables lighting effects. Using a combination of the texture created in the image-based remodeling system and radiosity form-factor calculations, we estimate the irradiance at any location in the model. We can utilize this irradiance estimate to further refine light source estimates, and calculate surface reflectance properties. Given the additional estimates, edited models can be re- rendered to reflect lighting changes due to edited geometry, changing light properties, and the addition of synthetic objects.

I conclude in Chapter 6 and briefly outline some interesting projects for future work.

Chapter 2
RELATED WORK

There is a large body of literature related to image-based modeling and rendering. In this chapter, I will highlight some of the related literature in this area, primarily focusing on the literature applicable to modeling architectural structures from photographs. The literature can be grouped into four broad categories: (1) creating image-based models; (2) rendering and navigating through image-based models; (3) relighting and inserting synthetic objects into image-based models, and (4) editing image-based models.

The above task-driven grouping correlates approximately to the information that must be gathered (through interaction, measurements, or other calculations) to complete each task. Roughly speaking, such information includes (1) a geometric representation of the geometry's 3D shape, (2) surface materials, (3) the illumination of the scene, and (4) images (often acquired as photographs or rendered as images) of the scene.

2.1 Model Creation

2.1.1 Computer Aided Design

Traditional computer aided design (CAD) systems have a long history starting with Ivan Sutherland's Sketchpad in the 1960s [116]. Since then, the tools for creating and editing geometry, particularly those for architectural and mechanical purposes, have become increasingly sophisticated and functional. A geometric model is often comprised of constrained primitives and functional entities (e.g., windows, walls, etc.) that can be easily modified. This approach works well for creating an editable model, but is time consuming to create. Further, geometric models created of existing buildings include only information added manually to the model. Therefore, such models lack important information (such as texture, lighting, or physical dimensions) that has not been added manually to the model.

There are many commercial and free CAD tools available to model existing architectural

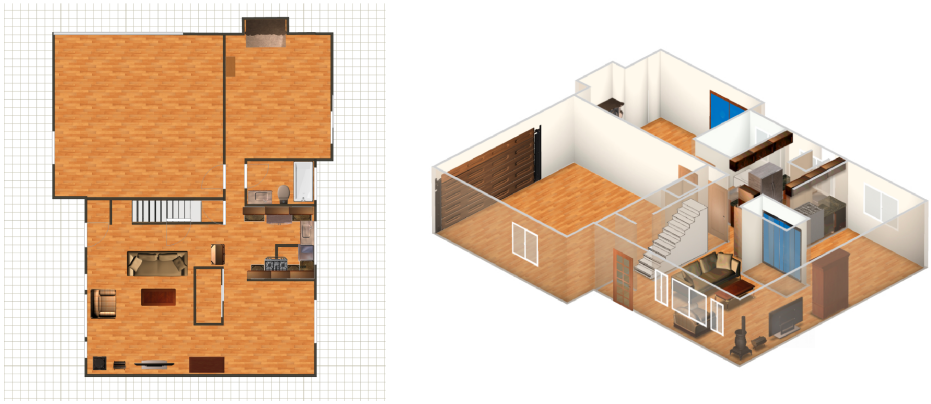


Figure 2.1: A simple layout modeled with Autodesk Homestyler. Even with the aid of a detailed architectural floor plan providing dimensions, this geometric model took several hours of user time to produce. Note how the stylized rendering makes it easy to grasp the layout of the home. Some Occluding walls are rendered transparently to reveal detail.

structures and visualize renovations. Often CAD software supports features that are beyond the scope of this thesis, but are required for manufacturing and the construction of real buildings. Although important, features such as support for compliance with building codes, structural analysis, etc. are not considered here. The work in this thesis focuses primarily on casual users, rapid prototyping, and easily constructing and editing models.

Even with advanced user interaction, CAD modeling requires significant expertise. Simpler tools such as Autodesk's Homestyler [5] are targeted at non-experts and can be used to author attractive, but highly-abstracted, floor plan renderings. These tools do not support the creation of photorealistic 3D models based on the original environment. These systems do not model the natural lighting or depict the clutter (e.g. furnishings) in existing structures. Thus the resulting visualizations are somewhat artificial. Another obstacle to model creation with CAD software is the lack of measurement data. It can be very tedious and time consuming to accurately measure all of the objects and relationships between objects one wishes to model.

In an effort to increase realism and provide basic photogrammetry, Trimble (previously Google) SketchUp [119] (Figure 2.2) allows interactive modeling using photographs with the "Photomatch" feature. This feature is designed to support modeling and visualization

by projecting textures onto an existing scene. However, Sketchup does not provide a good representation for remodeling once photographs are projected onto a model. For example, objects are split by occlusions during texture projection (e.g., when a column interrupts a view of a wall). Split polygons make subsequent editing extremely difficult. SketchUp assumes a single, globally defined texture mapping, which behaves poorly from novel viewpoints when using coarse proxy geometry (e.g., for plants, furniture, and other clutter), and when making edits to geometry. The photogrammetry tools consist only of tracing over an existing image, which is a not suitable replacement for actual measurements.

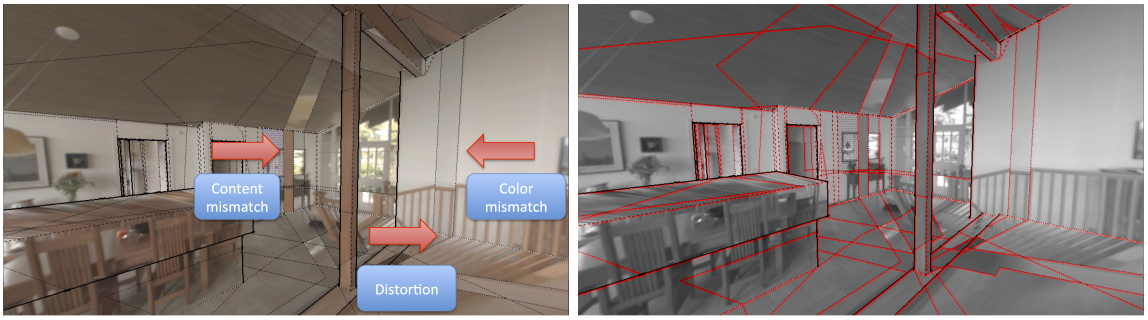


Figure 2.2: SketchUp. Left Image: a model with projected globally-defined texture-mapping, which behaves poorly from novel viewpoints when using coarse proxy geometry. The texture displays mismatched content, color mismatches and projective distortions. Right Image: Unconstrained subdivided polygons make subsequent editing extremely difficult.

Analysis: Current geometry-based methods of modeling existing architecture have several drawbacks. First, the process is extremely labor-intensive, typically involving surveying the site and digitizing architectural plans (if available). It is difficult to verify whether the resulting model is accurate. Second, personalizing the model takes even more effort. It is very difficult to create a detailed model that contains all of the features of your own home. Finally, the renderings of the resulting models are noticeably stylized; even those that employ texture mapping or advanced global illumination techniques can't provide the real look and feel of your house.

Summary: While there are deficiencies in displaying and creating CAD models, the parameterized, constrained editable models of Professional CAD software, such as Autodesk's

Revit [7] inspire my proposed work, as do the many affordances designed to make model creation easy and efficient.

2.1.2 Interactive image-based modeling

By starting with a collection of photographs, image-based modeling techniques can create photo-realistic virtual environments with much less effort than traditional CAD programs. This section describes the fundamental principles of creating geometry from photographs with user interaction.

Debevec et al. [31] introduced two concepts that are fundamental to much of the work in image-based modeling of existing architectural structures. With the Façade system, they combined both geometry-based and image-based techniques to create models of architectural structures from a few photographs. First, they used interactive photogrammetric modeling to help solve the stereo image correspondence problem. Due to the poor performance of automatic image feature matching and correspondence algorithms at the time, they employed user interaction to identify matching features between photographs. They overlaid coarse architectural models onto photographs and used features in the geometry (user-supplied correspondences) to calculate stereo correspondences between pairs of images. With this correspondence, they further solved the structure from motion problem to recover the 3D locations of the geometric points, as well as the positions of the original cameras, up to an unknown scale factor. Once they obtained stereo image pairs, they could use the coarse geometry to further refine features with model-based stereo. They showed that projecting pairs of images onto an initial approximate model allowed stereo techniques to robustly recover accurate depth measurements from images with widely varying viewpoints.

Secondly, they introduced view-dependent texture mapping, a method of blending multiple views of a scene in a way that better simulates geometric detail while still using coarse models. In essence, view-dependent texture mapping is a method of blending and interpolating multiple texture maps, in this case original photographs of the architecture, to create a better texture for a novel viewpoint.

Buehler et al.'s *Unstructured lumigraph rendering* [17] described a general approach to

image-based rendering that generalizes many image-based rendering algorithms, including light field rendering [77, 48] and view-dependent texture mapping. In particular, the algorithm allows for lumigraph-style rendering [77] from a set of input cameras in arbitrary configurations (i.e., not restricted to a plane or to any specific manifold). In the case of regular and planar input camera positions, the algorithm reduces to a typical lumigraph approach. When presented with fewer cameras and good approximate geometry, the algorithm behaves like view-dependent texture mapping. This paper is particularly useful for my work because it describes an objective function that determines the best pixels to sample in an assortment of photographs to reproduce a new texture from a novel viewpoint.

After the introduction of the Façade system, interactive approaches to 3D modeling from photographs have continued to improve by using cues such as vanishing points and to aid in both camera calibration and geometry placement within the scene.

One of the initial challenges in creating models from images is discovering the relationship between the pixels in a photograph and the real-world geometry. Once a camera's projection matrix, including internal calibration (e.g., focal length), and the explicit relationship between the camera and the world is known, modeling becomes a much simpler task. Cipolla et al. [22] exploited parallelism and orthogonality present in indoor and outdoor architectural scenes to calibrate cameras and recover the projection matrices. Criminisi et al. [26] further refined recovering camera parameters from a single image using vanishing points and a vanishing line of a reference plane. Criminisi's work enabled 3D modeling from single images.

Automatically recovering 2D feature correspondences between images, camera poses, and sparse 3D point clouds became practical with the advent of reliable image feature matching (such as SIFT [81]) and dependable SfM algorithms (such as Snavely et al.'s *Bundler* [114, 115, 112]). After this work, the primary challenge in constructing geometry became accurately placing geometry within the scene and creating image-based texture.

The *VideoTrace* system demonstrated by van den Hengel et al. [122] enables interactively generating 3D models of objects from video. The user traces polygon boundaries over video frames, and the traced boundaries (represented by 3D lines or curves) are automatically refined in 2D by fitting them to local, strong superpixel boundaries. The VideoTrace system

repeatedly re-estimates the 3D model using other video frames. The VideoTrace system has an advantage of extracting higher order primitives such as NURBS surfaces which are more easily editable than unstructured polygons.

Sinha et al. [111] utilize vanishing points found in a sparse set of photographs to guide geometry placement when modeling building exteriors. They extract lines and use them to automatically estimate vanishing points in the scene. In this system, users interactively sketch geometry over photographs, and vanishing points are used to constrain geometry placement. Additionally, edges may be snapped to vanishing lines (detected by proximity) to help provide more accuracy. In a departure from the approach used by the Façade system, this system creates a global 2D texture from the source photographs rather than using view-dependent textures.

Arikan et al., in their *O-Snap* system [3], take a different approach to modeling building exteriors by working directly on a point cloud created from SfM and MVS. They fit planar primitives to the point cloud. Then users interactively refine the primitives. The key insight of this system is that a polygon's placement is jointly optimized to "snap" to adjacent polygons and fit to the point cloud. This optimization-based snapping algorithm is interleaved with user interaction, allowing the user to sketch modifications with coarse and loose 2D strokes.

In addition to making geometry easier to create by exploiting data priors, many techniques increase automation by utilizing smarter geometric primitives to help automate repetitive tasks in the user interface. Of note are the following two systems.

El-Hakim et al. [37] exploit the repetitive nature of many architectural components by providing a component "copy and paste" mechanism. They build a project-specific collection of reusable highly detailed components such as windows, columns and archways. The goal is to create a component (such as a window) once, and reuse the component in other portions of the model. They create a detailed component directly from one example structure, and copy and paste it onto a similar but un-modeled destination location. During the paste operation, they automatically find the transformations, such as scaling, between the original component and the destination location to account for any size or shape disparity between the source and destination. In order to preserve visual fidelity, they use textures from the

destination location.

Similarly, Nan et al. [87] in their SmartBoxes system interactively fit solid architectural building blocks to 3D point cloud data from LiDAR scans for large-scale urban reconstruction. SmartBoxes assumes Manhattan world scenes [39], and reconstructs façades with repetitive axis-aligned structures.

Analysis: While all of the systems discussed above show how to interactively create models, none addresses the problem of authoring and rendering edits that depart from the original structures being modeled. With the exception of Arikan's O-Snap system, VideoTrace, and Debevic's Façade system, all of the systems focus on reconstructing building exteriors or objects as collections of disconnected slabs and polygon meshes, which we know are difficult to edit. In the case of Arikan, it is unclear if the point cloud clustering mechanism and adjacency detection mechanism scales to the complexity of cluttered home interiors. VideoTrace focuses on creating models of individual solid objects (such as a car exterior), rather than trying to reconstruct an entire structure with the complexity of a home interior, which includes gaps (or voids), and occlusions. Façade creates a hierarchical system of constrained primitives termed "blocks" that are useful for determining the stereo correspondences. They didn't discuss if the the system would be useful for subsequently editing geometry. The systems that exploit repetition in structure (such as SmartBoxes) focus on large features that don't necessarily exist in home interiors.

Summary: There are many interactive tools and techniques available to aid in building image-based models. Creating geometry for highly detailed image-based models using such tools certainly can be improved, but care must be taken so that the final output can be edited. Rather than trying to fix un-editable geometry, the obvious approach is to create well-formed editable geometry from the ground up.

2.1.3 Automatic image-based modeling

Automatically constructing image-based models is an active area of research. Systems for automatically creating models range from using simple planes as geometric proxies for rendering novel viewpoints (Photo Tourism [114]), to dense MVS reconstruction of point clouds

and meshes. This section will first summarize the work on creating geometry using SfM and MVS, then discuss alternatives that use higher level geometric representation and domain knowledge to better construct models.

Pollefeys et al. [98] describe a fundamental approach to 3D reconstruction from photographs. Starting with photographs taken with a hand held camera, the system uses SfM to automatically calibrate camera parameters and find camera poses. Then MVS is used to obtain a dense point cloud of the surface geometry. From this point cloud, they created both traditional and image-based geometry with view-dependent texture. More specifically, they created two different types of meshes: simple 2.5D meshes from single viewpoint depth maps, and full 3D meshes. They used a depth map created via MVS to construct a regular mesh by triangulating the depths, while avoiding regions with large disparities and unknown depths. This approach creates multiple surface patches. In order to create a single surface representation, they implemented the method of Curless and Levoy [29] to obtain an implicit representation of the surface, followed by the marching cubes algorithm to obtain an explicit mesh representation [80], and finally mesh simplification.

This basic approach has been augmented using additional hardware and clutter removal. Cornelis et al. [25] used a video camera stereo pair reconstruction framework to model urban environments such as cities. In addition to the second camera, they integrated an object recognition module that automatically detected cars in the input video streams and localized them in 3D. Once located, the cars could be removed from the textures and excluded from MVS reconstruction. The car recognition helped reduce clutter, and build more robust models. The occluded surfaces revealed by the cars were replaced by generic car models which helped reduce visible artifacts.

Furukawa and Ponce [41] create a (quasi) dense set of rectangular patches from a set of input images. Their method creates patches covering the surfaces visible in the input images, they subsequently create a mesh representation for their surfaces. To ease gathering of image data, Goesele et al. [46] run SfM on community photos to create sparse 3D points. Their method iteratively grows surfaces from these points. Their goal is to create high quality models, which necessitated estimating high-quality depth maps. They also show examples of merging the resulting depth maps.

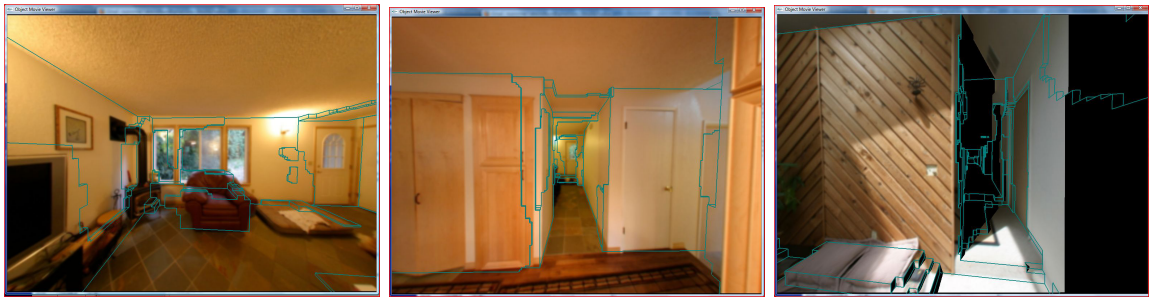


Figure 2.3: Image-based geometry of the house depicted in Figure 2.1 generated via Furukawa et al.[39]. This model provides a photo-realistic visualization of home interior, but is difficult to edit due to its structural representation and lack of texture in occluded regions.

Furukawa et al. [39, 40] improved automatic reconstruction by using a Manhattan-world assumption (scenes consisting predominantly of piece-wise planar surfaces with dominant directions) to help constrain the reconstruction problem. Their system is a fully automated 3D reconstruction and visualization system for architectural scenes (interiors and exteriors). The final output produces simplified 3D models with texture.

Using *a priori* knowledge about scenes such as home interiors also aides in reconstruction. Hedau et al. [53] fit a cuboid to the spatial layout of a cluttered indoor scene using a single image. They model the global room space with a parametric 3D cuboid, by iteratively localizing clutter and refitting the model. They introduce a structured learning algorithm that chooses a set of parameters to minimize error based on global perspective cues.

In another approach that leverages the physical constraint that 3D objects interact with each other, Lee et al. [74] proposed the use of volumetric reasoning between objects and surfaces of a room. They parametrically represented the 3D volume of objects and rooms, and applied constraints for volumetric reasoning, such as spatial exclusion and containment. They used line segments and vanishing points of a single image to create a set of possible cuboids that could be used to model the room. Using simple volumetric constraints such as finite volume, spatial exclusion, and the fact that all the objects are bounded in the space by the room, they searched for a cuboid configuration that best models the room. Although this work is promising, it has a high error rate and only attempts to fit a few cuboid primitives.

Xiao et al. [126] present a 3D reconstruction and visualization system to automatically produce clean and vertically-aligned cuboid texture-mapped 3D models for large indoor scenes, from ground-level photographs and 3D laser scans. The key component is an algorithm called "Inverse CSG" for reconstructing a scene in a Constructive Solid Geometry (CSG) representation consisting of volumetric primitives.

There have been efforts to model architectural structure using higher level primitives such as parameterized shapes, and generative grammars. These methods have the potential to be much more useful for editing due to their structural representation.

In an effort to create models from parameterized reusable components, Dick et al. [33] reconstruct whole buildings by optimizing a generative model of architectural primitives. They describe an approach to automatically acquire three dimensional architectural models from short image sequences. They model a building as a set of walls together with a "Lego" kit of parameterized primitives, such as doors or windows. They use a Bayesian framework to create the generative model.

Procedural representations provide powerful means for generating complex archetectural structures [69, 79, 117], however it is difficult to model existing structures with such tools. Müller et al. [86] use procedural shape grammars to reconstruct building façades composed of regular, repeating structures. Their ideal input imagery consists of downtown façades with repetitive tiles. The input is a single rectified façade image. From this image, they find the dimensions, ratios and spacing of architectural elements automatically. They automatically derive shape grammar rules from façade images and build a rule base for procedural modeling.

Analysis: Although some parameterized and procedural methods are gaining traction, most automatic construction techniques produce either point clouds or unstructured meshes; such models are sufficient for rendering but not editing, since they lack architectural primitives that can be selected and manipulated. The meshes also lack the connectivity, structural representations, and constraints of easy-to-edit models. The problem of "bad geometry" is widespread through the modeling community; most tools have help pages devoted to walking users through the problem. There is some hope that mesh repair [65] and parameterization

[58] techniques can transform these automatically created meshes into editable structures. However, the fundamental problem remains: the best fit meshes are akin to a shrink wrapped cover enclosing the structure. There is no distinction between structures such as a floor, and the furniture that sits on top of it. Like most interactive techniques, none of these systems address the problems associated with authoring and rendering edits that depart from the original structures being modeled.

The degree of usable repetition in a home interior remains an open question. The amount of variability in home interiors might severely limit the applicability of Dick's generative models or shape grammars. Shape grammars might be useful in specific instances such as tiled floors or wall paper patterns, but it is unclear how to adapt a 2D shape grammar to view-dependent textures.

Summary: Automatic methods have been shown to create reasonably accurate geometry for image-based models with little user input. The automatic methods that produce higher level primitives hold promise for quickly creating editable models due to their underlying geometric representation. No matter how good an automatic method becomes, I believe there will always be errors that a human will need to fix through interaction. When these automatic techniques become robust enough for general use, they should be very useful for creating editable models. The efforts designed to produce re-usable components (such as shape grammars) have potential to save user effort in modeling and subsequently editing highly repetitive patterns.

2.1.4 Automatic model creation with additional hardware sensors

Automatically constructing models can be assisted using additional hardware and active depth sensing. One such system is exemplified by Bahmutov et al. [8] which used the *Model-Camera* system that employs a structured light pattern to acquire sparse depth and texture data at interactive rates. This data was further combined to form a texture-mapped triangle mesh of building interiors.

With the emergence of widely-available commodity sensors that incorporate both a depth sensor and an RGB color camera, utilizing these sensors to capture dense depth maps and

to quickly create geometric models has become another active area of research. The sensors often utilize either structured light or use time-of-flight sensors to provide a depth map to the user via a standard programming interface. Combination hardware comprising of both color cameras and depth sensors, RGB-D sensors, typically provide output of both color video and synchronized calibrated depth data at 15 to 30 fps. The color video stream is usually lower quality than one would expect from a video camera. RGB-D sensors are widely available and easily accessible for a range of consumer and industrial applications as evidenced by the large number of RGBD camera brands such as: Microsoft Kinect[1], Intel[2], Apple[3], SoftKinetic DepthSense[4], and industrial sensors such as the Mesa Imaging Swiss Ranger[5] to name a few.

Using commodity[3] hardware, Henry et al. [55] built a system to reconstruct building interiors. They approached the problem using a surfel representation [96] rather than retaining a point cloud as the final representation. The resulting model is highly accurate, however they do not generate a polygon mesh or retain the RGB image data that could be used as texture. In a related system, Du et al.[34] built an interactive mobile system for 3D mapping and modeling. Their system also builds highly accurate point cloud models.

Newcombe et al.'s [88] KinectFusion system and subsequent work by Salas-Moreno et al.[104] 3D Simultaneous Localization and Mapping, SLAM, uses commodity RGB-D sensors to create accurate polygonal representations of interior environments. The KinectFusion and SLAM++ are interactive and also enable predictions of accurate camera model tracking at each live frame, thus providing both accurate geometric models and realtime RGB-D sensor position and direction. The intent of the system is to enable augmented reality systems that allows virtual geometry to interact with the geometry modeled in the system. The interaction is rendered over the RGB-D sensor's video stream, typically using the modeled geometry for physics simulations and occlusion culling. Their goal is not to render high

[1] `http://www.microsoft.com/en-us/kinectforwindowsdev/default.aspx`

[2] `http://software.intel.com/en-us/vcsource/tools/perceptual-computing-sdk`

[3] Primesense, purchased by Apple `http://www.primesense.com`

[4] `http://www.softkinetic.com`

[5] `http://www.mesa-imaging.ch`

quality image-based models. Unlike Henry et al.'s and Du et al.'s work, the KinectFusion system creates polygonal models rather than surfels and point clouds, however this work still discards the image data after scene reconstruction because it is unnecessary for their application.

Summary: In many respects the system that utilize the depth sensors closely resemble the automatic methods described in Section 2.1.3. Notable systems such as KinectFusion produce compelling geometric models very quickly. Unfortunately depth camera based methods lack visual fidelity due to the low resolution camera imagery, and like other automatic methods, the resulting geometric models are not generally suitable for editing. Systems like KinectFusion hold great potential if combined with high quality imagery because the detailed geometry could be used to quickly create a highly detailed geometric representation with high quality texture. The problem remains of creating well-formed editable geometry with this approach. I do not address the problem of creating well-formed editable geometry from the dense geometric output of the RGB-D sensor aided approaches, however the geometric techniques and lessons learned outlined in Chapter 4 informs future work as to how to create such a system.

2.2 Rendering and Navigation

2.2.1 Rendering and Navigation

Successfully navigating through an image-based model is a difficult task unless one is familiar with the model, and has experience identifying the best transitions and viewpoints. Discovery of model structure and visually pleasing transitions is often a matter of trial and error. Automatically planning a path through the most interesting objects and viewpoints, revealing structural layout, and animating between viewpoints with compelling 3D transitions that convey the spatial relationships between views can be a difficult problem.

Following the success of Photo tourism [114] and Photosynth [84], there has been some research into how to best navigate image based models. Snavely et al. [113] in *Finding Paths through the Worlds Photos* show that viewpoints often cluster along certain paths in a large set of community or personal photos. These paths are largely specific to the scene being

photographed, and follow interesting regions and viewpoints. They use this trend to discover a range of such paths and turn them into controls for image-based rendering, automatically creating controls for orbits, panoramas, canonical views, and optimal paths between views for the image set.

Photo Tours [71] further refines this idea and automatically creates curated paths. This system automatically creates tours of popular tourist destinations from geotagged user-contributed photos obtained from the Internet. These photo tours visualize each site's most popular viewpoints on a tour that maximizes visual quality and traversal efficiency. The path planning problem is framed as a form of the Traveling Salesman Problem on a graph with photos as nodes and transition costs on edges and pairs of edges, permitting an efficient solution even for large graphs containing thousands of photos.

Analysis: In the context of a home interior, the idea that viewpoints cluster around interesting features is not valid. There are salient viewpoints in the relatively sparse number of photographs, but data capture is more concerned with scene coverage than with taking interesting photographs. Thus, we cannot draw saliency conclusions from viewpoint frequency. Therefore, the issue remains of finding salient photos and viewpoints in home interiors. Neither Snavely et al. or Kushal et al. explicitly address how a tour might explain architectural structure or layout. Any pathway that conveys the underlying architectural structure seems to be a result of the source image sampling rather than intentionally choosing a path that displays the architectural structure.

It would be interesting to augment Kushal's Photo Tours work with user input in order to leverage the user's expertise and knowledge of the scene. This is especially important in the case of curating tours of home interiors.

Summary: Navigating large image-based models consisting of thousands of photos is a challenging problem, especially in terms of discovering interesting viewpoints and salient features. There are outstanding problems in automatically finding tours of home interiors that convey layout and display salient viewpoints. The work outlined in Chapter 3 begins to address some of the issues with finding navigation paths that are both visually pleasing and enable story telling. While the work does not explicitly address home interiors, it would

be interesting to extend the themes presented to address this problem.

2.3 Relighting and Synthetic Objects

There are several levels to editing an image-based model and subsequently rendering the model in a photorealistic manner. At the most basic level, are edits that change the illumination or material properties of objects in the model. At another level there are edits that insert synthetic novel elements into the model. At a final level, there are edits that rearrange, remove, or replicate the elements of the model itself. Each level of editing presents its own unique challenges and problems.

In this section, I will discuss the different approaches and work related to editing image-based models. I do not go into the details related to modifying the geometric primitives underlying objects, but instead discuss the topics related to changing the appearance of an image-base object in a photorealistic manner.

2.3.1 Inverse global illumination

One of the most fundamental manners in which to edit an object is to re-render the object with different lighting or material properties. The goal of inverse global illumination (sometimes referred to as inverse rendering) is to recover the material properties of the image-based geometry given accurate geometry, photographs (which are observations of how the geometry looks under illumination), and illumination. However, with image-based geometry, we have only geometry with varying degrees of accuracy and photographs. We do not have illumination information unless it is specifically captured.

Yu et al. [128] demonstrate that inverse global illumination can recover both diffuse and specular material properties of a single room. The limitations of this approach are that the source photographs must be photometrically calibrated (i.e., taken under calibrated lighting conditions); the room must be well modeled with well defined light sources; and regions with the same material properties must be delineated. Within each region of the same material properties, Yu et al. assume that specular reflectance remains constant while the diffuse reflectances can vary and produce an albedo map.

Going into more detail, Yu et al. minimize the following objective function for each material property M and surface points P_i:

$$\underset{\rho_d,\rho_s,\alpha}{\operatorname{argmin}} \sum_{i\,\varepsilon\,M} \left| L_i - \frac{\rho_d}{\pi} I_i - \rho_s K(\alpha,\Theta) I_i \right| \tag{2.1}$$

where L_i is an observation, I_i is the irradiance at the observation calculated from known light sources, $\frac{\rho_d}{\pi} I_i$ is the diffuse term, and $\rho_s K(\alpha,\Theta)$ is a specular lighting term. Yu et al. used the Ward [124] shading model defined in A.12.

To recover a diffuse albedo map $\rho_d(i)$ for the material over the span of a geometric face, rather than a single diffuse component for the entire material, Yu et al. use the following relationship:

$$\rho_d(i) = \pi D(i)/I(i) \tag{2.2a}$$

$$I(i) = D(i) + S(i) \tag{2.2b}$$

where $D(i)$ is a diffuse radiance map, $S(i)$ is a specular radiance map computed from the recovered specular lighting term, and $I(i)$ is the irradiance map. Yu et al. also noted that this approach has several problems, namely that $S(i)$ may be much larger than the diffuse component $D(i)$. As a result, relatively small errors in the estimated $S(i)$ will cause large relative errors in $D(i)$ and thus $\rho_d(i)$.

Since the initial inverse global illumination work, there have been a series of efforts designed to characterize viable solutions, and relax the constraints of the problem to find more generally applicable solutions. Ramamoorthi and Hanrahan [100] introduced a mathematical framework for inverse rendering under general illumination conditions. They describe the problem in terms of a signal-processing framework that describes the reflected light field as a convolution of the lighting and BRDF, and inverse rendering as a deconvolution. Assuming homogeneous surfaces with no self shadowing, they give an algorithm to factor the reflected light field of a scene captured under unknown illumination into the reflectance and illumination components. They also provide a theoretical framework that predicts under which conditions this can be achieved and how the recoverable detail of the reflectance functions is determined by the frequency content of the incident illumination.

Given known scene geometry and constant specular reflectance over the surface, Nishino et al. [93, 92] use a sparse set of images to separate the diffuse from the specular reflectance of an object captured under a single point light source with a fixed but unknown position. Given the object's geometry, they then compute an initial estimate for the illumination and determine both the illumination and the parameters of a simplified Torrance-Sparrow model. Both of these methods assume known scene geometry and constant specular reflectance over the surface. They estimate the diffuse component by taking the minimum pixel value of a 3D point value under all views. This might be a valid diffuse estimate for a single light source with no inter-reflection, but it is at best an upper bounds estimate in the general case.

Georghiades [44], with somewhat different constraints, further relaxes the input by estimating geometry, a single point light source, in addition to the spatially varying diffuse reflectance, and the parameters of a single non-Lambertian reflectance model from a sparse set of images with fixed viewpoint. In this system, they estimate the shape of an object (faces and spheres), and a single reflectance model with a constant specular term and varying albedo over the surface of the face by varying the light source. They simultaneously solve for light positions, a single global specular component, and per pixel surface normals and diffuse reflections. In this case, the specularity is useful because it disambiguates a problem with Lambertian surfaces and unknown light sources termed the Generalized Bas-Relief[11].

Habor et al. [49] reduce the dependency on perfect knowledge of light sources, and well segmented material properties, with an approach for recovering the reflectance of a static scene given highly accurate geometry from a collection of images taken under distant, unknown, and varying illumination. They simultaneously estimate the per-image incident illumination and the per-surface point reflectance. The wavelet framework allows for incorporating various non-Lambertian reflection models, but neglects inter-reflection. Combined with an accurate MVS reconstruction, they are able to recover the reflectance of a scene by restricting the space of BRDFs to linear combinations of basis BRDFs.

Analysis: Inverse global illumination, often termed inverse rendering, demonstrates that given enough information about the model, and some assumptions, it is possible to deduce

material properties without extensive measurement [83] or specialized analytical apparatus such as a gonioreflectometer. In limited cases it is possible, with accurate models, to simultaneously estimate both a small finite number of light sources and restricted material properties.

Summary: With highly accurate geometric models, inverse global illumination methods are capable of determining both light source estimates for distant illumination and estimates of non-Lambertian BRDFs. The main limitation of these methods is that they require very accurate geometric models. The effects of local shadowing and inter-reflection are large for architectural interiors and can probably not be ignored. While Yu et al. [128] does handle local shadowing and inter-reflections, it also demonstrates the necessity to explicitly model material property boundaries and light sources.

2.3.2 Intrinsic Images

Intrinsic images are a convenient image representation where a pixel's color is defined as the product of intrinsic characteristics of an image such as reflectance and illumination. Although more complex than a typical RGB representation, intrinsic images are useful for many operations especially when the material properties, shape, and lighting information of the objects depicted in the image are unknown. Given that image-based models might have coarse geometry and inverse global illumination methods discover material and/or lighting under limited conditions, it is useful to consider intrinsic images.

There is a long history of decomposing photographs into intrinsic images to discover information about a scene, recolor, or relight the image of the scene. Typically the methods for decomposing images into intrinsic components are grounded in broad assumptions [73, 13, 15] about the scene, namely shading variation is spatially slow, reflection is Lambertion, and albedo consists of piecewise constant patches with potentially sharp boundaries. Barrow and Tenenbaum [10] suggested separating images into intrinsic (veridical) characteristics such as range, orientation, reflectance, and incident light. Commonly, intrinsic image decomposition is concerned with separating images into illumination and albedo components. Recent work further separates images into three components, direct illumination,

indirect illumination, and albedo.

User-guided intrinsic image decomposition, such as the method demonstrated by Bousseau et al. [14] can be very useful for complex textures. Bousseau et al. obtain intrinsic images from photographs with a user guided optimization approach. In order to solve the problem of decomposing an image into a product of diffuse reflectance and illumination, they restrict reflectance values to a local color plane, and reduce illumination to a scalar. They then formulate the problem as a linear least squares system operating on overlapping windows. Users can further constrain the problem by indicating regions of constant reflectance or illumination. Due to the overlapping windows, these local constraints propagate throughout the image.

More recently, Laffont et al. [72] estimated intrinsic image decomposition of calibrated photographs from multiple views of an outdoor scene with imprecise geometry created from MVS reconstruction, and a single light probe. Due to the imprecise geometry and the use of a single light probe, it is difficult to determine direct luminance at each 3D point in the scene (i.e., whether the point lies in a shadow). To solve this problem, they note that pixels with the same reflectance trace similar paths in a reflectance space parameterized by sun light. By clustering the paths, they group pixels with similar reflectances into an aggregate path. A single material will form two disjoint but continuous paths if the material is partially in shadow. By combining pairs of these disjoint paths to form continuous curve, they create a single curve that represents the reflectance under both indirect and direct sunlight. The parametric position of a point along the curve indicates sunlight visibility and reflectance. Using this information, they then separate reflectance from illumination, and decompose the illumination into sun, sky and an indirect layer.

Many intrinsic image applications rely on a pure diffuse reflection model, despite the fact that specular reflection is common in real world materials. It is important to be able to detect and possibly remove specular components. Artusi et al. [4] survey methods of specularity removal. In short, Artusi states that currently available methods are able to achieve good component separation results, but are limited by the conditions of their applicability. In particular, most of the techniques rely on a specific reflection model and assume that the specular reflectance varies insignificantly with wavelength, which means that its color is

essentially the same as that of the light source. Together with noise sensitivity, this limitation reduces the range of applications where the current methods can be used.

Analysis: Laffont assumes a Lambertion reflection model and a sparse set of materials properties. This work is particularly relevant for image-based models in that it might help recover diffuse material properties. It is difficult to determine how well this method will work with a scene having many material properties (e.g., highly variable texture).

Summary: The inverse global illumination problem is difficult to solve with the incomplete information typically acquired with image-based models. Estimating intrinsic images can help approximate missing lighting information and material properties where Lambertian assumptions hold. In general, you cannot relight from intrinsic images, since the illumination component is the combination of both geometric shading and incident light, both of which remain unknown.

2.3.3 Image-based relighting

Image-based relighting sometimes refers to interpolation methods used to simulate different lighting conditions in a scene. The scene is captured under different lighting conditions, and these captured images are termed "basis" images [91]. Novel lighting conditions can be simulated by a linear combination of the basis images.

Choudhury et al. [21] survey image-based relighting techniques, and provides a classification of various approaches. The main advantage of image-based relighting is that the rendering time is independent of scene complexity because the rendering is actually a process of manipulating image pixels, instead of simulating light transport. However, due to the lack of input data, image-based lighting techniques are not appropriate for editing image-based models captured under constant lighting conditions.

2.3.4 Synthetic objects

Another goal of editing images, and image-based models, is to insert novel geometric elements into the scene. This difficult task requires that the objects be lit in a manner

consistent with nearby surfaces, and that the interplay of light between the objects and their surroundings be properly simulated. Specifically, the objects should cast shadows and appear in reflections, as well as refract, focus, and emit light just as natural objects would. Debevec et al. [30] pioneered much of research in this area by measuring the scene radiance and global illumination to add synthetic objects to image-based models with appropriate lighting. This method uses a high dynamic range image-based model of the scene to illuminate the new objects. The illumination is separated into three components: the distant scene, the local scene, and the synthetic objects. The distant scene is assumed to be photometrically unaffected by the objects, obviating the need for its reflectance model information. They construct an image-based model from an approximate geometric model of the scene and estimate Lambertian material properties via a simple iterative refinement. The synthetic objects are able to cast shadows and reflect light onto the model. Renderings are created using a standard global illumination method by simulating the interaction of light amongst the three components. A light probe is used to measure the incident illumination at the location of the synthetic objects, and to capture the illumination for the distant scene. The resulting image is a composite of the original image and the rendered synthetic object including reflections and shadows.

Unger et al. [120, 121] extend the idea of capturing illumination by using thousands of light probes. They note that image-based lighting[1] using a single light probe can only represent an angular variation and assumes spatially invariant lighting. This means that a single light probe can accurately approximate the illumination for small synthetic objects. However, large objects or the entire scene require much denser sampling to fully represent the strong spatially varying illumination found in real scenes. Unger et al.'s goal is to demonstrate photo-realistic rendering using standard techniques by incorporating strongly spatially variant illumination captured from real scenes into the rendering pipeline. They present a dense volumetric real-time sampling of 3D sets of thousands of light probes together with efficient methods for data representations supporting data reduction, editing and fast rendering.

[1]Not to be confused with image-based relighting

More recently, Karsch et al. [67] proposed a method of realistically inserting synthetic objects into existing photographs without requiring access to the scene or any additional scene measurements. With a single image and a small amount of annotation, the method creates a physical model of the scene that is suitable for realistically rendering synthetic objects with diffuse, specular, and even glowing materials while accounting for lighting interactions between the objects and the scene. The interesting aspect to this work is that it refines illumination estimates using intrinsic image estimates, thus relaxing the need for explicit reflectance properties. The intrinsic images are expressed as a sum of reflected direct and indirect light.

$$B = \rho S, B = D + I, I = \Gamma, B = D + \rho\Gamma$$

The original image B can be expressed as the product of albedo ρ and shading S, as well as the sum of reflected direct light D and reflected indirect light I. Indirect irradiance Γ is computed by gathering radiance values at each 3D patch of geometry. The gathered radiance values are obtained by sampling observed pixel values from the original image. With Lambertian assumptions, they developed an objective function to decompose an image B into albedo ρ and direct light D.

Given the direct light D, they can refine the illumination estimates with a non-linear least squares model, where they find the lighting parameters that minimize the difference between between a rendered model and the original image.

Analysis: Much of the research devoted to inserting synthetic objects into a scene focusses on solving the problem of finding illumination in the scene. While using a single light probe is useful for detecting the illumination of a smaller object, it is not sufficient for discovering the illumination of a home interior which exhibits spatially variant illumination. Denser illumination sampling requires additional hardware, although some of the techniques used for compressing light data might be interesting for storage and computation.

Karsch et al. find illumination sources with sufficient accuracy for inserting synthetic objects into a scene. However their methods do not appear sufficient to re-render existing structures. Given the vastly greater amount of data available in image-based models compared to a single image, it might be possible to adapt this approach to more accurately

estimate light sources in image-based models.

Summary: Given a reasonable estimate of an image-based model's lighting and material properties, there are very good techniques available to relight and insert synthetic objects into a scene. Currently, the methods are not sufficient for relighting or editing the model itself.

2.4 Remodeling

The ultimate goal of editing image-based models is to re-arrange and change elements in the scene itself. This task is in some ways more difficult than inserting synthetic objects into a scene. As a model is edited, dis-occluded texture is revealed and must be re-constructed or filled in seamlessly. As a model changes shape, these changes should be appropriately propagated to its texture. In addition, interactive model editing requires an interaction paradigm and a geometric representation that is appropriate to the task. The image-based edits should cast shadows and appear in reflections, and emit light just as real objects would.

2.4.1 Plenoptic Image Editing

Early efforts to edit image-based models focused directly on pixel and voxel operations such as painting. Seitz and Kutulakos [107] describe a class of interactive image editing operations termed plenoptic image editing. These operations are designed to maintain photometric consistency between multiple images of a physical 3D object. The approach works by extending operations such as image painting, scissoring, and morphing so that they alter an object's plenoptic function in a physically-consistent way, thereby affecting object appearance from all viewpoints simultaneously. The key to their implementation is that the sparse plenoptic samples (e.g., photographs) are converted to a voxel space with an estimated per voxel Lambertian radiance model. The image editing operations operate at the voxel level, thus edits are propagated through all source images via re-rendering the voxel space from the viewpoint of the image. The radiance modeling method is restricted to Lambertian surfaces with light sources at infinity and does not account for shadows or local reflections.

Subsequently, there were some efforts to compose and deform light fields. Chen et al. [20]

enabled animators to deform light fields. Their pipeline can be used to deform complex objects, such as furry toys, while maintaining photo-realistic quality. The system worked by decomposing a light field into sub-components that could be recombined and rendered after deformation. They handle visibility changes due to occlusion among sub-light fields. However, to ensure consistent illumination of objects after they have been deformed, the light fields must be captured with the light source fixed to the camera, rather than being fixed to the object. This introduces shading and shadow artifacts that cause the resulting image to appear unnatural.

The *LightShop* system presented by Horn et al. [59] allows a user to interactively manipulate, composite and render multiple light fields. The primary limitation of LightShop is that the input is a 4D light field with fixed illumination. This means compositing different light fields can result in an unnatural appearance as there is no light transfer between composited objects.

2.4.2 Image-based model editing

Oh et al. [94] also used a painting approach to edit image-based models created from a single photograph. The core of their system is a user editable representation consisting of layers of images with depth. Interactive editing, using a variety of editing operations, permits the definition and manipulation of image-based scenes. The primary editing tool is a "clone brushing tool," that permits the distortion-free copying of parts of an image. They performed relighting by separating the luminance channel from RGB texture. The initial separation was automated, and relighting could occur either by painting directly on the luminance channel or by specifying light sources.

Recently Zheng et al. [131] introduced *Interactive Images*, a semi-automatic system that constructs elements of a scene with cuboid-proxies and user interaction. The system simulates image editing by performing edits on the cuboid proxies to mimic real-world behavior. This system is interesting due to its semi-automatic proxy fitting, automatic object layer decomposition, and constrained geometry editing. The scene is decomposed into a background layer, and layers of cuboid proxies. Occlusion holes are filled by image synthesis

[9]. They estimate light sources from user input and create simple shadows during editing. During editing, they also support real-time interactions with optional collision handling (using the open-source physics simulator Box2D [19]). Importantly, they allow the user to manipulate objects while respecting object layer relationships. The user initiates an edit using a selected single cuboid. Then the edit is automatically propagated with constraints across the other proxies to preserve both the intra-object characteristics as well as their non-local mutual relationships. Their use of iWire and higher level shape manipulators [43, 127, 132] appear to be well suited for manipulating the geometry in this system.

Analysis: The interaction metaphor and capabilities of Zheng et al. are promising for editing objects in a room. However, none of the systems support large-scale geometric changes such as removing entire walls. Zheng et al.'s work is concurrent with, and complimentary to the *Image-based remodeling* system [24] (described in Chapter 4).

Summary: The ability to edit image-based models is gradually improving, but there are several unsolved problems. There is no solution for editing large scale geometric structures in which the edit produces a large occlusion hole. Filling in occlusion holes remains a difficult problem for which I propose a solution in (described in Chapter 4). However, this problem is especially difficult for texture that is entirely occluded in all viewpoints. Propagating geometric changes to the associated view-dependent texture in a manner other than scaling or cropping has not been studied and is not addressed in this thesis. Finally, relighting edits remains largely unaddressed, except in the case of Oh et al., where they provide a manual touchup mechanism.

2.5 Limitations of image-based geometry

The discussion in this chapter demonstrates the limitations of current approaches used to create, edit and navigate through image-based geometry. To date, most of the efforts related to creating image-based geometry have been limited to creating models, relighting and inserting synthetic objects into the model under specialized circumstances, and some limited editing of scene elements such as tables and clutter. Creating truly editable image-based geometry has not yet been achieved.

In the following chapters I describe the research that comprises my thesis.

Chapter 3

PARALLAX PHOTOGRAPHY

In this work [1] we address one aspect of navigation and viewing image-based geometry. We present an approach that converts an image-based model exhibiting parallax into a cinematic effect with simulated, smooth camera motion that highlights the sense of 3D parallax in the scene. We develop a taxonomy of the cinematic conventions for these effects distilled from observations of documentary film footage and organized by the number of subjects of interest in the scene. We present an automatic, content-aware approach to apply these cinematic conventions to an input image-based model. We use a face detector to identify subjects of interest. Then, we optimize for a camera path that conforms to a cinematic convention, maximizes apparent parallax, and avoids missing information. We describe a GPU-accelerated, temporally coherent rendering algorithm that allows users to create more complex camera motion interactively, while experimenting with effects such as focal length, depth of field, and selective, depth-based desaturation or brightening. We evaluate and demonstrate our approach on a wide variety of scenes and present a user study that compares our 3D cinematic effects to their 2D counterparts. I particularly focused on user interaction, creating the GPU-accelerated rendering algorithms, and runtime effects.

3.1 Introduction

Documentary filmmakers commonly use photographs to tell a story. However, rather than placing photographs motionless on the screen, filmmakers have long used a cinematic technique called "pan & zoom," or "pan & scan," to move the camera across the images and give them more life. The earliest such effects were done manually with photos pasted on animation stands, but they are now generally created digitally. This technique, which goes

[1] Parallax Photography [130] was published the Proceedings of Graphics Interface 2009. `http://dl.acm.org/citation.cfm?id=1555909`.

by the name of the "Ken Burns effect," after the documentary filmmaker who popularized it, is now a ubiquitous feature in consumer photography software such as Apple iPhoto, Google Picasa, Photoshop Elements, and Microsoft PhotoStory.

In recent years, filmmakers have begun infusing photographs with more realism by adding depth to them, resulting in *motion parallax* between near and far parts of the scene as the camera pans over a still scene. This cinematic effect, which we will call *3D pan & scan*, is now used extensively in documentary filmmaking, as well as TV commercials and other media, and is replacing traditional 2D camera motion, because it provides a more compelling and lifelike experience.

However, creating such effects from a still photo is painstakingly difficult. The photo must be manually separated into different layers, and each layer's motion animated separately. In addition, the background layers are typically painted in by hand so that no holes appear when a foreground layer is animated away from its original position.[1]

In this paper we look at how 3D pan & scan effects can be created much more easily, albeit with a small amount of additional input. Indeed, our goal is to make creating such cinematic effects so easy that regular users can create them from their snapshots and include them in their photo slide shows with little or no effort. To that end, we propose a solution to the following problem: given a small portion of a light field [77, 48], produce a 3D pan and scan effect automatically (or semi-automatically if the user wishes to influence its content). In most of our examples, the input light field is captured with and constructed from a few photographs from a hand-held camera. We also include results from two one-shot, multi-viewpoint cameras, for which we envision our solution will be most useful. Some predict that the commodity camera of the future will have this capability [76] (perhaps beginning with the recently announced consumer stereo camera "Fuji FinePix Real3D").

The 3D pan & scan effects are generated to satisfy two main design goals:

1. The results should conform to the cinematic conventions of pan & scan effects currently used in documentary films.

2. The conventions should be applied in a fashion that respects the content and limitations

[1] `http://blogs.adobe.com/bobddv/2006/09/son_of_ben_kurns.html`

of the input data.

Our approach takes as input a light field representation that contains enough information to infer depth for a small range of viewpoints. For static scenes, such light fields can be captured with a standard hand held camera [98] by determining camera pose and scene depth with computer vision algorithms, namely structure-from-motion [51] and multi-view stereo [106]. Capturing and inferring this type of information from a single shot has also received significant attention in recent years. There are now several camera designs for capturing light fields [90, 45, 78] from which scene depth can be estimated [106]. Other specialized devices, such as coded imaging systems, capture single viewpoints with depth [75].

Light fields with depth have the advantage that they can be relatively sparse and still lead to high quality renderings [48, 78] for scenes without strong view-dependent lighting effects such as mirrored surfaces. However, such sparse inputs, taken over a small spatial range of viewpoints or even a single viewpoint, present limitations: novel viewpoints must stay near the small input set, and even then some portions of the scene are not observed and thus will appear as holes in new renderings. Our approach is designed to take these limitations into account when producing 3D pan & scan effects.

Our solution processes the input to produce 3D pan & scan effects automatically or semi-automatically to satisfy our design goals. To achieve the first goal, we describe a simple taxonomy of pan & scan effects distilled from observing 22 hours of documentary films that heavily employ them. This taxonomy enables various communicative goals, such as "create an establishing shot of the entire scene", or "transition from the first subject of interest to the second". Second, we describe algorithms for analyzing the scene and automatically producing camera paths and effects according to our taxonomy. Our solution then applies the appropriate effect by searching the range of viewpoints for a linear camera path that satisfies cinematic conventions while avoiding missing information and holes, and maximizing the apparent parallax in the 3D cinematic effect. Third, we describe GPU-accelerated rendering algorithms with several novel features: (1) a method to interleave pixel colors and camera source IDs to multiplex rendering and guarantee optimal use of GPU memory; (2) the first GPU-accelerated version of the soft-z [99] technique, which minimizes temporal artifacts; and (3) a GPU-accelerated inverse soft-z approach to fill small holes and gaps in the rendered

output.

In the rest of this paper we describe the components of our approach, which include a taxonomy of the camera moves and other image effects found in documentary films (Section 3.3); techniques for automatically computing 3D pan & scan effects that follow this taxonomy (Section 3.4); a brief overview of our representation of a light field with depth and how we construct it from a few photographs (Section 3.5); and finally two rendering algorithms, both real-time (Section 3.6.1) and off-line (Section 3.6.2). We then demonstrate results for multiple photos taken with a single camera, as well as two multi-viewpoint cameras (Section 3.7). Finally, we describe the results of a user study with 145 subjects that compares the effectiveness of 3D vs. 2D pan & scan effects (Section 3.8).

3.2 Related work

The process of creating a 3D pan & scan effect is challenging and time consuming. There are a number of techniques that help in creating 3D fly-throughs from a single image, such as Tour Into the Picture [60] and the work of Oh et al.[94], though the task remains largely manual. Hoiem et al.[57] describe a completely automatic approach that hallucinates depths from a single image. While their results are impressive, substantially better results can be obtained with multiple photographs of a given scene.

To that end, image-based rendering (IBR) techniques use multiple captured images to support the rendering of novel viewpoints [66]. Our system builds a representation of a small portion of the 4D light field [77, 48] that can be used to render a spatially restricted range of virtual viewpoints, as well as sample a virtual aperture to simulate depth of field. Rendering novel viewpoints of a scene by re-sampling a set of captured images is a well-studied problem [17]. IBR techniques vary in how much they rely on constructing a geometric proxy to allow a ray from one image to be projected into the new view. Since we are concerned primarily with a small region of the light field, we are able to construct a proxy by determining the depths for each of the input images using multi-view stereo [129], similar to Heigl et al. [54]. This approach provides us the benefits of a much denser light field from only a small number of input images. Our technique merges a set of images with depth in a spirit similar to the Layered Depth Image (LDI) [108]. However, we compute depths for

segments, and also perform the final merge at render time. Zitnick et al. [133] also use multi-view stereo and real-time rendering in their system for multi-viewpoint video, though they only allow novel view synthesis between pairs of input viewpoints, arranged on a line or arc. Most IBR systems are designed to operate across a much wider range of viewpoints than ours and typically use multiple capture devices and a more controlled environment [118, 76]. To date, the major application of capturing a small range of viewpoints, such as ours, has been re-focusing [85, 89].

A number of papers have used advanced graphics hardware to accelerate the rendering of imagery captured from a collection of viewpoints. The early work on light fields [77, 48] rendered new images by interpolating the colors seen along rays. The lightfield was first resampled from the input images. The GPU was used to quickly index into a lightfield data structure. In one of the early works leveraging per-pixel depth, Pulli et al. [99] created a textured triangle mesh from each depth image and rendered and blended with constant weights. They also introduced the notion of a soft-z buffer to deal with slight inaccuracies in depth estimation. We take a similar approach but are able to deal with much more complex geometries, use a per-pixel weighting, and have encoded the first soft-z into the GPU acceleration. Buehler et al. [17] rendered per-pixel weighted textured triangle meshes (one simple mesh per light field). We use a similar per-pixel weighting, but are also able to deal with much more complex and accurate geometries. We also use a "reverse soft-z" buffer to fill holes caused by disocclusions during rendering.

Automatic cinematography that follows common film idioms has been explored in the context of virtual environments, e.g., by He et al. [52]; we focus on the idioms used in 3D pan & scan effects.

3.3 3D pan & scan effects

Our first design goal is to automatically create 3D pan & scan effects that follow the conventions in documentary films. To that end, we examined 22 hours of documentary footage in order to extract the most common types of camera moves and image effects. We examined both films that employ 2D pan & scan effects (18.5 hours, from the Ken Burns films *The Civil War, Jazz,* and *Baseball*) and the more recent 3D pan & scan technique (3.5 hours, *The*

Subjects of interest	Camera moves	Image effects
0	Establishing dolly, Dolly-out	
1	Dolly in/out, Dolly zoom	Change *DOF*, Saturation/brightness
2	Dolly	Pull focus, Change *DOF*

Table 3.1: A taxonomy of camera moves and image effects. DOF *refers to depth of field.*

Kid Stays in the Picture, and *Riding Giants*). These films contained 97 minutes of 2D effects and 16 minutes of 3D effects. Of these 113 minutes, only 9 exhibited non-linear camera paths; we thus ignore these in our taxonomy (though, as described in Section 3.4.4, curved paths can be created using our interactive authoring tool). Of the remaining 104 minutes, 102 are covered by the taxonomy in Table 3.1 and described in detail below (including 13 minutes that use a concatenation of two of the effects in our taxonomy).

We organize our taxonomy according to the number of "subjects of interest" in a scene: zero, one, or two. For each number there are several possible camera moves. There are also several possible image effects, such as changes in saturation or brightness of the subjects of interest or background, or changes in depth of field. These effects are typically used to bring visual attention to or from a subject of interest. The complete set of 3D pan & scan effects in our taxonomy includes every combination of camera move and image effect in Table 3.1 for a specific number of subjects of interest (e.g., no image effect is possible for zero subjects of interest). The most typical subject of interest used in these effects is a human face.

For scenes with no specific subject of interest, we observed two basic types of "establishing shots." These shots depict the entire scene without focusing attention on any specific part. In one type of establishing shot, the camera simply dollies across the scene in order to emphasize visual parallax. We will call this an *establishing dolly*. In the other type of establishing shot, the camera starts in close and dollies out to reveal the entire scene. We will call this an *establishing dolly-out*.

For scenes with a single subject of interest, two types of camera moves are commonly used. The first uses a depth dolly to slowly move the camera in toward the subject, or, alternatively to pull away from it. We will call this type of move a *dolly-in* or *dolly-out*. A

variant of this move involves also reducing the depth of field while focusing on the subject to draw the viewer's attention. Another variant, which can either be combined with a changing depth of field or used on its own, is an image effect in which either the subject of interest is slowly saturated or brightened, or its complement (the background) desaturated or dimmed. The other type of camera move sometimes used with a single subject of interest is a kind of special effect known as a *dolly zoom*. The camera is dollied back at the same time as the lens is zoomed in to give an intriguing, and somewhat unsettling, visual appearance. This particular camera move was made famous by Alfred Hitchcock in the film, *Vertigo*, and is sometimes known as a "Hitchcock zoom" or "Vertigo effect." Like the other single-subject camera moves, this move works equally well in either direction.

Finally, for scenes with two primary subjects of interest, the camera typically dollies from one subject to the other. We call this move, simply, a *dolly*. There are two variations of this move, both involving depth of field, when the objects are at substantially different depths. In the first, a low depth-of-field is used, and the focus is pulled from one subject to the other as the camera is simultaneously dollied. In the other, the depth of field itself is changed, with the depth of field either increasing to encompass the entire scene by the time the camera is dollied from one subject to the other, or else decreasing to focus in on the second subject alone by the time the camera arrives there. In general, any of the camera moves for scenes with n subjects of interest can also be applied to scenes with more than n. Thus, for example, scenes with two or more subjects are also amenable to any of the camera moves for scenes with just one.

3.4 Authoring

In this section, we describe how to generate 3D pan & scan effects, initially focusing on automatically generated effects that follow our taxonomy, and concluding with an interactive key-framing system.

The input to this step is a light field with depth information. We assume that the input light field is sparse, and that novel views can be synthesized by projecting and blending textured depth maps. Due to the sparseness of the input, however, novel renderings will typically exhibit holes. While small holes can often be inpainted, large holes are best avoided.

Computing a 3D pan & scan effect automatically from this input requires solving three problems. First, an effect appropriate for the imaged scene must be chosen from the taxonomy in Table 3.1. Second, a linear camera path must be computed that follows the intent of the effect and respects the limited sampling of the input. Third, any associated image effects must be applied.

3.4.1 Choosing the effect

Choosing an effect requires identifying the number of subjects of interest. In general, it is difficult, sometimes impossible, to guess what the user (or director) intends to be the subjects of interest in a scene. However, for our automatic system, a natural guess for a scene with people is to select their faces. We therefore run a face detector [123] on the centermost input view and count the number of faces. Then, one of the effects from the appropriate line in Table 3.1 is randomly chosen. The possible effects include image effects such as changing depth of field and focus pulls. Saturation and brightness changes, however, are left to the interactive authoring system, as they are less likely to be appropriate for an arbitrary scene.

3.4.2 Choosing a camera path

Each of the camera moves used in 3D pan & scan effects described in section 3.3 can be achieved by having the virtual camera follow a suitable path through camera parameter space. This parameter space includes the 3D camera location, the direction of the camera optical axis, and focal length. All of these parameters can vary over time. If we assume that all parameters are linearly interpolated between the two endpoints, the problem reduces to choosing the parameter values for the endpoints. The result is 6 degrees of freedom per endpoint — 3 for camera position, 2 for the optical axis (we ignore camera roll, uncommon in pan & scan effects), and 1 for focal length — and thus 12 degrees of freedom overall (two endpoints). A candidate for these 12 parameters can be evaluated in three ways.

1. The camera path should follow the particular 3D pan & scan convention.

2. The camera path should respect the limitations of the input. That is, viewpoints that

require significant numbers of rays not sampled in the input should be avoided (modulo the ability to successfully fill small holes).

3. The camera path should be chosen to clearly exhibit parallax in the scene (as we show in our user study in Section 3.8, users prefer effects that are clearly 3D). Unfortunately, finding a global solution that best meets all 3 goals across 12 degrees of freedom is computationally intractable. The space is not necessarily differentiable and thus unlikely to yield readily to continuous optimization, and a complete sampling strategy would be costly, as validating each path during optimization would amount to rendering all the viewpoints along it.

We therefore make several simplifying assumptions. First, we assume that the second and third goals above can be evaluated by only examining the renderings of the two endpoints of the camera path. This simplification assumes that the measures for achieving those goals are generally greater at the endpoints than at points between them; e.g., a viewpoint along the line will not have more holes than the two endpoints, or at least not substantially more. While this assumption is not strictly true, in our experience, samplings of the space of viewpoints suggest that it often is. Second, we assume that the camera focal length and optical axis are entirely defined by the specific pan & scan effect, the camera location, and the linear interpolation parameter. For example, a dolly effect starts by pointing at the first subject of interest, ends by pointing at the second subject of interest, and interpolates linearly along the way. The focal length is set to keep the subjects of interest a certain size. As a result of these assumptions, the problem of choosing a camera path reduces to finding two 3D camera locations, such that the renderings of both viewpoints contain few holes and exhibit significant parallax relative to each other.

Valid viewpoint sampling

We first discuss how to identify viewpoints from which we can successfully render the input photographs processed by our system (measure #2). The set of all viewpoints can be described by a hypervolume parameterized by the camera parameters described above. We constrain this hypervolume to the finite region of valid viewpoints, where a valid viewpoint is defined as a viewpoint from which the rendered scene is complete or contains holes that

are likely to be inpainted easily. We take advantage of the interactive renderer described in Section 3.6 to quickly determine valid viewpoints. Given the scene rendered from viewpoint V, we evaluate the success of the rendering using the following metric H:

$$H(V) = \frac{\sum_{p \in V} [d(p)]^k}{w \times h}$$

where w and h are the width and height of the rendering, $d(p)$ is the minimum distance from pixel p to a valid pixel, i.e., the boundary of the hole ($d(p)$ is set to 0 if p is not inside a hole), and k is a constant. Larger values of k penalize larger holes (which are harder to inpaint) over many smaller holes; we found $k = 3$ to be a be good value. We use a distance transform [38] to compute $d(p)$ quickly. We consider a viewpoint as valid if $H(V) < 2$.

The next step is to explore the hypervolume and define the 3D region of viewpoints that satisfy this viewpoint validity constraint. We assume that the coordinate system is aligned so that the input images mostly samples rays from viewpoints spread roughly across a plane (call it the x-y plane) looking in a direction (call it the z-axis) roughly perpendicular to that plane. Section 3.6 and Figure 3.5a describe this plane and associated view mesh in greater detail. We also assume that the centermost viewpoint is roughly at the origin of the coordinate system. For some effects, such as dolly-out, the camera motion is constrained to travel down the z-axis of this coordinate system. Other effects have more freedom; for these, we search for two regions of valid viewpoints, one for the starting and one for the ending camera viewpoints (since these will have different constraints on pointing direction and focal length).

To explore the range of valid viewpoints we uniformly sample along x and y at $z = 0$, and then search along z until $H(V)$ exceeds the threshold. The uniform sampling along x and y is done at a resolution of a 12×12 grid across a region that is 2 times the size of the bounding box of the viewpoints in the input light field (we have explored wider and denser samplings, and found these settings to be a good trade-off between efficiency and quality). We search in each direction along z until $H(V)$ exceeds the threshold. We search using an adaptive step size, increasing or decreasing the step size by a factor of two depending on whether the $H(V)$ threshold is met or exceeded, respectively. Figure 3.1 demonstrates a real example of such a grid (with a denser 100×100 sampling for visualization purposes), and

the results of searching forwards in z.

Maximizing parallax

Next, we need an approach to measuring the parallax induced between two viewpoints. There are a number of possibilities for measuring parallax. We found that the most perceptually noticeable areas of parallax are those that are visible in one viewpoint and occluded in the other. We therefore project the starting viewpoint into the ending viewpoint and vice versa, and sum up the area of holes; this sum is our measure of parallax.

We assume that the starting and ending viewpoints will both be extremal in z; we thus have 144 candidates for both the starting and ending viewpoints (from sampling the 12×12 grid twice). We choose the highest scoring pair according to our measure of parallax. Since this measure requires projecting the starting viewpoint into the ending viewpoint and vice-versa, choosing the optimal pair would require $2 \times 144 \times 144$ projections, which is time-consuming. However, there is a strong correlation between the length of the camera path and the amount of parallax. We selected a dozen examples and performed the full set of projections. We found that the best pair of viewpoints by our parallax metric was always among the top 12 pairs when sorted by the length of the camera path. We therefore increase the speed by only performing the projections on the 12 longest camera paths, and choosing the one with the most parallax.

Constraints on the path

Each camera move in our taxonomy imposes specific constraints on the camera focal length, optical axis, and in some cases, camera motion. These constraints are designed to mimic the effect's typical appearance as we observed them in documentary films. The constraints make use of information that we assume is contained in the input light field, such as the 3D centroid of the scene (straightforward to compute given that the input light field contains depth information), and the average focal length f of the capture device (in our case a standard camera). We now address each camera move, beginning with the case where there is no specific subject of interest.

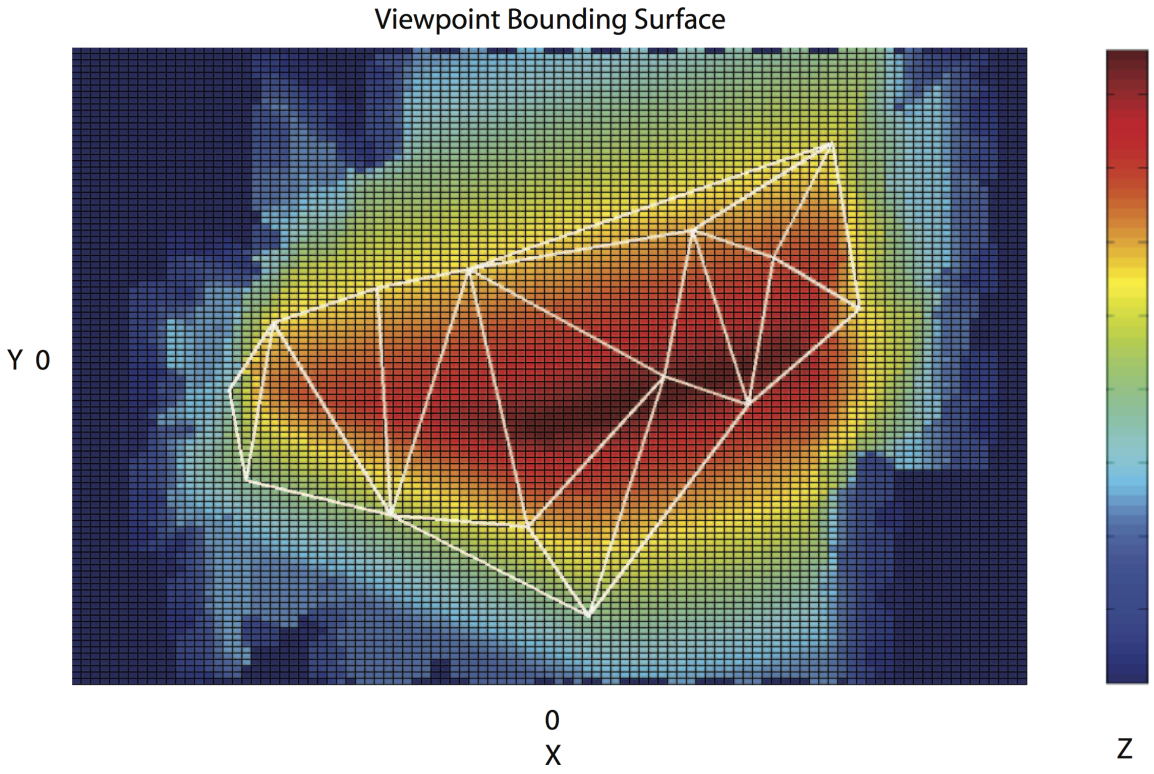


Figure 3.1: A 100×100 *grid of valid viewpoints color coded by z-value. A mesh connecting the original viewpoints in the input light field is shown in white. Note viewpoints in the central area are closer to the original sampled viewpoints; thus they can move much closer to the scene (larger z) than peripheral viewpoints.*

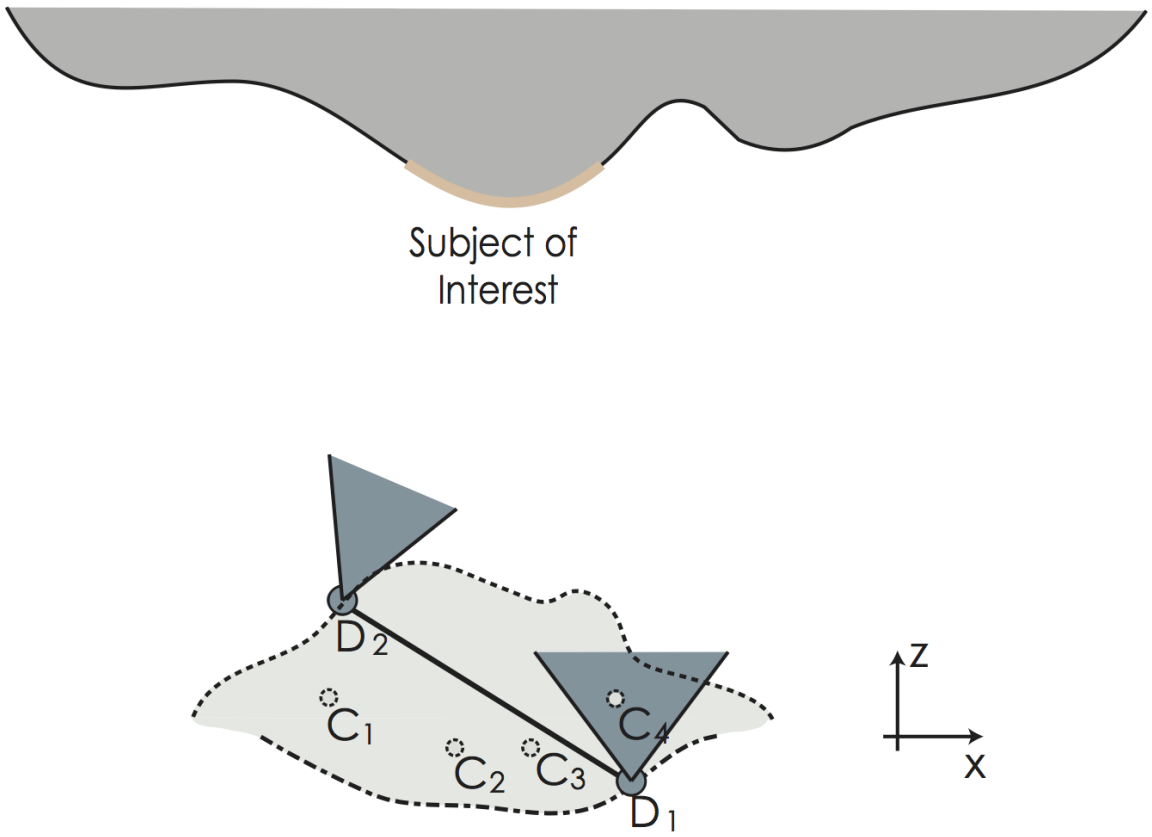


Figure 3.2: For one subject of interest, our algorithm locates the linear camera path (D_1D_2) for dolly-in/out within the valid hypervolume. D_1 and D_2 are on the front and back edges of the valid hypervolume with respect to the z axis. Here C_x denotes the position of a viewpoint sampled in the input. Note how D_2 adjusts its orientation and focal length to keep the subject of interest centered.

Establishing dolly. The camera always points at the scene centroid, and the focal length is set to f for both ends of the camera path.

Establishing dolly-out. The camera moves along the z axis, always pointing at the scene centroid. The starting focal length is set to $1.5f$, and the ending focal length to f. In this case, no sampling grid across x, y is needed, (though search in the $+z$ direction and another in the $-z$ direction must be performed).

Next we consider the cases containing one or two subjects of interest. To test for the number of subjects of interest, we run the face detector, truncate to the two largest faces if more than two are found, and use the detected rectangles to construct a geometric proxy

for the faces. We compute the median depth within each rectangle, and construct a 3D quadrilateral at that depth. Here we discuss 3D pan & scan effects for a single region of interest.

Dolly-in/out. In this case (Figure 3.2), the starting camera points at the lower half of the rectangle containing the subject of interest (so that the face is slightly above center), and the ending camera points at the scene centroid (or vice-versa). The focal length starts at $1.5f$ so that the face is zoomed-in, and ends with f (or vice-versa).

Dolly zoom. Here the algorithm follows the same procedure as when executing an "establishing dolly-out" with one exception: the focal length is adjusted during the animation to force the region of interest to have the same size as seen in the starting viewpoint.

Finally, we consider two regions of interest.

Dolly. The camera starts by pointing at the first subject of interest, and ends by pointing at the second. The focal length starts at $1.5f$, and ends at the same focal length times the ratio of the size of the subjects of interest (so that the final subject of interest ends at the same size as the first).

3.4.3 Image effects

For one or more subjects of interest our solution may choose to add depth-of-field and/or focus pull effects. Depth-of-field adds another degree of freedom per camera endpoint, namely the aperture diameter. After the two camera endpoints are chosen, a maximum aperture diameter must be chosen so that it does not change the validity of a viewpoint; we therefore search for this maximum. We assume that an aperture diameter can be evaluated by the maximum $H(V)$ of the four corners of the bounding box of the aperture. Starting with an initial maximum aperture diameter, we search adaptively for bigger apertures until one of the four corner viewpoints becomes invalid.

To add a changing depth of field to a dolly or dolly in/out, the aperture is linearly interpolated from a pinhole to the maximum aperture. For a focus pull, the aperture is kept at the maximum, and the in-focus depth is simply transitioned from the depth of one subject to the other.

3.4.4 Interactive camera control

We also allow a user to design camera paths that are outside of the taxonomy giving them the freedom to design novel camera paths, or to chain together and modify automatically generated paths. The user is free to navigate, using the interactive renderer, to desired viewpoints and adjust camera settings, as well as color effects (like saturation) and enter them as keyframes along a cubic spline path. Alternatively, the user can author paths by simply drawing one or more regions of interest, and choosing a camera move from the taxonomy; the system will then compute a path using the method for automatically detected regions of interest. Finally, the user can add depth-dependent brightness and saturation effects tied to a manually specified region of interest.

3.4.5 View morphing

For the special case where only two views are provided as input to the system, the method of Zheng et al. [129] computes stereo correspondences using a soft epipolar constraint. The soft constraint allows for some scene motion. The automatic system simply interpolates linearly between correspondences, i.e., morphs between the views, regardless of scene content.

3.5 Constructing light fields with depth

The input light fields used to create our results are constructed using existing computer vision techniques from a few photographs of a scene taken from slightly different viewpoints, either by multiple shots from a standard camera, or with a single shot from a multi-viewpoint camera. First, a structure-from-motion system [114] is used to recover the relative location and orientation of each photograph, as well as the camera focal lengths.

Next, we compute a depth map for each viewpoint using a multi-view stereo algorithm [129]. Like some other top-performing stereo algorithms [70, 105], this method performs an over-segmentation of the input photographs and then determines a depth per segment, rather than a depth per pixel. In our representation, within a single view, the constant-z segments have spatially varying colors and can overlap each other, at most two segments covering each pixel, with blending weights that sum to one at each pixel. We

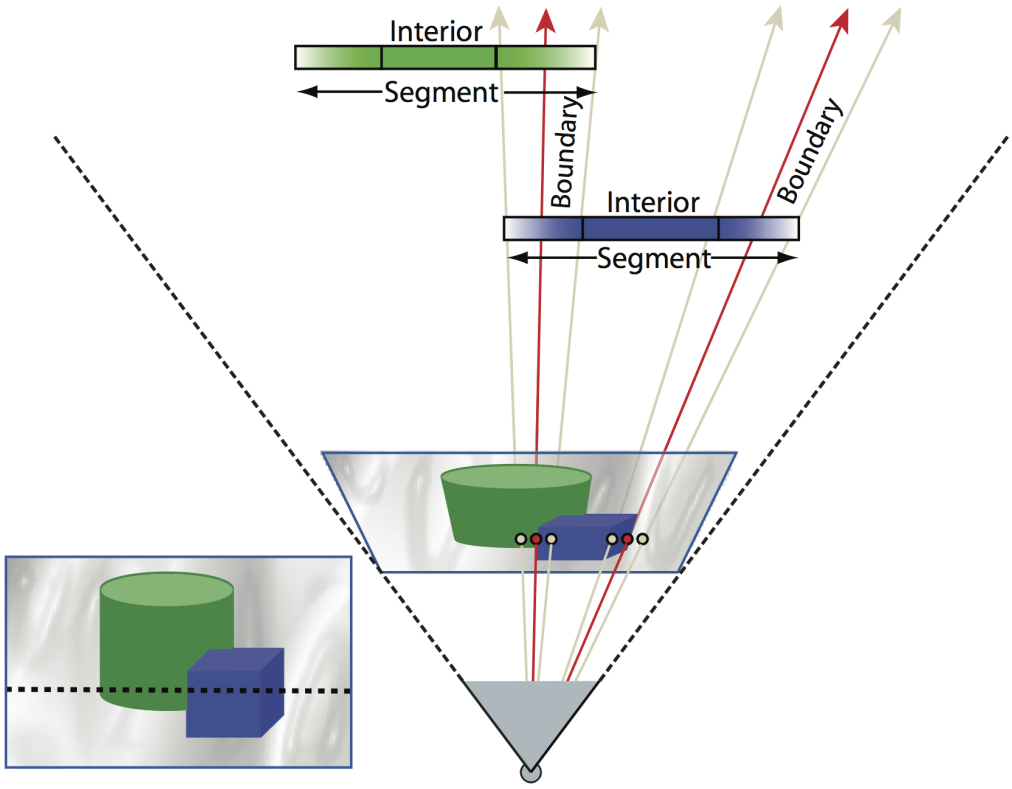


Figure 3.3: Segments are partitioned into two regions, opaque interior regions which do not overlap, and semi-transparent overlapping boundary regions which are decomposed into a two-color model representing color mixing between adjacent segments in the scene.

can express the color and weight variations within a segment using RGBA textures. By rendering these segmentation-based depth maps into novel viewpoints, we can effectively "hallucinate" the light field in the neighborhood of the input views.

This two-component segment representation is used to compensate for color mixing between adjacent segments due to foreground and background object matting. The first component represents the interior region of the segment where there is no color mixing. The second component, the boundary region, is the segment's contribution to the mixed color region between adjacent segments. The interior regions of adjacent segments do not overlap. However, the boundary regions do since they represent estimated the foreground and background colors. Compositing the segments for one input photograph onto its own viewpoint regenerates the original photograph.

3.6 Rendering algorithms

We have developed two rendering algorithms to display novel views from the textured segments with depth. The first rendering algorithm is implemented as a real-time renderer leveraging the GPU. The second algorithm is implemented as an off-line renderer and produces higher quality results; it is used to render the final sequences. Both renderers take the same basic approach; we therefore first describe the general rendering algorithm, and then we describe the specifics of the interactive renderer implementation including GPU acceleration, and finally we describe the differences in the off-line rendering algorithm.

To render the scene from a novel view, we project a ray from the origin of the novel view through each pixel in the image plane. If the ray intersects any segments, we combine their color and depth values and assign these values to the pixel. We calculate each segment's contribution in three steps. First we choose which segments should contribute to the the pixel value. Second, we compute a *blending weight* for each contributing segment color value. Finally we employ a *soft z-buffer* to resolve depth inconsistencies and combine the color values. When choosing which segments should contribute to a pixel value, we only consider those segments belonging to the three viewpoints adjacent to the ray being rendered.

In order to quickly determine the adjacent viewpoints to a ray and to calculate blending weights, we construct a *view mesh*. A *view mesh* is a triangle mesh with vertices at the centers

of projection (COPs) of the input views. The adjacent viewpoints are found by intersecting the *view mesh* with forward and backward projected ray. The viewpoints corresponding to vertices of the intersected triangle are the adjacent viewpoints, and therefore the ones we wish to use.

To construct the mesh, we fit a plane to the COPs, project them to that plane, and perform a Delaunay triangulation [109]; the connectivity of this triangulation is used, but the original COP locations are retained. In the case that the projected ray does not intersect the *view mesh*, we still wish to select the two viewpoints closest to the origin of the novel view. To do so, we extend the *view mesh* by duplicating the vertices on the mesh boundary. These duplicate vertices are positioned radially outward from the mesh center. We incorporate the vertices into the mesh and re-triangulate. To calculate the position of the duplicate vertices, we compute the centroid of the mesh vertices and form a vector from that centroid to a vertex on the boundary of the mesh. We multiply this vector by a constant value and add it to the centroid. A multiplier of 4.0 effectively gives us a sufficient radial extent. The resulting vertex is inserted into the *view mesh*. This process is repeated for all vertices on the boundary of the *view mesh* as shown in Figure 3.5.

To render a single ray from a novel viewpoint, we intersect the ray with the view mesh. The intersected triangle indicates which three views will contribute to the ray. We use the barycentric coordinates of the intersection as the *blending weights* associated with the corresponding views. This blending weight is similar to that described in [99] and [17].

The ray is then intersected against the segment geometry in each of the three views. Because each view has independent reconstructions of scene depths, the same surface point may be reconstructed with somewhat different depths. This depth incongruity may cause popping artifacts during animation, so we combine the depths and colors along the ray with a *soft z-buffer* to resolve depth inconsistencies [99]. The intuition behind a soft z-buffer is that some noise will inevitably arise during computation of segment depths. Therefore, segments that represent the same scene geometry will not have exactly the same depth. A soft z-buffer allows us to consistently resolve conflicting depth information by sampling all of the segments that may contribute to a pixel, and estimating the most likely RGBA and z values for the pixel. The result of the *soft z-buffer* is a z-weight for each contributing

segment. The final pixel value is the normalized sum of the contributing segments multiplied by their z-weight and *blending weight*.

For views that are too far from the input views, some rays will intersect no segments; this is most prevalent around depth discontinuities. The resulting holes in the rendering are filled by inpainting.

3.6.1 Interactive renderer

To render the scene from a novel view at interactive frame rates (at least 30 fps), I developed a GPU-based algorithm. I render the scene in four steps. First, we choose which views should contribute to a pixel value and calculate the per-pixel blending weight. Second, we render all of the segments to three offscreen buffers. Third, I employ a soft z-buffer to resolve depth inconsistencies between the three offscreen buffers and combine their color values. Finally, I fill holes using a *reverse soft z-buffer* and local area sampling.

Rendering the extended view mesh

To choose which segments contribute to the pixel value and to calculate the blending weights, I render the extended *view mesh* from the novel viewpoint to an offscreen buffer, encoding the barycentric coordinates (blending weights) and viewpoint ID into the RGB channels. The viewpoint's ID are encoded in the color channels using a consistent ordering, thus creating a 3-element list storing the ID's of the three viewpoints contributing to each pixel. For simplicity, the vertex order of the *view mesh* triangles are used to order the list. The specific ordering mechanism does not matter, but the fact that the list is ordered is important for rendering. Note that I have not yet chosen which segments should be rendered, however I now have enough information at each pixel to choose segments during the next rendering step, and to direct the segment to a specific offscreen buffer.

With respect to rendering the extended *view mesh*, there are two special cases that should be highlighted. First, if the novel view lies in front of the *view mesh*, the origin of the novel view lies between the *view mesh* and the scene. In such cases this projection step requires backwards projection (i.e., projecting geometry that is behind the viewer through

the center of projection). Second, the projection of the *view mesh* nears a singularity as the novel view moves close to the *view mesh*. Therefore, if the novel view is determined to lie within some small distance from the *view mesh*, the nearest point on the mesh is found. The barycentric coordinates, and therefore the *blending weights*, of this point are taken to be constant across the novel view.

Rendering segments

Next the image segments will be rendered and combined. It is worth revisiting the segment representation for this overview. The segment is represented by a 3D rectangle determined by the bounding box of the segment, and an associated texture. One could create an individual texture map for every segment, setting the alpha channel to zero for areas outside of the segment boundaries. This representation makes rendering very simple. However, this process creates thousands of small textures and is very inefficient in practice. [1] I avoid creating separate textures maps for each segment by using three texture maps per viewpoint. The segments created using the process described in Section 3.5 can be compactly represented by two RGBA color texture maps and one *Segment ID* texture map per viewpoint. The first RGBA color texture map consists of all of the interior regions (which do not overlap), and one of the overlapping boundary region from adjacent segments. The second texture map consists of the remaining boundary regions. Finally I create a third 2-channel integer *Segment ID* texture map containing segment ID values indicating to which segment a pixel belongs. I create a quadrilateral for each segment, using the segment's bounding box to set the quadrilateral's texture coordinates and vertex locations. In addition, each quadrilateral is associated with the segment's ID and viewpoint. While this representation is quite efficient, there is not an explicit alpha value that defines the boundary of the segment. In order to properly render a segment, one must use a pixel shader to discard fragments laying outside the segment boundary. When rendering the quadrilateral, I encode the segment ID into a vertex attribute (e.g. color or normal data). The shader compares this value to the segment ID encoded in the texture to determine if a fragment should be rendered as part of the

[1] We tried this storage mechanism originally, and the performance was roughly ten times slower than what we describe in this section.

segment.

Using this GPU based texture representation has two benefits both of which increase rendering speed. First, it reduces the number of texture swaps from thousands per viewpoint to three. Secondly, by removing the texture swap from the rendering inner loop, I am able to take advantage of vertex arrays or vertex buffer objects and utilize the GPU more efficiently.

A pixel in a rendered scene is ultimately a sum of segments originating from three different viewpoints. Rather than determining which segments should be rendered on the CPU, we use the GPU to direct the output of a rendered segment to an offscreen buffer for further blending, or discard it, entirely based on its viewpoint. When rendering a segment, I encode its viewpoint ID into a vertex attribute, in the same manner as we encoded the segment ID. I use the offscreen buffer generated by rendering the *view mesh* as a texture map. The pixel shader chooses to which of the three buffers a segment should contribute, if any, by determining if the viewpoint's ID is encoded in the *view mesh* texture map at that pixel location. If the viewpoint's ID is encoded in the *view mesh* texture map, we store the segment's color and z-value, otherwise they are discarded. I choose in which buffer to store the values by using the ordering of the viewpoint IDs encoded into the *view mesh* texture map. Recall that the 3 contributing viewpoint IDs are encoded in an ordered list. This is order arbitrary (the vertex order of the *view mesh* triangles), however using this order to determine the destination buffer guarantees that only a single viewpoint will write to that buffer at that pixel location. Specifically, if a segment's viewpoint matches the first viewpoint encoded in the list, its values are written to the first buffer, likewise if a segment's viewpoint matches the second viewpoint encoded in the list, its values are written to the second buffer, and so on for the third. Otherwise, the values are discarded. As a result, each pixel in each of the three buffers accumulates RGBA values for a single viewpoint. The resulting three buffers may be chimeras, each composed from different contributing viewpoints. These three buffers are combined to form the final image in the next rendering step.

The renderer performs a front-to-back depth sort of all segments for each viewpoint, using the z-axis of the centermost camera in the *view mesh* as the global z-axis. Each segment from every viewpoint is rendered to an offscreen buffer (initialized to black background

and zero alpha) using the pixel shaders described above. In order to maintain proper color blending and z-buffer, The segments are rendered from front to back, using a saturated blending mode. The pixel shader calculates the pre-multiplied pixel values and alpha.

Hardware texture limitations At this point, it is worthwhile to note that the three separate offscreen buffers described in the prior step are a simplification. Each offscreen buffer consist of two buffers, an RGBA buffer and a z buffer, resulting in six buffers in addition to the *view mesh* buffer. Some hardware has a limit of four texture buffers that can accessed via a pixel shader. To overcome this limitation, it possible to create a single offscreen buffer three times as wide as the rendering viewport to represent the three offscreen buffers. A geometry shader can be used to retain a single pass rendering loop. The function of the geometry shader is to simply duplicate the quadrilaterals, scale and translate them to the three regions of the offscreen buffer.

Soft z-buffer

I employ a *soft z-buffer* to combine the three rendering buffers. In this rendering pass, the *view mesh* texture map is used to determine the blending weights for each pixel location of the three buffers buffers generated in the previous step. For each corresponding pixel in the three buffers, we compute a soft weight w_z by comparing each pixel's z-value with the z-value of pixel closest to the origin of the novel view. This distance Δz, where $\Delta z = z_{\text{closest}} - z$, is used to compute w_z in the following equation:

$$w_z(\Delta z) = \begin{cases} 1 & \text{if } \Delta z \leq \gamma \\ \frac{1}{2}(1 + \cos\left(\frac{\pi(\Delta z - \gamma)}{\rho - 2\gamma}\right)) & \text{if } \gamma < \Delta z \leq \rho - \gamma \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where ρ is the depth range (max$-$min) of the entire scene, and γ is set to $\rho/10$.

The set of w_z's are normalized to sum to one. The final pixel depth z is then given the sum of the z-values weighted by the w_z's. Each blending weight stored in the viewmesh texture is multiplied by its corresponding w_z and the alpha value of the buffer. These new weights are normalized, and the final pixel value is computed by summing scaling each pixel

by the normalized weight.

Hole filling

Holes occur when, due to parallax, a nearby segment separates from a more distant segment. I fill small holes of less than 6 pixels in diameter during the final *soft z-buffer* pass. Since holes occur due to disocclusion, given two neighboring pixels, we prefer to use the more distant one to fill the gap. To do so, I compute a weighted sum of pixel colors and z-values of the pixels in a 7×7 neighborhood. The weights are taken from those used in the soft z-buffer calculation described above except "in reverse." In particular, more distant z-values are given higher weights by setting the z-values to $1 - z$.

Depth-localized effects

Image based effects are often seen in documentary films. These effects are typically used to bring visual attention to or from a subject of interest by selectively brightening and/or desaturating a region specified either automatically or by the user. I extend these effects to 3D by localizing them to a specific depth. Depth-localized effects are image processing effects constrained to a 3D bounding region in x, y and z. These effects are potentially any image + depth based operations, including saturation, color shifting, brightening, or re-lighting. I model the effect after the layer paradigm commonly used in image and video processing tools such as Adobe Photoshop or Adobe After Effects. Each effect is considered to be an independent layer composited over the rendered image. If the user applies more than one effect, it's effect layer is composited in a user specified order.

Each effect is implemented as a pixel shader with input parameters that vary at run-time. The parameters include the bounding 3D region, a modulation parameter, and three texture maps consisting of the rendered scene, the z-buffer, and the result of the prior effect. Each effect may have additional parameters to control its application, for example a saturation effect may have an amplitude parameter controlling the amount of saturation or de-saturation the effect should apply. In order to provide more user control, the modulation parameter controls the compositing of the effect based on the distance from the bounding

region. The effect is modulated with a Gaussian falloff from the user specified 3D bounding region. This allows the effect to smoothly falloff in a scene with a noisy depth map. Finally, each user editable effect parameter can be specified with keyframes, in this way the effect can animated along with the camera path.

Depth of field

Efficient, approximate depth-of-field rendering is accomplished using a variation on existing methods [32, 68, 50]. For each pixel, we calculate a circle of confusion based on a user defined aperture size, and blur the result of our rendered scene accordingly. The blurring operation is performed efficiently by loading the scene into a MIPMAP and indexing into it based on the blur kernel radius. To improve visual quality, we index into a higher resolution level of the MIPMAP than strictly needed, and then filter with a Gaussian filter of suitable size to achieve the correct amount of blur. Note that when focusing on the background in a scene, this approach will not cause foreground pixels to be smeared over the background; i.e., the blurry foreground will have a sharp silhouette.

To avoid such sharp silhouettes, when processing a pixel at or behind the focus plane, the pixel shader blends the pixel with a blurred version of the image at that pixel. The blur kernel size is based on the average z-value of nearby foreground pixels. The blending weight given to this blurred version of the image is given by the fraction of the neighboring pixels determined to be foreground. The size of the neighborhood is determined by the circle of confusion computed from the user specified aperture and focal depth.

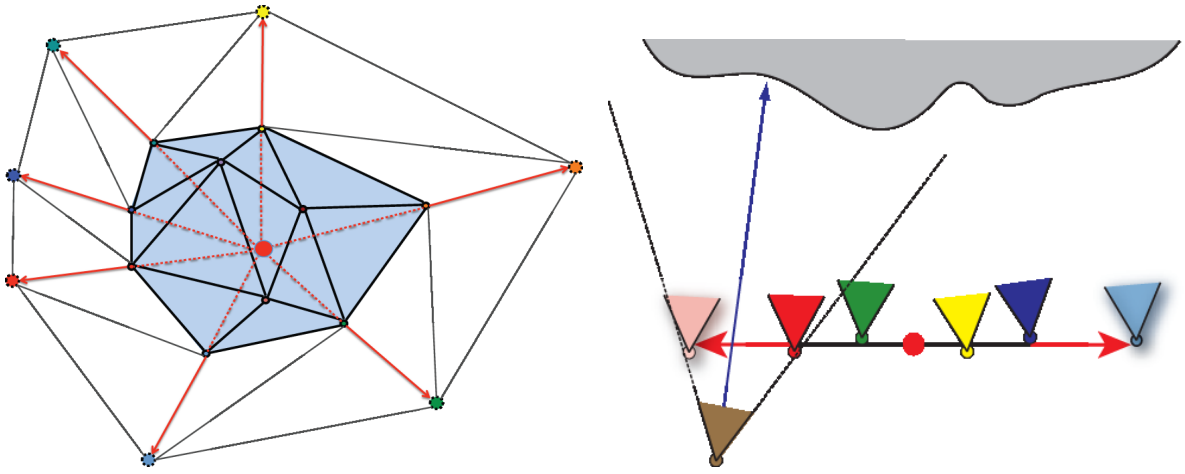
GPU based rendering summary

This fast GPU based algorithm for rendering high quality images at high frame rates. We can render scenes at 30-45 frames-per-second on an NVIDIA 8800 series graphics card, whereas an implementation using one texture per segment achieved only 7.5 frames-per-second and did not calculate the blending weights on a per pixel basis, use a *soft z-buffer* or hole filling. The algorithm enables an interactive user interface allowing users to preview a camera move in real-time, and interactively edit camera paths and depth-localized image effects.

(a) (b)

(c) (d)

Figure 3.4: Side by side comparison of images produced by a GPU based rendering algorithm and a software only algorithm. The GPU based algorithm renders this scene at 30 frames per second, 450 times faster than the software only version. The top row compares a frame taken from the middle of an automatically computed camera path. The image **(a)** *is the GPU rendered image,* **(b)** *is rendered in software only. The bottom row is the first frame of the computed camera path. This frame is designed to have holes and artifacts that the software only algorithm can correct. The image* **(c)** *is the GPU rendered image, notice the artifacts on the right hand side of the scene. The image* **(d)** *is rendered in software with high quality hole filling.*



(a) (b)

Figure 3.5: **(a)** *The view mesh is extended by creating copies of boundary vertices and placing them radially outward (red arrow) from the centroid (red dot) of all of the mesh vertices. These new vertices are then re-triangulated and added to the view mesh to form an extended view mesh.* **(b)** *The dotted blue arrow shows a ray projected through the image plane. The* blending weights *at this pixel correspond to the barycentric coordinates of the intersection of the ray and triangle defined by the viewpoints in the view mesh.*

In addition, this algorithm enables a camera path optimization method (described in the Section 3.4.2) that would not be feasible without both the high quality and high frame rate. The fast GPU-accelerated rendering algorithm has contributed several novel features. First, it has contributed a method to interleave pixel colors and camera source IDs to multiplex rendering and guarantee optimal use of video memory. The method is used in this context to blend three weighted values, in principle it could be used for any number of weights. This indexing method enabled a second contribution, the first GPU-accelerated version of the soft-z buffer technique, which minimizes temporal artifacts. Finally, a GPU-accelerated inverse soft-z approach to fill small holes and gaps in the rendered output.

3.6.2 Off-line rendering

The higher quality off-line rendering algorithm differs from the interactive renderer in three main ways. First, we extend the *soft z-buffer* described above to increase the accuracy of our

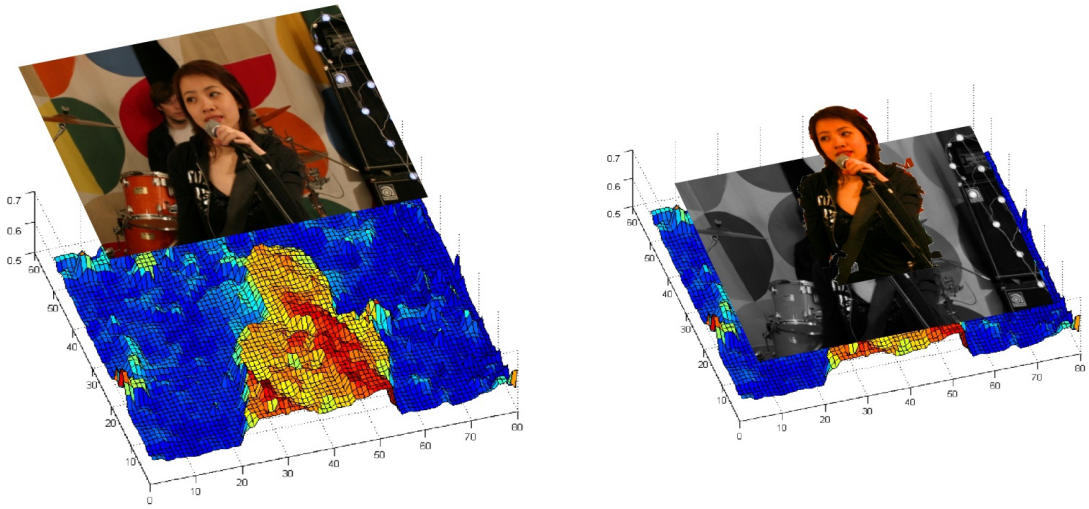
(a) (b)

Figure 3.6: Comparison of an image **(a)** *rendered using the GPU soft-z approach to an image* **(b)** *rendered without a soft-z buffer. Notice the front edge of the lower shelf, in image* **(a)** *the shelf is solid, however in image* **(b)** *there are visible artifacts. The artifacts are the result of blending background pixels from different viewpoints with those of the foreground. Image* **(b)** *is produced by only considering the* blending weights.

pixel value estimate. Second, the renderer uses a texture synthesis approach to fill any holes and cracks that might appear in a novel view due to sparse data generated from the input photographs. Finally, depth of field effects are rendered with increased quality by simulating a camera aperture.

Soft z-buffer The soft z-buffer calculation is very similar to the process described in the real-time renderer. However, rather than using a traditional hard z-buffering within each viewpoint followed by a soft z-buffer across viewpoints, all segments from all contributing viewpoints are combined in a uniform manner.

Hole filling To fill holes the offline renderer uses an approach similar to the in-painting algorithm of Criminisi et al. [27] with two modifications. First, to accelerate computation,



(a) (b)

Figure 3.7: Pixel depth is used as a constraint specifying the boundary of an effect. The image **(a)** *shows the rendered scene and the associated depth map, the image* **(b)** *shows two layers, one layer corresponds to a desaturation effect applied to whole image. The second layer corresponds to a hyper-saturation effect on a region bounded by depths in the red-orange range.*

we search for matching (5×5) neighborhoods within a restricted window (100×100) around the target pixel. The second, more significant, modification is based on the observation that nearly all large holes occur along depth discontinuities, because some region of background geometry was always occluded in the input photographs. In this case, the hole should be filled from background (far) regions rather than foreground (near) regions. We thus separate the depths of the pixels along the boundary into two clusters, and use these two clusters to classify pixels, as needed, as foreground or background. We fill the hole with Criminisi's propagation order, using modified neighborhoods and neighborhood distance metrics. When copying in pixel color, we also inpaint its z by weighted blending from known neighbouring pixels, again favoring the back layer. The inpainted z assists in region selection for color manipulation effects. Note that Moreno-Noguer et al. [85] also explored depth-sensitive inpainting, though their application has lower quality requirements since they use the inpainted regions for rendering defocused regions rather than novel viewpoints.

Depth of field Our rendering algorithm now provides a way to reconstruct any view within

the viewing volume. In addition to changing viewpoint, we can synthetically focus the image to simulate camera depth of field. To do so, we apply an approach similar to what has been done in *synthetic-aperture photography* [77, 63]. We jitter viewpoints around the center of a synthetic aperture and reconstruct an image for each viewpoint. We then project all the images onto a given in-focus plane and average the result.

3.7 Results

We have tested our overall approach (including the construction of the light field from a few photographs) on 208 datasets capturing a variety of scenes. About half of the datasets (103 out of 208) produced successful results that were comparable to the results shown in this paper. The failures were largely due to subject motion and errors in structure-from-motion and multi-view stereo.

A subset (20) of the successful results are included in the accompanying video. Only 2 examples were interactively authored. The rest were generated entirely automatically through our pipeline. The best way to experience the cinematic effects produced by our system is in animated form, as shown in the supplemental video. The number of input images ranges from 8 to 15 and are typically captured with just 3-4 inches of space between the viewpoints. Most datasets are captured at resolution 1200×800. A 3GHz PC with 4GB memory and an NVIDIA 8800 class GPU interactively renders scenes at 30-45 frames-per-second. Automatically generating a 3D pan & scan effect automatically takes about 3 hours, including 10-15 minutes for structure-from-motion, 100-120 minutes for multi-view stereo, 2-5 minutes for computing the effect, and 25-35 minutes for the off-line rendering.

While some users may be willing to capture multiple viewpoints, a single-shot solution is certainly more appealing and would address the problem of subject motion between viewpoints. To that end, we include two examples from multi-viewpoint cameras: an integral-lens camera [45], and the $50 Pop9 film camera.[1] The results are not perfect, notably for the prototype integral-lens camera, due to chromatic aberrations in the prisms used to capture 20 images at once. However, these results demonstrate that the small baselines of these

[1] `http://shop.lomography.com/pop9/`

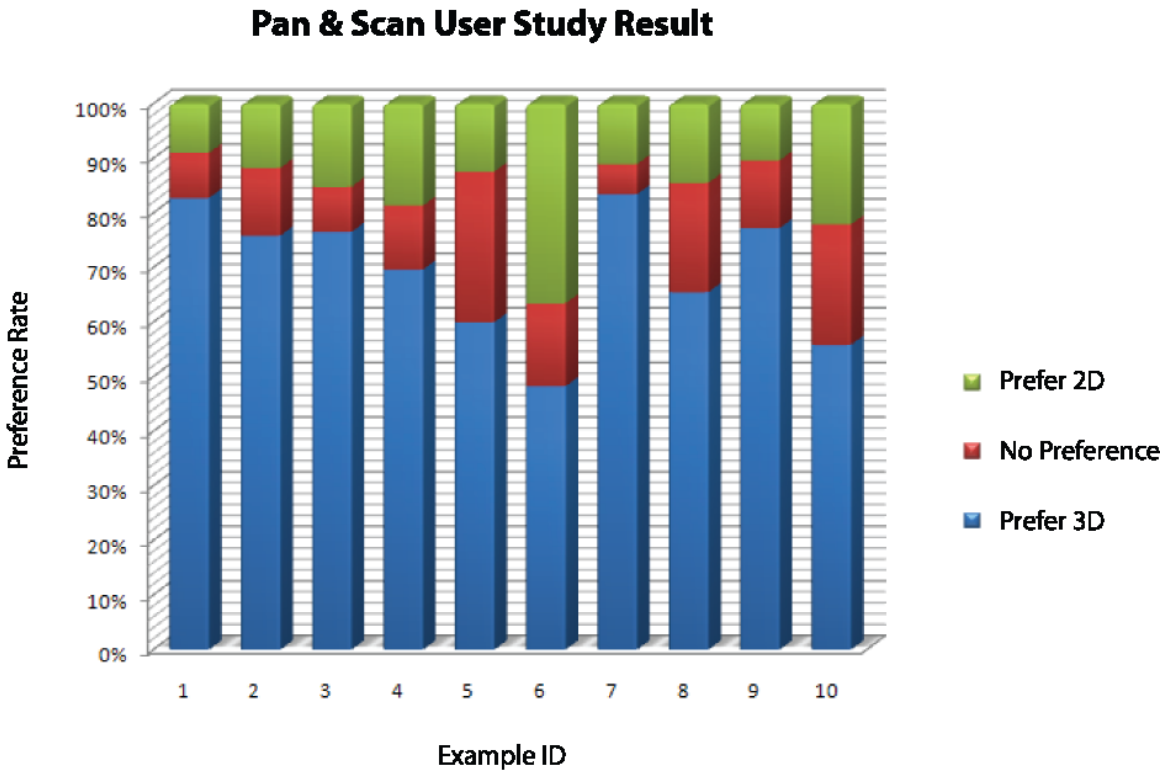


Figure 3.8: The preference rate of the 10 examples in the user study. Note that fewer than 65% of the subjects preferred the 3D version of examples 5, 8 and 10; these examples contained the most subtle parallax or motion, and thus support the notion that maximizing parallax is important.

cameras can still yield subtle but noticeable parallax. In addition, we include two results using two input views with small scene motion (the guitarist and the smiling girl), for which our automatic system generates morphs from one view to the other.

Overall, the sequences are smooth and convey mild to strong motion parallax. However, occasional artifacts can be seen, often near depth boundaries. These artifacts come from depth mis-estimates; as multi-view stereo algorithms improve these types of errors should be reduced.

3.8 User study

In order to assess the perceptual quality of the 3D pan & scan effects, we conducted a user study study with 145 subjects, the majority of whom were in the IT industry but not in

the graphics industry. Each subject was asked to watch 10 examples, with each example shown in two versions: a 2D pan & scan, and a 3D pan & scan. All effects were rendered at 480×320 pixels at 30 frames per second and were 3-5 seconds long. For each example, subjects had to indicate whether they noticed any difference between the two versions after watching them as many times as they liked. They also had to select which version they preferred, and optionally provide reasons for their choices.

The results show that there was a significant perceptual difference between 3D pan & scan and 2D pan & scan effects. Overall, the two versions of each example were judged to be different 94% of the time, and at least 80% of subjects noticed at least some difference between the two versions on every example.

The results also show that the majority of participants prefer the 3D versions to the 2D ones. Summing over all examples, 70% of subjects, on average, preferred 3D pan & scan, 16% preferred 2D pan & scan, and 14% had no preference. As shown in Figure 3.8, the preference rate exhibits correlations with the amount of parallax contained in each example. Note also that roughly half the people preferred the 2D to the 3D "dolly-zoom" example (#6 in Figure 3.8); in written comments they noted that the 3D effect was too dramatic and uncomfortable, which is the intended purpose of the effect. The users' written comments clearly show that apparent parallax was the dominant reason (69% of all reasons collected) behind the preference for the 3D results. Taken together, the results of the user study clearly indicate that the cinematic effects we created offer a more compelling depiction than a simple 2D pan & scan.

3.9 Conclusion

Recent advances in computational photography have dramatically increased the amount of information that we can capture from a scene. Until now, techniques that capture depth along with an image have been used primarily for digital re-focusing, on the assumption that small parallax changes are uninteresting. On the contrary, we believe that subtle parallax can lead to a richer, more informative visual experience of a scene that feels fundamentally different than a still photograph or 2D pan & scan. As multi-view camera designs and computer vision techniques continue to improve, we see an opportunity for parallax photography

to become a widely used approach to capturing the moments of our lives.

Chapter 4

IMAGE-BASED REMODELING

4.1 Introduction

Remodeling a home is expensive, time-consuming, and disruptive, and it is rarely practical to revise or undo the changes [1]. Unfortunately, it is also hard to imagine what a proposed remodel might actually *look* like; ideally, one would like a realistic visualization of a proposed change to a home interior before any hammers are swung. Professionals can use CAD programs to create 3D models, but considerable skill and time are required to achieve photorealism. The alternative is to use image-based modeling and rendering, which allow photorealistic virtual environments from a few photographs. But how easy is it to *edit* an image-based model? Can large-scale edits, like removing a wall or enlarging a window, be performed easily and in real-time, with realistic results?

At present, the answer is no. The most visually realistic IBR systems combine geometric proxies with view-dependent texture mapping that blend multiple photographs from nearby viewpoints. If the geometry is too simple (planar proxies, as in PhotoTourism [114]) or too unstructured (a collection of polygons, as in Furukawa et al. [40]), there is no easy

[1] Image-Base Remodeling [24] was published in IEEE Transactions on Visualization and Computer Graphics, Jan. 2013 (vol. 19 no. 1). `http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.95`.

Figure 4.1: After modeling an interior through the images, a user can start from an original viewpoint (left), click and drag to pull back a wall between a living room and a bedroom (center left), and view the edited result (center right). Then, the user navigates to visualize the edit from another viewpoint (right).

and effective way to edit it. Simple texture-mapped models are flat and unrealistic, while view-dependent texture mapping has no affordances for editing, since typically a few input photographs are simply blended together to create a rendering. If you suddenly remove a wall, how do you find the appropriate disoccluded bits of texture and blend them in seamlessly, in real-time?

Performing large-scale geometric edits within a realistic image-based virtual environment is an unsolved problem. Using home interior architectural modeling and remodeling as an example application, I present a method for building models using photographs in a manner that allows the geometry and texture to be interactively remodeled. I focus on a *representation* for both the geometry and the texture in a scene that allows large-scale edits to be easily performed and rendered in real time. My interface supports the creation of concise, parameterized, and constrained geometry, as well as remodeling *directly from within the photographs*. This frees the user from the need to manually specify 3D coordinates or maintain a consistent architectural model. View dependent texture mapping minimizes visual distortions while allowing the user to navigate within the home.

The user begins by collecting photographs of the interior, which are processed with structure from motion (SfM) and multi-view stereo (MVS) to estimate camera poses and a semi-dense 3D point cloud. From these inputs, I provide an interactive technique for quickly creating a geometric model of the existing house. The original photographs provide photorealistic, view-dependent texture for rendering. The user may then remodel by adding, removing, and modifying walls, windows and door openings directly within the photographs. All edits are visualized in real-time, and can be viewed from any direction. Actual photographs depict natural lighting, and make the scene appear familiar since existing objects and decoration such as furniture, plants, and so on remain within the scene. The ability to remodel directly within the context of photographs provides a means to quickly experiment and understand the implications of possible changes (Figure 4.1).

The work I present does include some simplifications and trade-offs. Furniture, plants, etc. that are not explicitly modeled with geometry introduce some unavoidable artifacts during navigation. I also assume for now that the majority of walls are oriented in one of two orthogonal directions, and that floors are level. Remodeling operations do not currently

Figure 4.2: Comparison of Google Sketchup's Photomatch using geometry exported from our system. Top row: Google Sketchup, Bottom row: My System. In the center images, the column was removed and a passage opened into a study. Note how Sketchup breaks up polygons due to original occlusions in the image projections. This makes remodeling almost impossible. Also note poor textures in revealed and new surfaces. Right: the same edit viewed from a different viewpoint. My view dependent texture automatically utilizes texture relevant to the new viewpoint.

support large scale additions of new rooms, but rather focus on removal and additions within the existing building footprint. I also leave interior design operations such as lighting changes, furniture and cabinetry edits, and material property edits (such as paint color) to future work.

4.2 Related Work

Commercial tools are available to model existing home interiors and to visualize proposed renovations. Professional CAD software, such as Autodesk's Revit, allow the creation of parameterized and constrained models, so that changes propagate through associated primitives, e.g., changing the height of a wall affects adjoining walls. These systems do not handle the natural lighting nor the clutter in existing structures and thus the resulting visualizations are somewhat artificial. My system of constraints and building primitives are, however, inspired by these systems.

While CAD modeling requires significant expertise, simpler tools such as Autodesk's

Homestyler are targeted at non-experts and can be used to author attractive—but highly-abstracted—floor plan renderings. These tools do not support the creation of photorealistic 3D models based on the original environment.

Google SketchUp allows interactive modeling using photographs with the "Photomatch" feature. This feature is designed to support modeling and visualization by projecting textures onto an existing scene. However, it does not provide a good representation for remodeling. For example, objects are split by occlusions during texture projection, e.g., when a column interrupts a view of a wall. Split polygons make subsequent editing extremely difficult. SketchUp assumes a single, globally-defined texture-mapping, which behaves poorly from novel viewpoints when using coarse proxy geometry (e.g., for plants, furniture, and other clutter), and when making edits to geometry. Combining multiple textures requires manually defining a common frame-of-reference, and there is no support to have the rendered texture respond to viewpoint changes. I illustrate these and other issues with remodeling in SketchUp in Figure 4.2. In contrast, my system is designed to support efficient editing and refinements to geometry, and renders with view-dependent texture, thereby making the editing process both simpler and more visually faithful to the real space.

By starting with a collection of photographs, image-based modeling techniques can create photo-realistic virtual environments with little or no user effort. Automatic modeling and rendering of photographed environments range from systems like Photo Tourism [114], where simple planes are used as geometric proxies for rendering novel viewpoints, to automatic, dense, MVS reconstruction [98, 25, 8, 46, 40]. However, most MVS techniques produce either point clouds or unstructured meshes; such models are sufficient for rendering but not editing, since they lack architectural primitives that can be selected and manipulated, and also lack the connectivity, structural representations, and constraints of easy-to-edit models.

A few methods automatically reconstruct higher-level primitives. Dick et al. [33] reconstruct whole buildings by optimizing a generative model of architectural primitives. Müller et al. [86] use procedural shape grammars to reconstruct building façades composed of regular, repeating structures. A few methods estimate room layouts of simple volumetric primitives, from single images [53, 74]. Our goal, instead, is user-controllable modeling; however, when these automatic techniques become robust enough for general use they should be compatible

with our editing and rendering techniques.

The other approach to creating image-based models is interactive [22, 31, 94, 28, 37, 122]. The system of Sinha et al. [111] is closest to mine, in that they allow "in-image" modeling bootstrapped from an automatically-reconstructed 3D point cloud. However, they focus on reconstructing exteriors as collections of disconnected slabs and polygons which are more challenging to edit than our solid structures. Similarly, Nan et al. [87] interactively fit solid architectural building blocks to 3D point cloud data from LiDAR scans for large-scale urban reconstruction. One advantage of interactive model construction is that the primitives are often larger and more coherent, e.g., a single polygon for a wall. Our interactive modeling system borrows many ideas from these methods and adapts them to building interiors. However, none of these previous systems address the problem of authoring and rendering edits that depart from the original structures being modeled, which is my focus.

A few previous authors have described methods for editing image-based models. These efforts have focused on pixel operations such as painting [107] and cloning [94], or larger operations such as compositing and local deformations [59, 20]. Carroll et al. [18] allow editing of perspective in a single photograph of a building interior or exterior. To the best of my knowledge, my technique is the first image-based modeling system to support large-scale geometric changes such as removing entire walls.

4.3 Overview

Interactive, real-time editing of geometry with view-dependent texture mapping is not a trivial extension of existing techniques. There are three main requirements for such a system.

First, the geometry proxy used during rendering should be concise, parameterized, constrained to architecturally meaningful edits, and have affordances for manipulation. The underlying geometric representation must support a user being able to change the shape and position of geometry without creating holes, while maintaining a consistent, coherent model. At the same time, the texturing operations must automatically follow the editing operation in an intuitive manner. In other words, the geometry cannot be a point cloud or unstructured mesh. I solve this problem by creating a constrained solid geometric representation for our model, and utilize homogeneous 3D texture coordinates for all texture

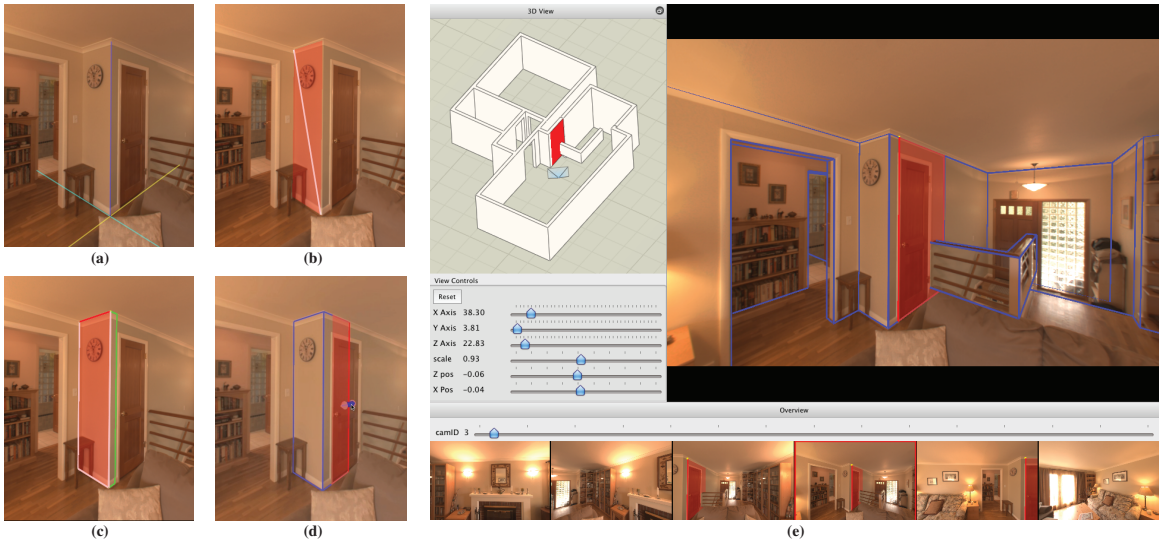


Figure 4.3: ***(a)-(d)*** *A corner composed of two walls is built interactively.* ***(a)*** *The canonical axes of the interior are refined to align with a corner in a photograph; these axes are used for the entire model building process.* ***(b)*** *The user drags out the corners of a wall face. A 3D rectangle is constructed and projected into the image in realtime; the 3D rectangle is estimated based on the floor position and point cloud and automatically snapped to align with one of the canonical axes.* ***(c)*** *A solid wall is added to the model using the default wall thickness.* ***(d)*** *The user can then extend the model, adding walls as needed, ultimately constructing a complex wall unit.* ***(e)*** *A snapshot of the system's interface. The current model with the current view frustum and currently selected primitive highlighted in red is shown in the upper left, rendered orthographically. A wireframe is rendered over the current photograph on the right. A strip of alternate photos is shown on the bottom.*

operations.

Secondly, I need a mechanism to render the scene in real-time. My textures must not only maintain the appearance of the original geometry, but also provide a mechanism to display all geometry that might be revealed during an edit from any viewpoint. Since texture synthesis is currently not feasible in a real-time system, I pre-compute view dependent textures, and organize them in a manner that limits bottlenecks caused by hardware memory transfer. I also provide support for viewpoints not well represented by any view-dependent texture, by computing a view-independent texture to gracefully provide a degraded run-time view when needed.

Finally, a user should have an interaction metaphor with affordances that support modeling and remodeling without a deep understanding of 3D structure. To this end, I create an "in-image" modeling and remodeling interaction metaphor. The user can easily click and drag to create new walls, change existing walls, and perform basic texture modifications, without specifying 3D coordinates or worrying about the underlying 3D point cloud or source photographs. I discuss our solutions to each of these problems in turn in the following sections.

4.4 Interactive modeling from images

The modeling process begins by first capturing a collection of photographs of the interior. Structure from motion automatically recovers camera poses and a semi-dense point cloud. The user then interactively creates a solid 3D model working directly within the images. The 3D point cloud, together with geometric constraints typical of home interiors, are used to simplify the process for the user. The geometric constraints include a common thickness for all walls, and orthogonality of walls and floors; i.e., surface normals are typically aligned with orthogonal, canonical axes (sometimes called the "Manhattan World" assumption). These constraints can be relaxed as needed to handle non-axis-aligned features, such as tilted ceilings and angled walls. I next provide more detail on each of these steps.

4.4.1 Capture and initial reconstruction

Images of the home interior are shot with an SLR and wide-angle (typically, fish-eye) lens, with fixed focal length. I capture images using a tripod with the camera oriented in landscape mode, moving to many different viewpoints to cover the space well. Photographing interiors well is particularly difficult as the lighting in a home can vary dramatically, from dimly lit interior rooms with artificial light, to rooms with floor-to-ceiling windows lit by the sun, to views out the windows themselves. While not required, I get the best results by bracketing exposures (aperture priority, fixed ISO-value and f-number). From these I can recover HDR images in a linear radiometric space shared across all viewpoints, using the recorded exposure settings to scale pixel values suitably for each image. To minimize appearance differences between views, I keep the artificial lighting constant (typically all lights turned on) during capture. Natural lighting may still change during capture due to variations in cloud cover or in the angle of the sun; this variation is not problematic for SfM or MVS, but later requires more sophisticated compositing techniques when combining images, as discussed in Section 4.5. These techniques also help to handle non-HDR images taken under less controlled circumstances. The fisheye images are reprojected to wide angle perspective images using the nominal distortion parameters for that lens. Camera poses and sparse scene points are then recovered by the Bundler SfM tool [112]. It is important to note that the fish-eye distortion must be properly characterized in the bundler tool. Hughes et al. [61] thoroughly summarize various fish-eye models. The Bundler SfM tool has been updated [1] to properly recover the equisolid model used by both a Panasonic GH1 Micro Four Thirds system with a Panasonic fish-eye lens and a Canon EOS 10D with a Sigma fish-eye lens. Finally, I use PMVS [42] to recover semi-dense scene points with normals.

Canonical axes, floor height, and wall thickness

An initial set of canonical axes is recovered with the method of Furukawa et al. [39], which finds three cluster centers of normal directions that are nearly orthogonal. I take the "up" axis to be the axis most closely aligned with the average of the camera up vectors. Cross

[1] `https://github.com/alexcolburn/bundler_sfm`

products with the other cluster directions generate the other two canonical axes spanning a horizontal world plane, call them "east-west" and "north-south". An initial floor height is determined from all points with orientation within 25 degrees of the up direction. I compute distances of these points to the horizontal world plane, perform k-means clustering on these distances ($k = 3$), and take the distance of the largest cluster to be the floor height.

The resulting canonical axes and floor height may not be precisely aligned to the actual home interior. I provide a simple interface for adjusting them. The user clicks on a room corner at floor level in an image, and coordinate axes are centered at the intersection of the viewing ray and the floor and projected into the image (Figure 4.3a). The user can then adjust the orientation of the object coordinates until the axes align with the lines where the two walls meet each other and the floor. Switching to another view of the corner (from a significantly different viewpoint), the user can adjust the floor height until it coincides with the corner in this second view. Finally, the user can specify the global wall thickness through direct measurement in an image, e.g., across the width of a wall that terminates at a passageway.

4.4.2 Modeling within the images

Given the camera poses, canonical axes, floor height and wall thickness, the user is now ready to build interior geometry directly within the images. To create a new wall, the user chooses an image, clicks on a wall and drags out a rectangle (Figure 4.3b). A rectangular portion of a 3D wall appears automatically aligned with one of the canonical wall axes based on the predominance of point cloud normals within the rectangle. The wall position is also automatically determined by the predominant point locations. The wall has thickness and sits squarely on the floor plane (Figure 4.3e). Moving to any new image, the newly created wall appears in place. Fine adjustments can made to the wall position from the new view.

The user continues modeling by extending walls through automatically-created affordances for dragging the ends of walls to the corners of rooms. New wall segments can then be added to turn a corner. The new walls are automatically extended to include the solid intersection of the two walls resulting in a constrained, functional intersection. Continuing

around a room often requires moving from image to image to complete the room. Acceleration tools allow the user to simply extend a wall to meet another already existing wall and to close the loop in a room with a join operation.

Walls can be split to create passageways, and holes can be punched in walls to create windows and doors. The doorway and window holes have affordances automatically created for dragging the edges to adjust to match them to images. Exterior windows auto-generate a separate thin slab for the glass aligned with the outside for texturing. [The modeling process can be seen in the accompanying video.]

Alongside the image being viewed, a flat-shaded, orthographic view of the 3D model is shown, with a frustum icon denoting the viewpoint of the currently selected image. Figure 4.3e illustrates the interface. As the model is updated through the images, the 3D view updates as well; similarly, the user may make simple adjustments to the model in the 3D view and see the updates propagated to the image view. Note that the model does not need to be extremely precise to give plausible visualizations of viewpoint motion and remodeling, as we will see in Section 4.7.

Representation

Observing that most interiors are comprised of abutting walls of common thickness that are oriented parallel or perpendicular to one another, I initially model the geometry as a union of axis-aligned, fixed-thickness rectangular solids. I represent this model as a set of polygon meshes, each traversed and managed with a half-edge data structure [36]. Each mesh is closed, yielding a solid model. This provides a data structure for fast access to mesh connectivity information for mesh editing operations, initial texture creation, and runtime texture re-assignment. Edits are constrained to those deemed architecturally meaningful. Thus, for example, grabbing the edge of a doorway and moving horizontally is interpreted as opening the doorway. Or, grabbing the end of a wall and pulling along the wall normal moves the whole wall which might also automatically shorten or lengthen connecting walls.

4.5 Precomputed View-Dependent Texture

The heart of what makes my system effective for real-time remodeling is the crafting of precomputed view-dependent texture atlases. The goal is to provide the ability to view the geometry with a photorealistic quality, while being able to navigate and, more importantly, alter the geometry to visually assess remodeling operations.

My work draws on view-dependent texture mapping (VDTM) [31], which renders new viewpoints as a weighted blend of photographs taken from nearby viewpoints. Like Photo Tourism [114], my user interface encourages the user to be at or near the original viewpoints except when transitioning between views. A key advantage of VDTM is that rendering the original geometry from the point-of-view of where an image was captured results in exactly the original captured photograph.

However, conventional VDTM does not support live changes to the scene geometry. For example, suppose that a wall is removed, revealing the geometry behind. Since this geometry was not visible, VDTM provides no guidance on how to texture the revealed surface. One could generate texture for the newly-visible geometry by searching for other views of it among the input photographs. There are a number of problems with this approach. For one, there often is no single good view of the newly-visible geometry, requiring blending or hole-filling to produce realistic results. Second, searching the input images is too costly for real-time rendering. Consider the viewpoint rendered in Figure 4.6. This scene has low depth complexity, yet the texture generated for the newly visible geometry in the bottom image uses texture from 150 photos. The more complex scene in Figure 4.4 uses data from 261 cameras. The memory transfer rates of our test machine from the host to the CPU is roughly 2500 MB per second (as measured by oclBandwidthTest). If each images utilizes roughy 10 MB of texture RAM, the memory transfer alone would take a significant fraction of a second when assembling texture on the fly. The full set of images would require more RAM than the host's memory.

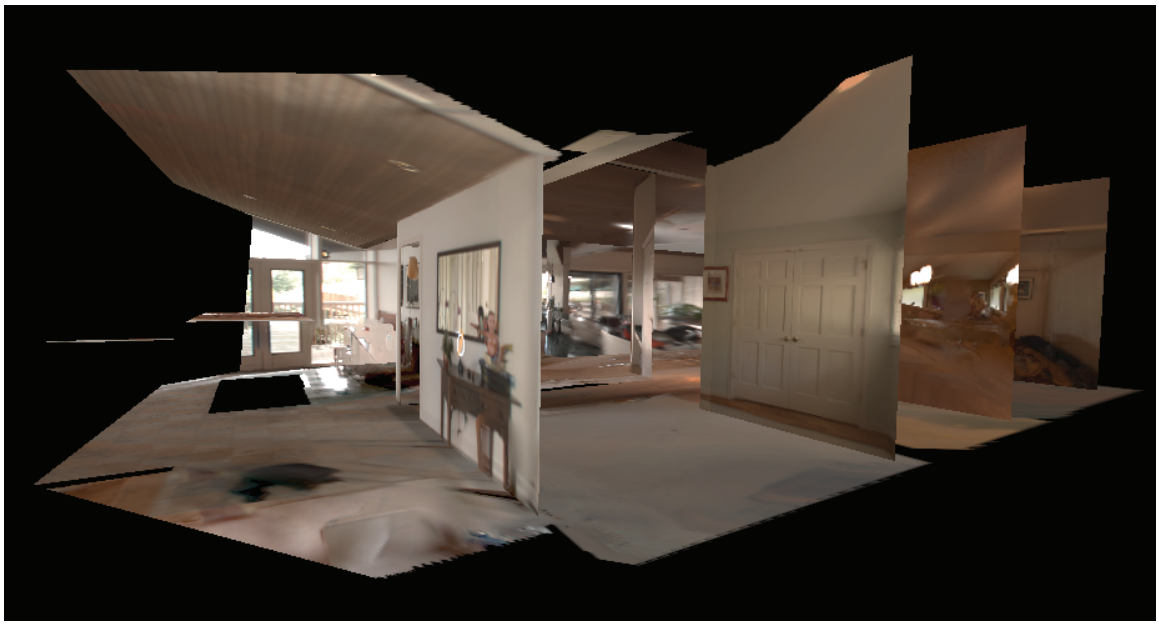


Figure 4.4: A VDTA for a single viewpoint viewed from the side; note that possibly dis-occluded textures are pre-computed.

4.5.1 View-Dependent Texture Atlases

To address these problems, for each view, I create a *View-Dependent Texture Atlas* (VDTA) as seen in Figure 4.4. In addition to the VDTA for each view, I also create a single, low resolution, global *View-Independent Texture Atlas* (VITA), which is similar in size, form, and function to the VDTA. The VITA fills in any texture gaps during transitions between adjacent VDTAs.

Each VDTA stores a texture for all polygons that could possibly be seen or revealed in that view—that is, all faces within the view's frustum. The textures are pre-projected to the corresponding viewpoint (i.e., with perspective foreshortening). This avoids wasted texture resolution on polygons that are highly oblique to the view. The textures are packed into a set of buffers to preserve memory coherence and to greatly reduce the number of unique textures required.

I pack the textures using a classic greedy first-fit algorithm [64]. The number of buffers is dependent on the page size of the buffer. I choose a page size of 1024×1024 which typically results in 3 or 4 buffers for a given view. I order the textures from largest to smallest, placing

Figure 4.5: A viewpoint is rendered using the VDTA (top). The bottom image is the same viewpoint using the VITA. Note the perspective distortions, and flattened clutter

Figure 4.6: Remodel with central walls removed. Images used to create the VDTA were captured on separate days under different lighting conditions, without HDR exposure bracket capture. The VDTA construction is robust to such lighting changes.

them sequentially in the first empty block in which they fit, continuing until all textures are placed. This simple scheme sufficiently obviates the need for frequent texture buffers switches when rendering.

The VDTA has an associated list of faces textured in each buffers, and a projection matrix that transforms for each face's 3D coordinates to image coordinates in the buffer. This is simply the projection matrix of the original viewpoint recovered by SfM translated and scaled to correspond to the position in the packed texture.

Since the texture for any given face may be clipped by the viewing frustum, I take care during rendering to only use texture coordinates within the original field of view. Validating texture coordinates uses a matrix multiply and bounds checking between the texture coordinate and the original viewpoint.

The texture atlases reduce the total number of runtime texture buffers per viewpoint from hundreds to a few combined texture buffers, while also reducing the total memory requirements by half. Expensive runtime texture swaps are reduced from hundreds per frame, to a few for each active VDTA. Once computed, any face that might become visible to any view, is rendered using only the VDTAs from nearby views, with some possible fill-in by the VITA. While texture atlases have been used before for image-based rendering [16], these methods do not support real-time rendering of edits.

4.5.2 Synthesizing the View-Dependent Textures

Having computed an assignment of faces to texture buffers, I still need to determine the appearance of each face. Each face can combine three kinds of texture:

1. **Original**: texture contained in the photograph from this viewpoint.
2. **Warped**: texture occluded from this viewpoint, but seen in other photographs.
3. **Synthesized**: texture not seen from any viewpoint.

The following procedure is used to fill the view-dependent texture atlas corresponding to each face.

Original Texture

Visibility of portions of a given face from the original viewpoint is computed using an ID buffer. Any pixels that are visible are copied directly from the original photograph. However, pixels within five pixels of any significant depth discontinuity are discarded. This prevents slight misalignments of depth discontinuities from creating erroneous mappings, such as a post being projected onto a wall far behind it. The remaining pixels of visible portions of faces are then filled from nearby viewpoints, or by hole-filling if not seen by other cameras.

Warped Texture from Other Views

Filling pixels occluded from the original viewpoint but visible in others requires solving two problems. First, I need to choose which other viewpoints to copy texture from. Second, I need to modify the copied content to blend seamlessly with the original view. The latter step is required because small lighting changes, non-Lambertian effects, and vignetting can lead to differences in appearance between different photographs of the same geometry.

As with the original view, candidates for views to fill-in occluded pixels are determined by projecting the occluded pixels into the ID buffers from each candidate view. If the ID matches (and is sufficiently far from a depth discontinuity) then this view becomes a candidate for fill-in. Of the possible candidates, I choose the best one by leveraging the criteria used by Unstructured Lumigraph Rendering (ULR) [17]. Pixels are filled from a viewpoint that is similar in 3D location and view direction of the original view, and with a pixel coordinate towards the center of the image to avoid vignetting and other distortions. I use these criteria to select the single best candidate for each occluded pixel.

To solve the second problem, I composite using both the texture from the selected viewpoint and the corresponding texture gradients through the use of gradient-domain blending [95]. A weighted least-squares problem is solved where the occluded pixels' RGB values are variables. Original viewpoint pixels act as hard boundary constraints.

I iterate over each viewpoint that has been selected to be *best* for at least one pixel and render the face into a buffer. Each *best* pixel contributes a weak data constraint: the final composited color should be close to the rendered color from that viewpoint. Each pair of *best*

neighboring pixels, and each adjacent *best* and non-*best* pair of neighboring pixels, contribute a gradient constraint: the difference between neighboring composited pixels should equal the difference between the pair of rendered pixels. The weak data constraints are weighted by a factor of 10^{-5} relative to the boundary and gradients constraints.

As noted by Bhat et al. [12], outlier gradients can cause bleeding artifacts when compositing in the gradient-domain. I suppress the influence of large gradients; if the gradient is larger than 0.1, I down-weight the constraint by a factor of 10^{-5}. Finally, I solve the the weighted least squares problem to create a composite.

Synthesized Hole Filling

Some pixels are not seen from any original photograph. These pixels are filled with a gradient-domain blend [95, 12] using the average color of the face as a weak constraint. More sophisticated hole filling methods could also be used, but is left for future work.

The above process is repeated for each viewpoint, beginning with the original view, then other views to fill in, and finally hole filling. This completes the off-line construction of the VDTAs.

4.5.3 View-Independent Texture Atlas

Creating the single view-*independent* texture atlas (VITA) proceeds in a similar fashion to the VDTAs with two exceptions. Since there is no preferred view direction, each surface is projected orthogonally before packing and texturing. There are also no original (preferred) textures, so all texture pixels are derived from either a warped texture or hole filling. Best views for each pixel are selected with the same unstructured lumigraph weighting, resulting in a view that is most orthogonal to the surface, and closer to the surface. Pixels from competing views are blended using the gradient domain blending as before but with no hard boundary constraints. The weak data terms from each view provide the anchor for the optimization. As before, any pixels not seen from any view are filled with synthesized hole fill.

4.5.4 Texturing New Surfaces

Finally, I also need to be able to texture newly-created surfaces resulting from remodeling operations. These occur both when a completely new wall is added, but also when a new doorway is cut through an existing wall, since new, small, wall-thickness faces are created to complete the passageway. If the newly created face has co-planar faces, we extend the texture associated with the coplanar face by simply repeating the neighboring texture edge. For example, if one of the walls of an outside corner (Figure 4.7) is extended, the texture on the edge of the co-linear wall is simply stretched to fill the new wall. New surfaces without a co-planar surface are given a default wall color. The user may override the initial texture by copying texture from a selected rectangular patch on an existing wall. Newly created textures are treated as view-independent and thus share the same texture across all views. This can cause minor artifacts when moving between views, but provides a good balance between visual fidelity and minimizing the need for user intervention.

4.6 Rendering

Given the geometric model and the complete set of VDTAs, I can render from any vantage point in real-time, regardless of which faces are revealed. Rendering is performed at interactive frame rates using a combination of OpenGL and GLSL shaders. The geometry is textured from our VDTA iterating over each texture buffer. I load each texture buffer, and render only those faces associated with the buffer. Using a pixel shader, the homogeneous 3D texture coordinates of our model are transformed to 2D texture coordinates by multiplying the coordinate at each pixel by the texture matrix of the source texture taking care to avoid out of bounds projections. There are two boundary tests that must be performed. First, I discard texture coordinates projected outside of the original viewpoint. Secondly, I discard texture coordinates projected outside of the texture matrix.

When rendering from an original viewpoint, only the VDTA from that view is required. When navigating between viewpoints, the scene is rendered three times using VDTAs for the source and destination camera views, and the VITA to three off-screen buffers. The VDTAs are linearly blended to form a smooth transition, and the VITA is only used to fill

any remaining holes.

The pixel shader adjusts the texture for exposure compensation. By default, the values are scaled to set the saturation point to be the same as the camera's auto-exposure would. A GUI is provided to override this for for manual exposure if desired.

4.7 Remodeling

Once the initial modeling is complete and VDTAs computed, the user can navigate through the model, change the virtual camera exposure, and remodel the structure by adding/removing walls, or adding and expanding/shrinking openings. All changes to the viewpoint and model are rendered in real time. Most importantly, the geometric model and texture atlases support rendering during remodeling operations.

As with the creation of the original model, remodeling operations are carried out in the original views with a similar set of tools. The user is able to click on a wall, and then grab edges of the wall or any opening and simply slide them to make the wall or opening larger or smaller. As an opening is enlarged (as in Figure 4.1), the room behind the wall is revealed by rendering those faces previously occluded in the original view. Widening a window (Figure 4.7) simply stretches the outdoor texture on that window since we have no additional textured geometry for the outdoors. Textures on walls are similarly stretched by locking the vertex texture coordinates as is. In some situations it is preferable to avoid a stretched texture appearance by cropping the texture. In this case the texture coordinates are updated during editing and constrained to the texture limits, (Figure 4.7, fourth row). As expected, all geometry changes are global and thus appear in all camera views.

4.7.1 Navigation

In addition to simply switching between original views as displayed in the filmstrip in Figure 4.3, the user can change the view more continuously. I support two modes. In the first mode, using the GUI or keyboard, the user can move left/right and forward/back, or turn the view direction left/right and up/down, and zoom in/out. Translational motion introduces parallax which accentuates the 3D nature of the model enhancing the visualization. This is made possible by the fully-textured layered model. Newly revealed surfaces, for example

through doorways appear naturally. Because the details such as furniture are not fully modeled, parallax does cause some artifacts.

The second mode involves moves from one original source view to a final destination view. The textures are interpolated between these two views as described in Section 4.6. Fly through paths are automatically created by interpolating between intermediary viewpoints with a similar motivation to that done in [40] and [113]. The idea is that the picture takers intuitively select good positions from which to view the scene. To determine the path from source to destination we construct a graph with a vertex for each original viewpoint. Each vertex has an edge to another vertex if a line segment between the camera centers does not intersect the model. I set the edge weight w_{AB} between two views, A and B, to

$$w_{AB} = 1 + d_{AB} + \alpha \frac{\theta_{AB}}{d_{AB}}$$

where the 1 penalizes additional path segments, d_{AB} is the Euclidean distance between the camera centers, and θ_{AB} is the angle between the optical axes of the cameras. I set $\alpha = 2/\pi$. I then solve for the shortest path between source and destination using Dijkstra's method.

When traversing the path, I treat each node, except the endpoints, as a virtual camera with camera center the same as the original viewpoint's, but the optical axis pointing in the direction from the that node's camera center to the next node's camera center. I then linearly interpolate the camera centers and optical axes when moving from node to node. An extra node is inserted along the edge leaving the source and another along the edge arriving at the destination, positioned so that it takes one second to ease out of the starting orientation and 0.5 second to ease into the final orientation. Thus, the camera orientation begins with the source orientation and over one second interpolates to point along the path. One half second before the destination is reached, the orientation begins to interpolation between the path direction and destination camera orientation, ending finally on the destination camera position and orientation.

4.8 Results

I have used my image-based remodeling technique on three house interiors. [The most effective way to see the usage and results is through the accompanying video.] Each photograph

Figure 4.7: Three remodeling results. First column: original viewpoint. Second column: remodeled geometry. Third column: same remodel from different viewpoint. First row: opening a wall to connect two rooms. Second row: expanding a window vertically. Third row: adding a wall segment. Fourth row: creating a cutout.

set was captured with a Canon EOS 10D and Sigma fisheye lens. The first two sets were captured with three bracketed images separated by two stops.

The first data set seen in Figure 4.3 has 143 viewpoints, and requires 15 minutes of interaction to yield a model of 223 faces. This is followed by about 30 minutes of processing time on an 18-node cluster to produce the VDTAs. The input photographs comprise 2.5GB of camera RAW files, processed into 1.3 GB of HDR files in EXR format, resulting in 1.1GB of VDTA files, also in EXR format.

The second data set visualized in Figures 4.1, 4.4, and 4.7 has 462 camera viewpoints, and required about 45 minutes of interaction to produce a model of 661 faces, modeling 8 rooms, followed by 2 hours of processing to produce the VDTAs. The input photographs were processed to 2GB of HDR files, resulting in 5.2GB of VDTA files.

The third data set shown in Figure 4.6 has 326 viewpoints, captured on separate days under different lighting conditions without bracketed exposures. It required 30 minutes of interaction to model 5 rooms with 438 faces, followed by 1.5 hours of preprocessing to produce the VDTAs. The input photographs were processed to 1.7GB of HDR files, resulting in 2.84GB of VDTA files.

Rendering during modeling, remodeling, and navigation proceeds at 50 frames/second. Remodeling operations can be seen in Figures 4.1, 4.6, and 4.7. These operations include widening doors, removing support beams, removing and adding walls, and resizing a window.

4.9 User Feedback

I conducted an informal study to solicit feedback on the use of the system for remodeling. I included five subjects, with professional experience ranging from two professional architects and designers with extensive CAD experience to three novice modelers including the home-owner of one model. Each subject was asked to navigate through viewpoints in a house, and then to perform several remodeling operations on a pre-constructed model. These included expanding a doorway as well as punching a hole in a wall between rooms. Note, I did not ask the subjects to construct the initial model from photographs although I did show them how the models were constructed. I was most interested in the perceived benefits of the image based remodeling visualization.

The subjects universally said that they experienced a sense of being in the house. They all felt that they knew what the house was like, even if they had never been there before. The realistic image based rendering enhanced their belief that they understood the result of a remodel. The owner of one house expressed an emotional response to the remodel operations. She said that "seeing real photos of my stuff added realism, and made the remodel seem more predictable." The architects liked the fact that a coarse model could be built quickly and was usable for rapid prototyping, exploring ideas, and visualizing changes. They thought our system could save time and effort by reducing the level of detail required in traditional models, details that are never used except to provide more realism. They wished the results could be integrated or exported to traditional CAD systems for further refinement and architectural notation and standards support. None of the users noticed the lack of relighting when asked.

4.10 Discussion

I have demonstrated a new interface for home modeling and remodeling using a through-the-photograph methodology. A synthesis of computer vision technology, texture atlas construction, and user-interface results in a system that provides an intuitive way to realistically visualize remodeling ideas in real-time as one edits the geometry. I have demonstrated how models can be constructed efficiently, and textured in a way that enables visually plausible re-editing. Pre-computation of view dependent texture atlases allows visualization of edits in real-time.

Although capturing large sets of images can be somewhat tedious, and the pre-computation is expensive, I am confident the advantages of freeing the user from having to model in an abstract space, coupled with the real-time realistic visualization provides significant advantages over current alternatives.

My technique has a number of current limitations. Since I do not model all the geometry precisely, there are rendering artifacts, most commonly in the rendering of household objects such as furniture and plants. Architectural details such as baseboards and window or door trim are also not modeled. I hope to leverage deeper architectural knowledge to help complete such details.

I do not currently support relighting, however my system has many advantages that can make this operation possible. For example, our HDR images provide environment (i.e., illumination) maps capable of acting as light sources. These maps plus the geometry can help separate lighting from material properties. While lighting inconsistencies due to remodeling edits were not apparent to my small sample of users, this is an interesting problem for future work. Diffuse component relighting can be approximated by simply comparing the virtual views from a surface point before and after an edit. Specular components can be similarly approximated but require sensitivity to incident and outgoing lighting directions. In general, the full global relighting problem is more complex, however the work of Yu et al. on inverse global illumination [128], Ramamoorthi et al. [101] on inverse rendering, and Romeiro et al. on blind reflectometry [103] provides some guidance and inspiration for my future work.

Finally, it would be helpful to support larger-scale edits such as adding a complete new room extension. This will require building a hybrid system that supports both image-based rendering where possible and more conventional modeling and rendering elsewhere. Again, I expect to leverage the captured imagery to aid in the description of the new geometry. An important question is how to visually represent large-scale additions. Should the addition be a blend of a stylized architectural rendering with the existing imagery, or should the addition visually mimic the existing structures in a seamless manner?

Achieving a visually seamless mix of novel geometry and image based rendering is an interesting problem. First, the material properties and texture of the new geometry must match the appearance of the existing geometry. Secondly, the lighting model must be consistent across the existing and new models. Recent work using incident light fields by Unger et al. [121] and work using user annotations by Karsch et al. [67] demonstrate how well one may integrate artificial objects into image based rendering environments within complex light fields. However the artificial geometry does not yet take on visual characteristics of the environment. I would like to explore extracting incident light fields from our existing data and apply them as light sources for novel geometry.

I believe that the techniques I have demonstrated to date can be extended into a complete system for visualizing remodels. Also, allowing users to edit directly within images should

open such applications to a much wider set of users.

Chapter 5

RELIGHTING AND SYNTHETIC OBJECTS

5.1 Introduction

The image-based remodeling system (described in Chapter 4) demonstrates a method for creating editable image-based geometry. While this technique enables further editing, it does not address the issues of estimating light sources and material properties within the model. In particular, there is no mechanism to properly render lit synthetic objects in the scene or to propagate lighting changes due to edits into the model. Relighting is particularly important when openings (e.g., doorways and windows) have been added to the model, or synthetic objects such as chairs have been inserted into the model.

5.2 Challenges of Relighting Image-Based Models

Relighting the image-based geometry created by our system is a difficult problem because of the incomplete nature of the geometric models and image data. Like many image-based models, our geometry has the following limitations:

- The lighting is unknown and captured under slightly varying conditions.
- The geometry is coarse, imprecise, and contains some occlusion and alignment errors.
- The material properties of the geometry are unknown and may be non-Lambertian.
- The materials properties are not segmented and may vary over the span of a geometric face.

My system utilizes coarse geometric proxies for the image-based models to simplify creation and editing processes. Additionally, I do not have perfect knowledge of lighting, since I have shafts of light coming in from windows and un-modeled lighting. Finally, in the work described in Chapter 4, I do not specify geometric material properties or delineate regions of the same (even if unknown) material properties.

In this chapter, I am primarily focusing on the challenge of determining unknown lighting conditions using Lambertian assumptions. It is important to note that even if the light sources are known, I would still need to overcome the problems coarse geometry, and unknown material properties to build a complete relighting solution. I hope by first tackling the problem of unknown light sources and modeling the light in a scene, I can provide a better framework to overcome the remaining challenges in future work. Following the lead of [67] which estimates the lights using a single photograph, I developed an interactive approach to estimate scene light sources. My system has the advantage of higher quality source imagery and model information due to using several photographs; therefore, I should be able to more easily and accurately model light sources. Once I have a light source model, I can attack the problems of rendering synthetic objects, re-lighting geometry with Lambertian material properties, estimating geometry with specular material properties, and re-lighting that geometry.

5.3 Determining Illumination Information

My first goal is to estimate the incident light at each surface point in the scene. That is to say, I want to estimate the incoming light or irradiance (Equation A.5) for all visible surfaces. If I had a camera or light probe at each point in the scene, or a setup similar to Unger et al. [120, 121] with thousands of light probes this problem would be much easier. Since I only have a few fixed locations I lack much of the data required to directly estimate all of the light from images. For example, consider a point on a floor in the room depicted in Figure 5.1. This room has a large sunlit window which provides most of the light in the room. If the point lies in direct sunlight, and can "see" outside the window, I would be able to estimate the light coming in from the sun. However in this example, none of the images used to create the model were captured from the viewpoint on the the floor, and thus the full sunlight coming in from the window was not captured. In this case the the sun is occluded in all of the captured images, so no texture mapped representation of the window could simulate the sunlight ray falling on the floor. Unless one takes care to capture the light coming in from the window, it is difficult to estimate irradiance for points similar to the one on the floor. Additionally, there are directional lights such as spotlights in the

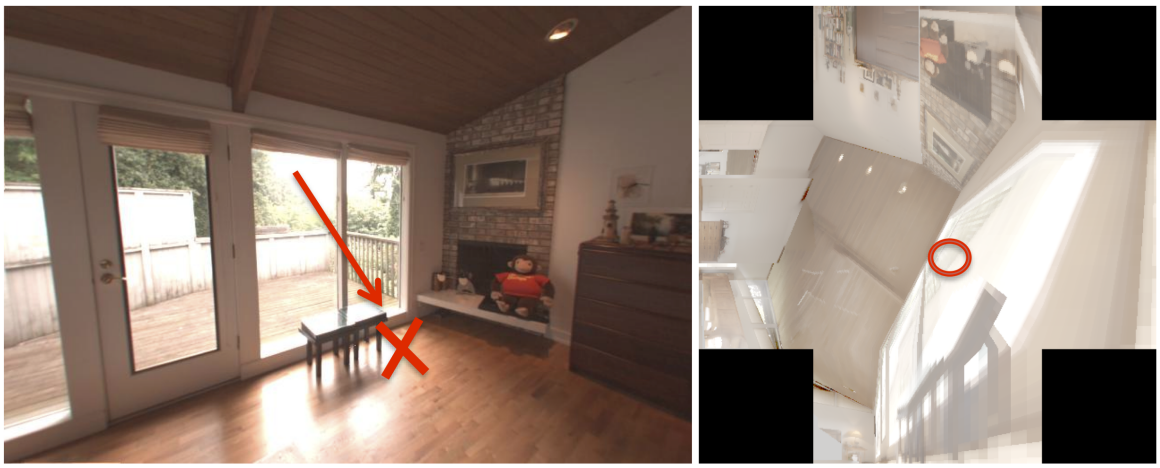


Figure 5.1: Left: A point on the floor receiving light from the window. The point on the floor is marked with an X, and a ray of light from the exterior is indicated with an arrow. Right: A rendered image from the viewpoint of the point on the floor. This image is rendered using a global texture map (the VITA) to represent the light coming in from the window. Since this texture is generated from a different viewpoint from the point on the floor, the source of the ray of light indicated by a circle does not have the same intensity as the actual ray of sunlight. Any rendering from this viewpoint using a VITA will mis-represent the light intensity of the scene. In this case, the ray of sunlight is not captured from any viewpoint and the intensity error is so great that it can not be reliably used for any subsequent calculations.

ceiling in this scene that are difficult to capture without photographs and setups specifically designed to capture their directional variation. It is difficult to estimate these directional and occluded light sources from a few photographs; however, the remaining indirect light in the scene can be fairly well approximated using the existing images. In order to solve this problem of estimating the incident light at each point in the scene, I can provide more data in the form of a few more selectively chosen photographs of sunlit windows, and provide a mechanism to better estimate the incident light at each point in a scene from existing interior photos. To better estimate the incident light I can further break the problem down into two parts: Estimating direct light from sunlight and other light sources, and estimating indirect light from images used to create the model.

5.3.1 Estimating Irradiance

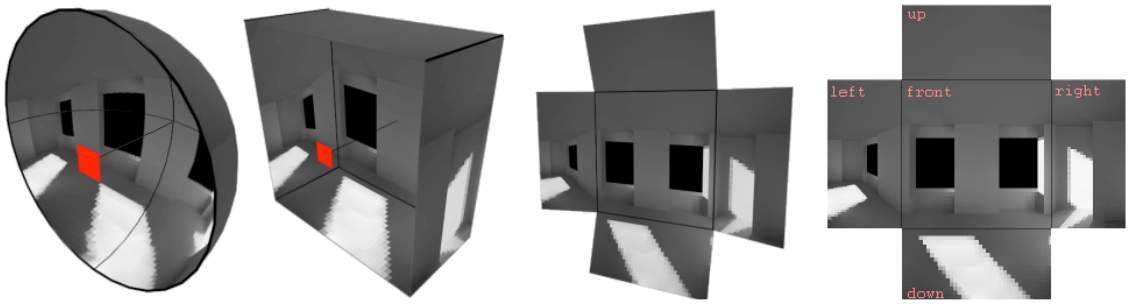


Figure 5.2: From left to right: A 180° hemispheric view from a point in space. A hemicube representing the same view by projecting the scene onto 5 planar sides of a cube. The hemicube unfolded onto a single planar image. Images courtesy of Hugo Elias [62]

Based on the global illumination radiosity equation [47] that assumes a Lambertian world, the radiosity, B, (the light leaving a surface) can be determined by solving a set of simultaneous equations given by:

$$B_i = E_i + \rho_i \sum_j B_j F_{i,j} \tag{5.1}$$

Where B represents radiosity of a surface patch, E is the emissivity of the patch (non-zero only for light sources), ρ is the reflectivity, and F is a form-factor. The form-factor is the geometric relationship between pairs of patches that defines the fraction of light leaving one patch and arriving at another. In the global illumination setting, the reflectivity ρ, and the geometry encapsulated in the form factor are known. The radiosities are the unknowns and thus this defines a set of simultaneous linear equations.

In this setting, I want to know B_i/ρ, the irradiance, the light arriving at a point in the scene located at some pixel. This is given by the summation:

$$\sum_j B_j F_{i,j}$$

I assume I know the B_j as given by our VITA. I choose to use the VITA rather than the VDTA because it is a simpler solution than blending between several VDTAs. Given

$$F_{dA_i A_j} = \int_{A_j} \frac{cos\theta_i cos\theta_j}{\pi |r|^2} dA_j$$

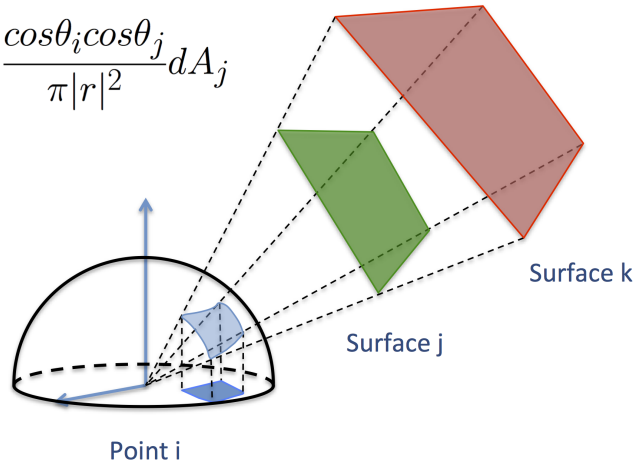


Figure 5.3: Nusselt's analog can be used to calculate the form-factor between a differential point and a differential surface. In the equation above, r is a line segment between the $Point_i$ and $Surface_j$. θ_j is the angle formed between the surface normal and r, and θ_i is the angle formed by the point normal and r. If the projection of $Surface_j$ on to a unit hemisphere is the same as that of $Surface_k$, the form-factor value is the same. More generally this is also true for projections onto any other surrounding surface.

this estimate of radiosity, I only need to compute the form-factors. For this I turn to the hemicube described in [23].

The hemicube is used to simplify the form-factor calculation. It achieves this by unifying all of the form-factor calculation into a single easily computed format. Using the Nusselt analog [110], I know that the projection of any two geometric primitives onto a surface that occupies the same area and location have the same form-factor value (see Figure 5.3). Expanding on this idea, I can project each geometric primitive onto a surface covering a point. I can then substitute the form-factor of geometric primitive for that of the surface. When combined with a standard polygon renderer with visibility culling this mechanism provides an effective means to choose which geometric primitives should be included in the summation.

I render a 180° view of the scene from a point on surface and compute the irradiance from the rendered image. To further simplify the rendering mechanism and form-factor computation, I compute a hemicube rather than a 180° hemisphere (as seen in Figure 5.2).

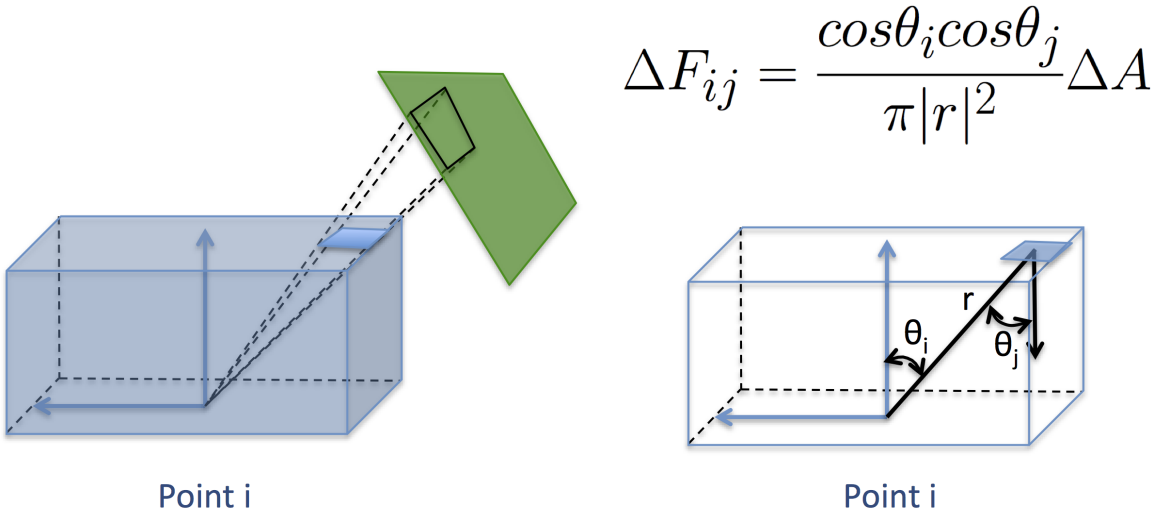


Figure 5.4: Left: A hemicube pixel element projected onto a surface. Right: The $\Delta form-factor$ equation. r is a line segment between the point on a surface and the hemicube pixel position. The hemicube pixel has an area of ΔA. θ_i is the angle formed between the surface normal and r, and θ_j is the angle formed by the hemicube face normal and r.

I render each side of the hemicube using a standard renderer as seen in Figure 5.4. Each geometric item is projected onto the hemicube with occlusion culling. The analytical form-factor computation between each geometric object becomes a discretely sampled Δ form-factor computation for each pixel sized planar object in the hemicube. This Δ form-factor can be quickly pre-computed. The Δ form-factor for each pixel in the hemicube is defined as:

$$|r| = \sqrt{x^2 + y^2 + z^2} \tag{5.2a}$$

$$\Delta F_{ij} = \frac{cos\theta_i cos\theta_j}{\pi |r|^2} \Delta A \tag{5.2b}$$

$$\tag{5.2c}$$

Where x, y, z, are the pixel positions on the hemicube faces in the coordinate system defined by the point on a surface. r is a line segment between the point and pixel position x, y, z. θ_i is the angle formed between the surface normal and r, and θ_j is the angle formed

Figure 5.5: Top: original image. Bottom: Irradiance calculated by hemicube integration at every pixel using a VITA texture mapped rendering. The rendered intermediate image used for the hemicube calculation is shown for the point marked by the red square.

by the hemicube face normal and r.

The irradiance at a point is estimated by summing all of the pixels of the hemicube scaling by the Δ form-factor.

Traditionally the hemicube technique is used to calculate light transfer in global illumination rendering, given geometry with material properties and light sources in a closed scene. In this scenario, a hemicube pixel is not a color value, but rather the geometry identifier used when solving the set of linear equations. In contrast, our photographic images are already globally illuminated, and I do not yet have light sources or material properties defined for our model. I assume that texture from the photographs and therefore the VITA, at least as far as reflected or indirect light is concerned, already accurately describes the scene's actual lighting state. The hemicube is rendered using the VITA to texture map the geometry in the scene. In this case, I do use pixel values and the sum of the pixel values

scaled by the Δ form-factor gives us an estimate of the irradiance at that point.

Figure 5.5 shows an example of a hemicube integration using only the VITA texture mapped geometry. Our system uses the VITA for simplicity, and as expected this technique is sufficient for estimating indirect light in Lambertian scenes. I assume that our scenes are Lambertian. However, since the VDTA locations are sparse, and the VITA is a global texture, I must augment the model in order to account for light sources such as sunlight and spotlights.

I model irradiance at each point in the scene as the sum of indirect light, and direct light sources such as natural light from windows, and artificial lights. The indirect light is modeled with the VITA. Both the natural and artificial direct light source models will be described below. Modeling direct light sources changes the rendering pipeline for the hemicube calculation. Rendering geometry designated as a direct light source (or a *light*) is no longer as simple as a texture lookup. To be more general and using the notation in Appendix A, when geometry at location x is rendered in a scene, I need to know the radiance at the location x given a viewing direction θ. $L(x \rightarrow \Theta)$ represents radiance leaving point x in direction θ. In our implementation, the renderer invokes a lighting function that calculates $L(x \rightarrow \Theta)$ and returns a light intensity. In the case of indirectly lit geometry, $L(x \rightarrow \Theta)$ is simply a VITA texture lookup. For direct light sources $L(x \rightarrow \Theta)$ becomes a more complex function. Looking further forward to Section 5.5, rendering geometry with associated material properties likewise computes a $L(x \rightarrow \Theta)$ returning the reflected light at that location taking into account material properties and a reflectance function.

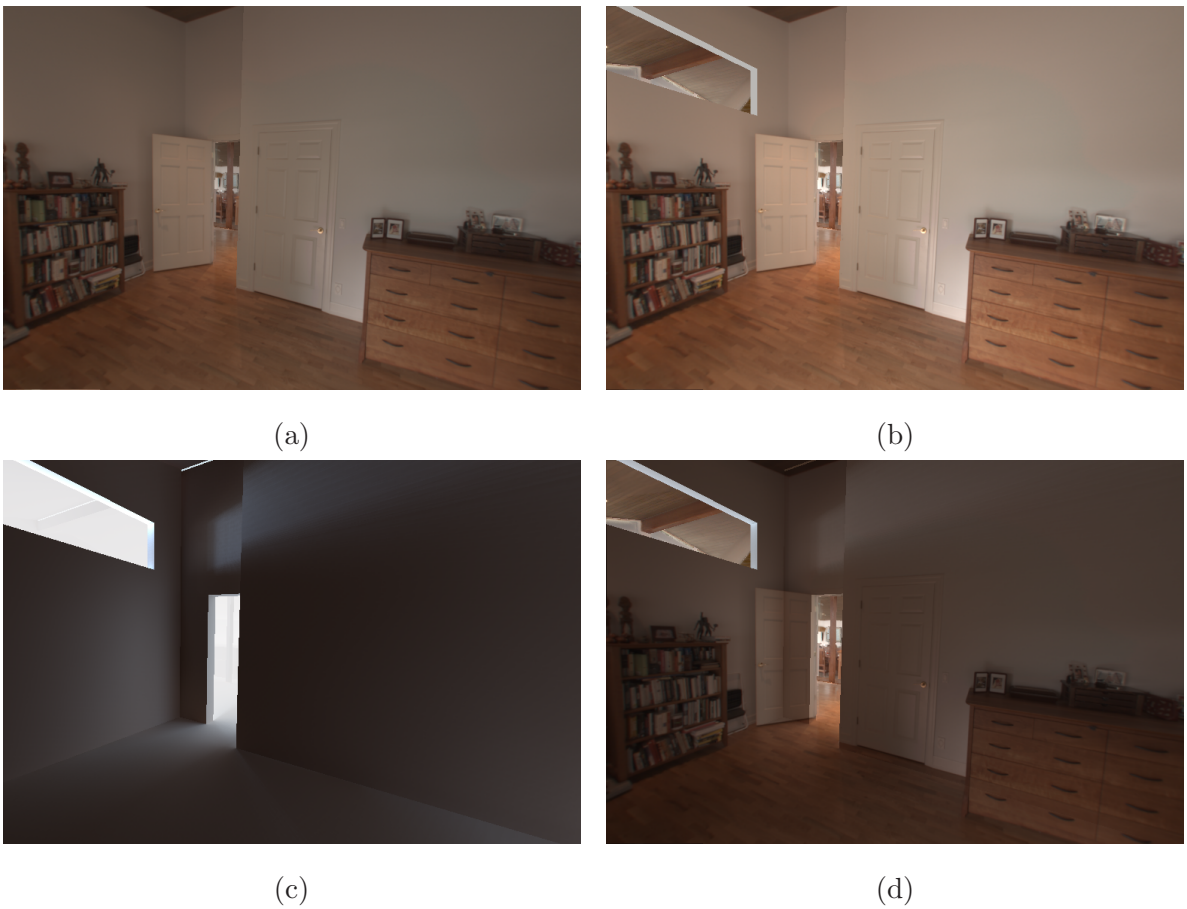
5.3.2 Modeling Direct Natural Light Sources

As may be seen in Figure 5.6, many scenes include windows or glass doors that function as light sources in the scene. To model such natural light sources, I assume that a hemispheric panorama with large radius can sufficiently model this exterior natural light. If the exterior world is far away, I do not have to worry about parallax or other depth effects when utilizing the images. I capture the images directly out the window or glass door in addition to the other images. These light source images are referred to as lighting reference images

Figure 5.6: Three views of a room lit primarily from daylight via the window on the left and from overhead directional spotlights. The sunlight source from the window is out of view and not well defined. The spotlights are directional and poorly modeled with a global texture. The floor is a highly specular surface with non-uniform texture. In the rightmost photograph, the floor model polygons are visible. The coarse geometry does not correspond to a single material property and the material properties are not segmented or otherwise separated. Many items, such as the book shelf and clutter, are un-modeled.

or *LRI*s. Similar to Debevec's [30] light model description, LRIs are distant light sources and remain unaffected by local lighting changes. LRIs are captured and run thought the modeling pipeline no differently from other photographs described in (described in Chapter 4). However, when used for direct light rendering, I do not convert the fisheye image into a standard perspective view. The LRI retains a fisheye camera model in order to maintain a full 180° field of view.

I use an interactive editor to associate model geometry, such as a window corresponding to a single LRI or multiple LRIs. This geometry is termed a *natural light*. When rendering a natural light, the lighting function that computes the $L(x \rightarrow \Theta)$ utilizes the LRI as a hemispheric panorama rather than texture mapping the VITA associated with the window geometry. The 3D location x, is first used to selected the nearest LRI to use as a texture, if multiple LRI's are associated with the geometry. The panoramic scene is assumed to be far away, the ray from the camera viewing direction Θ to x is simply converted to a ray in the LRI's fisheye camera model. The ray is then mapped to the corresponding pixel value in the LRI and returned to the renderer. In this way, I model the directional light of the window (as in Figure 5.8), without needing to explicitly model the exterior features of a structure such as the sun or trees.



(a) (b)

(c) (d)

Figure 5.7: Edited geometry can be relit using irradiance ratios. The top row shows the original **(a)** *and edited room* **(b)** *without relighting. A cutout is created in the wall and a bright exterior window behind our view is "closed". One would expect this edit to darken the room, and add a light streak to the wall above the doorway. The bottom row shows the irradiance ratio for every pixel* **(c)**. *Each pixel in the edited scene is multiplied by the irradiance ratio, resulting in the relit scene* **(d)**.

Figure 5.8: Top row: A hemicube view changing position from left to right. Bottom row: The kitchen window regions that utilizes an LRI lighting function is enlarged to enhance the view. The direct natural light function that computes $L(x \rightarrow \Theta)$ utilizes an LRI panorama. In this example we can see that the light computed by the LRI changes as hemicube position changes. The kitchen window is a natural light, and as the hemicube viewpoint changes, so too does the incoming light from the window. The tree located outside the window appears to move from left to right as it comes in and out of the field of view.

5.3.3 Modeling Artificial Light Sources

I model *artificial light* sources as a physically based light that casts directional rays from within a set boundary. The interactive editor allows a user to create an area light and specify the light direction with a simple click and drag of a mouse. This interaction creates geometric disk representing the light. When rendering an artificial light, the lighting function $L(x \rightarrow \theta)$ becomes similar to Warn's [125] definition of a directional spotlight. For simplicity, the direction of the light is defined by the surface normal of the light's geometric representation. The light has three user defined parameters: an angle β that defines a cone of illumination, an RGB intensity parameter S, and an exponential parameter which can be used to decrease illumination as the cosine of the angle between the spotlight direction and object point. This parameter is normally used to simulate a falloff effect, and is omitted in this formulation. If the angle formed by the ray from x to the camera viewpoint θ and the light direction is greater than β, the viewpoint is considered to be outside of the cone of illumination.

Once a light is defined in the geometry, I use an optimization technique with a small amount of user interaction to estimate the intensity and spotlight cutoff parameters. The user selects two points on a surface (with homogeneous material properties) partially lit by the spotlight. One point is in direct light, and the second point is unlit (Figure 5.9). If the surface points have the same material properties and are Lambertian, that is the albedo is identical at both locations, I can estimate the spotlight intensity by computing the difference in radiance arriving at each point.

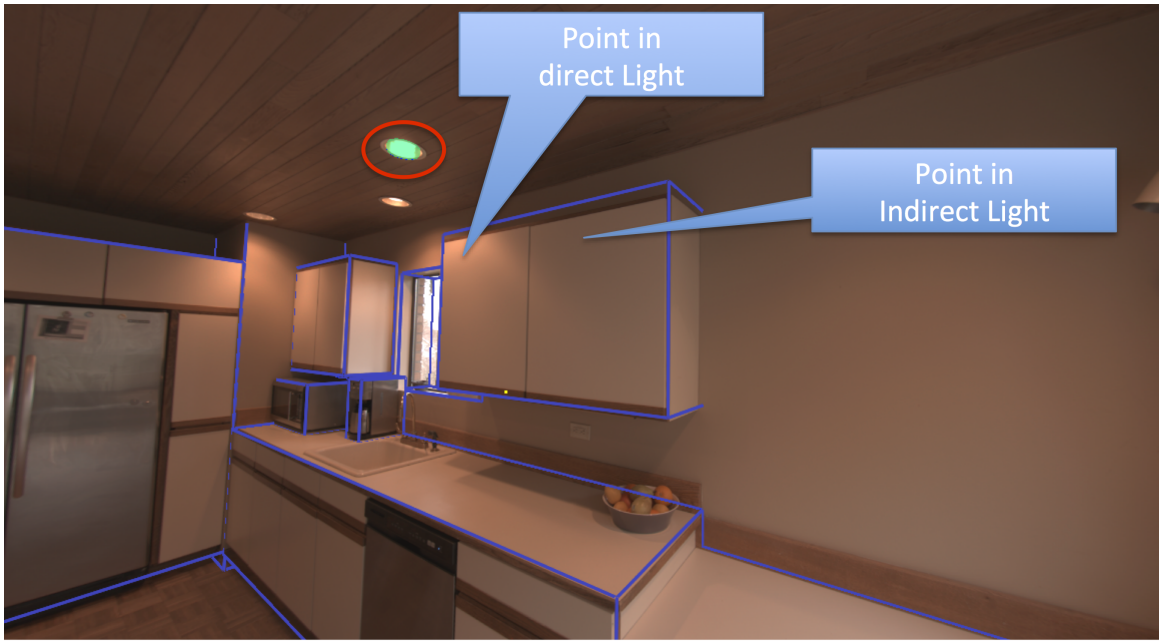


Figure 5.9: In order to estimate the value of the direct light circled in red, and user selects a point directly lit by this light, and an unlit point of the same material. The spotlight intensity can be calculated by the difference in irradiance intensities compared to the observed values in the image.

The goal is to estimate the direct lighting parameters S and β. The first step is to estimate the light intensity S. Using the notation in Appendix A, $L(x \leftarrow \Theta)$ is an observation from the viewpoint x with a viewing direction of Θ. The reflected light from a point on a reflective surface x is represented by $L(x \rightarrow \Theta)$. Since our reflection function $f_r(x, \Psi \rightarrow \Theta)$ for incident direction Ψ is Lambertian, and I are only concerned with identifying parameters for one light source I can simplify the problem to simultaneously solving for the Lambertian term k_d and a single light intensity. I can classify the light sources into *Direct* and *Indirect*

components, and further classify these components into known and unknown quantities. In this example, I know the *Indirect* components and want to determine a single unknown *Direct* light. Since our reflectance function is Lambertion, breaking light sources into *Direct* and *Indirect* becomes much simpler. First, let's separate the reflectance function out of the incident lighting integral.

$$L(x \to \Theta) = \int_{\Omega_x} f_r(x, \Psi \to \Theta)\, L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \tag{5.3a}$$

$$\text{Let } f_r(x, \Psi \to \Theta) \text{ be Lambertian} \tag{5.3b}$$

$$f_r(x, \Psi \to \Theta) = k_d \tag{5.3c}$$

$$L(x \to \Theta) = \int_{\Omega_x} k_d\, L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \tag{5.3d}$$

$$L(x \to \Theta) = k_d \int_{\Omega_x} L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \tag{5.3e}$$

$$\text{Let } E(x) = \int_{\Omega_x} L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \tag{5.3f}$$

$$L(x \to \Theta) = k_d\ E(x) \tag{5.3g}$$

Now I can more easily separate *Direct* and *Indirect* lighting terms.

$$L(x \to \Theta) = k_d \int_{\Omega_x} L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \tag{5.4a}$$

$$= L_{direct}(x \to \Theta) + L_{indirect}(x \to \Theta) \tag{5.4b}$$

$$L_{direct}(x \to \Theta) = k_d \int_{\Omega_x} L_e(y \to \vec{yx})\, V(x, y)\, cos(N_x, \vec{yx})\, d\omega_y \tag{5.4c}$$

$$L_{indirect}(x \to \Theta) = k_d \int_{\Omega_x} L_i(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \tag{5.4d}$$

$$L(x \to \Theta) = k_d(E_{direct}(x) + E_{indirect}(x)) \tag{5.4e}$$

I now have three unknowns k_d, our lighting function $L_e(y \to \vec{yx})$ describing the direct light arriving at location x from location y, and a visibility function $V(x, y)$, where $V(x, y)$ is the visibility between point x and light source point y expressed as a Kronecker delta. To

further simplify, since only one of the points (x_d) lies in direct light I can solve for k_d directly by using the point indirect light x_i. I can use a hemicube to calculate irradiance both with and without direct light. The direct light calculation includes the intensity parameter S for each point x_d and x_i. I denote the irradiance for direct light as $E_{direct}(x, S)$ and indirect as $E_{indirect}(x)$. $L(x \rightarrow \Theta)$ is given by pixel values, assuming minimal camera sensor and lens error. The visibility function $V(x, y)$ is calculated during the hemicube evaluation. This leads a solution for the spot light intensity parameter S.

$$k_d = \frac{L(x_i \rightarrow \Theta)}{E_{indirect}(x_i)} = \frac{L(x_d \rightarrow \Theta)}{E_{indirect}(x_d) + E_{direct}(x_d, S)} \tag{5.5}$$

If I have several points in both direct and indirect light or choose to use multiple viewpoints Θ, I can solve for k_d, and then S by solving a larger linear system.

The next step is to determine the cutoff angle β for the directional light parameter. This process is a minimization using the previously calculated k_d and light intensity S. I evaluate the lighting model from points (M) sampled along a line connecting the two user selected points (Figure 5.10). This sampling insures that I cross the lighting cone boundary. The hemicube irradiance estimate is updated to include the cutoff angle β, $E(x, S, \beta)$. The goal is to find the angle β that best explains the change in lighting in the 3D samples. I seek to minimize:

$$\underset{\beta}{\operatorname{argmin}} \sum_{x \, \varepsilon \, M} \left\| \frac{L(x_i \rightarrow \Theta)}{E(x, S, \beta)} - k_d \right\| \tag{5.6a}$$

using a simple search over the range of β.

5.4 Determining Material Property Values

After the illumination information is determined, I have a good estimate for the natural and artificial light sources as well as texture for diffuse light and Lambertian materials. The material properties for Lambertian surfaces trivally fall out of equation 5.1. However, scenes such as the one depicted in Figure 5.6 include specular surfaces. Therefore, relighting these surfaces requires estimating their specular reflectance function (material properties). I can

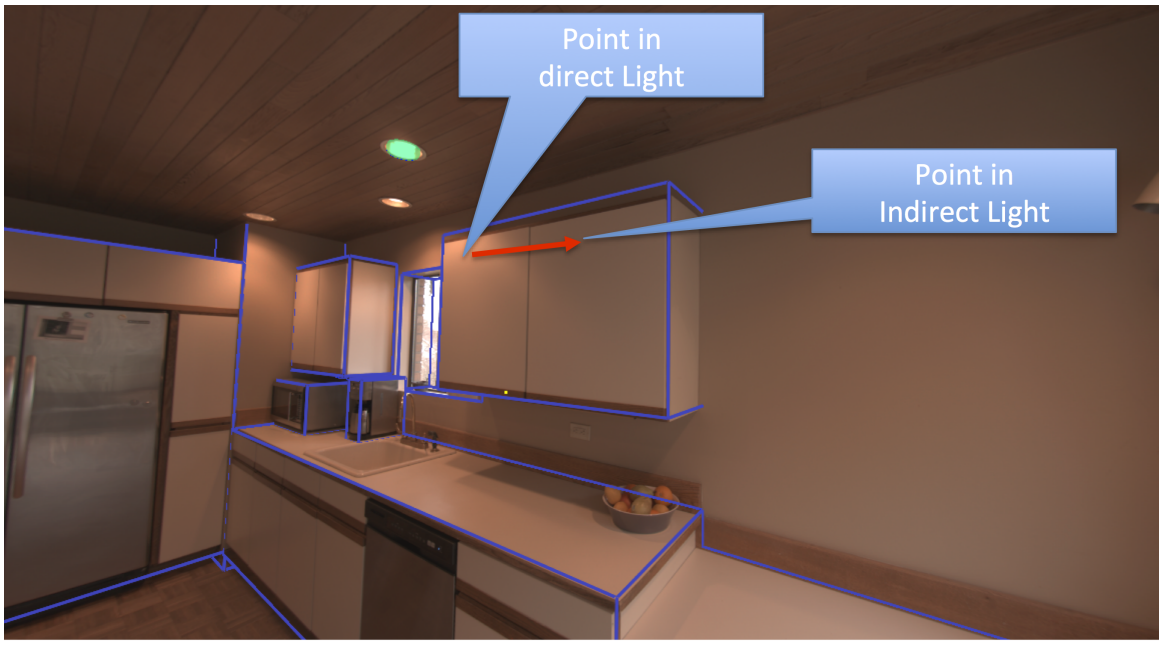


Figure 5.10: In order to estimate the spotlight cutoff parameter, I choose the cutoff angle β that minimizes the difference between and estimated albedo and the one calculated using the lighting calculation along each point along the 3D line between a point in direct light and one in indirect lighting.

substitute the estimated material properties for texture when rendering these surfaces for a realistic appearance.

To determine material property values I take inspiration from Yu et al. [128] albeit using a simpler shading model. A user selects an area of the same material (e.g., an area of hardwood flooring) using an interactive interface. Like Yu et al., I also avoid the hard problem of automatic image segmentation. Next, I automatically sample points from the selected area that are visible from different viewpoints. This process assumes accurate geometry and is problematic for highly textured surfaces. Then, for each selected area, I determine the value of the parameters in the Phong [97] or Ward [124] shading model $f_r(x, \Psi \rightarrow \Theta)$. These values are determined using a non-linear solver to minimize error between lighting values calculated using the lighting models and colors seen in the original unedited scene. I use the lighting parameters $L(x \leftarrow \Psi)$ previously estimated with our hemicube formulation.

Figure 5.11: Left: An original photo with the direction light to be estimated circled in red. Middle: An image of the irradiance for each point in the image. This image only considers the diffuse light generated by the VITA. Notice the lack of light on the cabinet. Right: The effect of the the estimated spotlight on the irradiance image.

In initial experiments I used a Phong shading model, calculating the diffuse and specular components k_d and k_s, and the shininess coefficient n, and a Levenberg Marquardt nonlinear least squares solver [82]. The results of a simple experiment can be seen in Figure 5.12.

$$\underset{\rho}{\operatorname{argmin}} \sum_{x \, \varepsilon \, M} \left| L(x \rightarrow \Theta) - \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta) L(x \leftarrow \Psi) \, cos(N_x, \Psi) d\omega_\Psi \right| \tag{5.7}$$

where

$$f_r(x, \Psi \rightarrow \Theta) = k_s \frac{(R \cdot \Theta)^n}{N \cdot \Psi} + k_d$$

and

$$\rho = k_s, k_d, n$$

In practice, the specular component k_s can be simplified to a single intensity scalar rather than a full RGB vector. In all I solved for five variables: the specular k_s component, the diffuse k_d RGB vector, and exponential term n.

Figure 5.12: Left: An original photo. Right: An estimated specular floor using a hemicube irradiance estimate calculated with only indirect light and a single image. In this example I estimated a Phong shading model, calculating the diffuse and specular components k_d and k_s, and the shininess coefficient n. The simple estimate does replicate the detailed texture of the original floor using a per pixel albedo estimate. Note that the coffee table is not modeled and thus the legs disappear when rendering along with its shadow. The floor is re-rendered using a coarse grained hemicube interpolation, thus producing a slighting grainy and blurry effect due to sampling error. This example illustrates the importance of having both detailed geometry and a full lighting model including LRIs in order to produce accurate results.

5.5 Rendering Remodeled Image-Based Geometry and Synthetic Objects

The light source models and material properties are used to light any synthetic objects inserted into the scene, and to relight the edited scene. Synthetic objects are realistically lit by calculating irradiance values (using hemicubes) for the synthetic objects.

Rendering the scene can be divided into a three step process. Similar to the Debevec differential rendering technique [30], I modulate the original image texture by the change in lighting. If I assume the model is Lambertian, then relighting is simply multiplying a pixel by the ratio of its irradiance before and after edits as in Figure 5.7. Let's revisit the lighting equation, substituting Lambertian reflectance for a general function, and further

substituting a hemicube lighting evaluation for the entire lighting integral. The light at any pixel location is the product of the diffuse coefficient and the hemicube light integration.

$$\begin{aligned} L(x \rightarrow \Theta) &= \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta) L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \\ &= k_d \int_{\Omega_x} L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \\ &= k_d H(x) \end{aligned}$$

Let x indicate a position in the unedited scene, and x' be x's edited position and corresponding environment. The corresponding pixel color $L(x' \rightarrow \Theta)$ becomes:

$$\begin{aligned} \frac{L'(x' \rightarrow \Theta)}{L(x \rightarrow \Theta)} &= \frac{k_d H'(x')}{k_d H(x)} \\ L'(x' \rightarrow \Theta) &= \frac{H'(x')}{H(x)} L(x \rightarrow \Theta) \end{aligned}$$

The first step in the rendering process is to determine the irradiance in the scene at every pixel before any edits. This is easily accomplished by using a hemicube approximation. I render a hemicube using the original unedited scene geometry and position for every pixel in our image. Even if a piece of geometry has been moved, the light information is from its original position and orientation. For example, if a wall has been moved, I need the irradiance from its un-edited position. Obviously, I don't want to store the hemicube at each pixel location, but I can store the relevant results such as the irradiance values, any specular lighting components and texture values. The result of this first step is an image comprising of irradiance values before and after the edit, the RGB texture values, and any specular reflection components if any. The second step is to estimate the irradiance for every pixel in the edited scene. Using the textures and lights of an edited scene, I have an approximate solution for the irradiance in the edited scene. I re-calculate and store each of the light integrals. Finally, I update rendered texture with the irradiance ratio.

Note that this description assumes that the lighting change due to the edit is minor and does not effect the VITA texture. If the edit dramatically changes the radiance, the VITA

texture for the edited scene must be updated to reflect the change in light due to the edit. Given our assumption that the VITA texture is Lambertian, I iteratively update the VITA texture by the irradiance ratio from the edit. This can be accomplished quickly by sampling each texture and updating the samples. The results are interpolated across the entire texture. More specifically, each face is sampled at a number of 3D locations. Each sample location is evaluated for an irradiance ratio. The samples are triangulated via Delaunay triangulation [56] to create an interpolation mesh over the texture. The irradiance ratio is interpolated over the mesh and the texture updated with the interpolated values. The irradiance ratio converges to 1.0 after a few iterations. Once the VITA texture is updated, I can then use it to calculate the irradiance for the edited scene geometry.

Figure 5.13: Three views of a synthetic object with specular highlights. Note how the figure casts shadows on the walls. Both the synthetic object and the floor have specular material properties, the synthetic object's are user specified while the floor uses an estimated value.

At this point I can incorporate material properties with specular highlights, and therefore synthetic objects into the scene. Now the hemicube approximation must not only account for lighting models, but material models. Further, when rendering an object with a non-Lambertian material property, evaluating the reflectance function necessitates another hemicube evaluation as the material calculation must determine the incoming light at its location. I am not interested in solving a radiosity solution, but rather a double bounce approximation more akin to ray tracing. Care must be taken to avoid deeply nested recursive hemicube evaluations. Fortunately the process is highly parallel and can be formulated

to take full advantage of all available CPUs on a cluster of machines. Using a differential rendering paradigm, I can easily render synthetic objects that cast shadows and interact with the surrounding environment. Unlike Debevec, I do not have to assume that all distant objects are photometrically isolated. I only assume the LRI's are isolated, therefore I do not need to explicitly model a local area that defines a lighting interaction (see Section 2.3.4). Figure 5.14 shows the light interaction with a synthetic cabinet with the surrounding area.

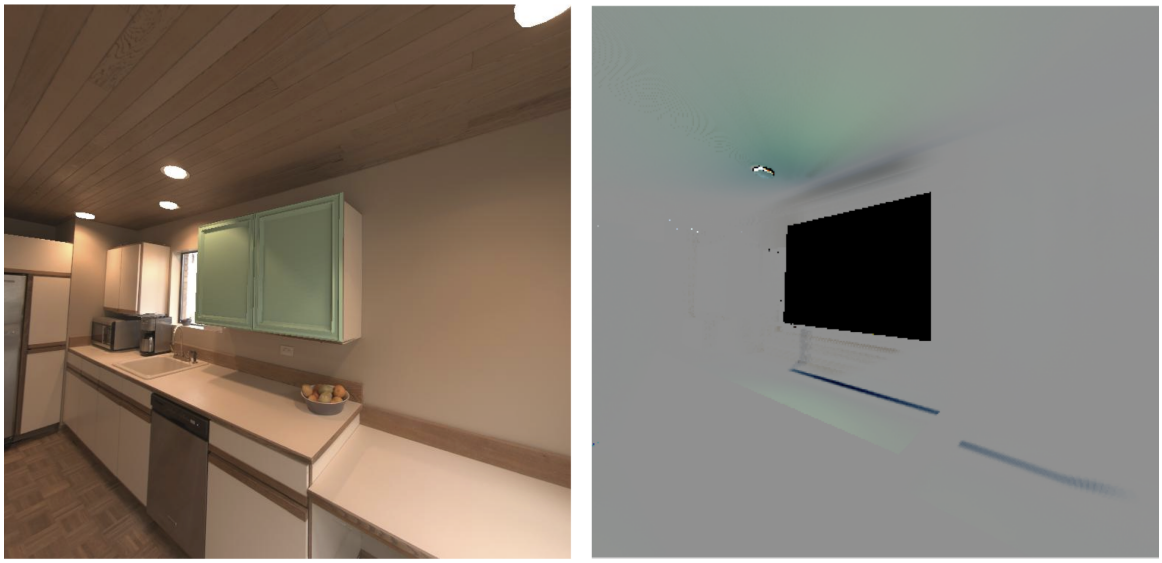


Figure 5.14: Left: The rendered scene with a synthetic cabinet. Right: An irradiance ratio image, exaggerated to show secondary lighting effects. Gray values indicate no change. The green tinge along the ceiling and countertop show the reflected light from the new cabinet door. The slightly larger synthetic cabinets cast proportionally larger shadows onto the ceiling and countertops. The enlarged shadow area is illustrated by the darker diagonal lines in the ratio image.

In addition to simulating how edits and synthetic objects might appear in an actual photograph of an interior, there I also support limited interactive editing of material properties. While a complete re-rendering and light evaluation is expensive, storing the results of the incident light integral is easily accomplished. Ignoring secondary effects, such as reflections onto surrounding surfaces, editing a material parameter outside of the integral such as diffuse k_d or the specular k_s can be done interactively. For example, one might choose to edit the diffuse component of a highly reflective cabinet to choose the best color in a particular

Figure 5.15: Interactive material editing can be accomplished by using stored lighting integrals. In this example, the cabinet door has a Phong material property, although any lighting model could be used. Both the diffuse and specular coefficients lie outside of the lighting integral and can be edited. The specular parameter does not, and cannot be interactively edited in this case. Note there are no secondary bounce effects such as the color of the cabinet reflecting from the ceiling.

lighting environment.

A preliminary implementation designed to show the feasibility of the approach results in effective visualizations of lighting edits.

Using the approach I've outlined in this chapter, I can effectively model direct lights such as conical spotlights, and capture and utilize natural light sources such light streaming in from a window. Our light model now enables rendering realistic synthetic objects in the scene, complete with inter-reflections and shadows with existing geometry using Lambertian ratios. If I estimate specular material properties for geometric objects modeled in the scene, I can also use these estimates while rendering. It is now possible re-render the scene and capture changes in lighting due to edits such as a new window.

One obvious area for improvement would be to refine the geometric model for lighting

purposes. The course model reveals lighting artifacts for un-modeled objects that would not be apparent from the photographs.

Figure 5.16: Top right: A synthetic cabinet door with a rail profile simulating that of a commercial product. This is rendered without the spotlight modeled, note how the spotlight pattern does not resemble the original texture to the left. Bottom right: The same model using estimated spotlight parameters.

Figure 5.17: Top Left: Original photo. Top Right, Bottom row: Synthetic stool, notice the shadows on the walls and floor. The glossy floor was not modeled so there is no reflection of the stool in the floor.

Chapter 6
CONCLUSIONS AND FUTURE WORK

In this thesis, I have described novel approaches to creating, visualizing, and editing image-based models, and demonstrated how these approaches can be used to create immersive image-based model viewing experiences, and more importantly provided a foundation for editing image-based models in an interactive environment.

6.1 Contributions

To develop these approaches, I began by identifying the advantages and limitations of traditional image-based geometry techniques. I designed new representations, computational, and interaction techniques to overcome a number of the limitations of previous techniques and provided a pathway towards creating image-based geometric models that are as editable as traditional geometry while retaining the visual fidelity of the original photographic images.

I applied these approaches in three chapters. In chapter 3, along with my coauthor Colin Zheng, I addressed the challenge of visualizing image-based geometry in a manner that is both visually pleasing and enables story telling. We developed a GPU-accelerated, image-based fast rendering algorithm that enabled the creation of such visualizations in the form of camera paths and cinematic effects commonly used by cinematographers. To avoid objectionable rendering artifacts (such as occlusion holes), we used the fast rendering algorithm to quickly sample the parameter space of camera viewpoints and define a viable region of camera parameters. This region was subsequently used to constrain an optimization for a camera path that maximized parallax and conformed to cinematic conventions. This rendering algorithm also allowed users to create more complex camera paths interactively, while experimenting with effects such as focal length, depth of field, and selective, depth-based desaturation or brightening. We presented an automatic, content-aware approach to

applying cinematic conventions to an input of a few photographs.

In chapter 4, I tackled the challenge of editing image-based geometry. To move image-based models away from the limitations of view-only geometry to the broader class of editable geometry, I describes a new approach for creating and visualizing editable image-based models in a photorealistic manner. Using this approach, I presented an interactive system that models, and then remodels image-based geometry in the context of home interior architecture. This system supported creation of concise, parameterized, and constrained geometry, as well as remodeling directly from within the photographs. Real-time texturing of modified geometry was made possible by precomputing view-dependent textures for all of the faces that are potentially visible to each original camera viewpoint, blending multiple viewpoints, and hole-filling where necessary. The resulting textures were stored and accessed efficiently, enabling intuitive real-time realistic visualization, modeling, and editing of the building interior.

Finally, in chapter 5, I demonstrated how to extend image-based remodeling systems to enable lighting effects. Using a combination of the view-dependent texture created in the image-based remodeling system and radiosity form-factor computations, I can estimate the irradiance at any point in the my model. Using this approach, I can further estimate light sources with the aid of some user annotation. Further, I can estimate reflectance properties for non-Lambertian objects. Armed with a good estimate of the light, I am able to render the lighting changes caused by model edits, and insert synthetic objects into the scene with realistic lighting.

6.2 Future Work

Aside from more fully developing the relighting and synthetic object rendering, there are several interesting directions to push this research. As noted earlier, there are several outstanding limitations:

- The geometry is coarse, imprecise, and contains some occlusion and alignment errors.
- The material properties of the geometry are unknown and may be non-Lambertian.
- The materials properties are not segmented and may vary over the span of a geometric face.

The first limitation of course geometric models may be ameliorated by using better capture techniques, dense reconstruction, and hardware such as depth cameras. However, the problem remains as to how to best model the geometry so it can be edited.

In regards to estimating the material properties, I have shown that once the light in the scene is estimated, and a user manually segments an area with the same material, existing techniques such as Yu et al.'s Inverse global illumination [128] can be used to estimate the material. This process of manual segmentation is tedious and remains dependent upon accurate geometry. It is an interesting problem to automatically estimate material properties given these to limitations.

6.2.1 Interaction

There are two interesting navigation related problems, one during model creation and the other during visualization of the image-based model. When interactively creating a model, a user must navigate through the partially built model to select images or regions of the point cloud to work on. Arikan's [3] system forces the user to directly edit on the point cloud. The user selects a plane on the point cloud, and the system automatically positions the user's viewpoint to be centered perpendicularly to the plane for further editing. In contrast, the *Image-based remodeling* [24] system avoids editing directly on point clouds under the assumption that point clouds are too difficult to comprehend. It would be useful to learn more about how users interact with point clouds and images. In particular, determining which operations are easier to perform on point clouds versus photographs, and how to seamlessly transition between the editing modes with a well designed user study might lead to better modeling interfaces and user experience.

6.2.2 Navigation

In regard to visualizing image-based models, it would be useful to study fly-though navigation of architectural structures to better understand the important aspects of a fly-though. It is important to understand exactly which factors convey a sense of layout. Another potential area of study is learning good viewpoints from examples photographs of home interiors.

6.2.3 Plenoptic texture synthesis

One outstanding, but interesting problem is texture reshaping and hole filling in the context of image-based models. While there is a considerable amount of work on texture synthesis, very little of this work uses multiple images or attempts to fill holes in images where there are known correspondences. An interesting problem to explore is hole filling in a manner consistent with multiple viewpoints. A related problem is propagating geometric changes in an image-based model to the associated view-dependent texture in a plausible manner. Finally, it would be interesting to generalize texture transfer to image-based models.

6.3 Conclusion

Image-based models far exceed traditional geometric models in terms of realism. However the ability to view, manipulate, and edit them is in its infancy. While there is great utility in creating visually beautiful but ultimately un-editable models, it would be a shame to let such models languish in an immutable world of display-only geometry. In addition to finding solutions for the immediate problems, I believe this area provides a rich vein of future research. Our ever-improving capacity to capture images and ability to create geometry from them only makes the need for interactive editing tools more important. By focusing on the goals of creating, viewing, and editing image-based geometry, and bringing these capabilities up to the level of traditional geometric, and perhaps image editing tools, I hope to exploit the realism and simplicity image-based geometry affords. It is my belief that such tools can improve the ability of both amateurs and professionals to work more easily with this sort of geometry, not only for editing but also for other tasks that involve understanding and communicating properties of real world objects, and transforming those properties in ways that we have only begun to explore.

Appendix A

RADIOMETRY

A.1 Definitions

Dutre et al. [35] describe many useful terms and relationships. For convenience, some of these terms are reproduced and adapted in this section.

A.1.1 Radiant Power or Flux

The fundamental radiometric quantity is radiant power, also called flux. Radiant power, often denoted as Φ, is expressed in watts $(W)(joules/sec)$.

A.1.2 Irradiance

Irradiance (E), also called (I), is the incident radiant power on a surface, per unit surface area. It is expressed in $watts/m^2$:

$$E = I = \frac{d\Phi}{dA} \tag{A.1}$$

A.1.3 Radiant Exitance or Radiosity

Radiant exitance (M), also called radiosity (B), is the exitant radiant power per unit surface area and is also expressed in $watts/m^2$:

$$M = B = \frac{d\Phi}{dA} \tag{A.2}$$

A.1.4 Radiance

Radiance is flux per unit projected area per unit solid angle $(\frac{watts}{steradian \cdot m^2})$. Radiance expresses how much power arrives at (or leaves from) a certain point on a surface, per unit solid angle, and per unit projected area. Radiance is a five-dimensional quantity that varies

with position x and direction vector Θ, and is expressed as $L(x, \Theta)$. $L(x \rightarrow \Theta)$ represents the radiance leaving at point x from direction Θ. Whereas $L(x \leftarrow \Theta)$ represents the radiance arriving at point x.

$$L = \frac{d^2\Phi}{d\omega dA^{\perp}} = \frac{d^2\Phi}{d\omega dA\, cos(\theta)} \tag{A.3}$$

The projected area $A^{\perp}$ is the area of the surface projected perpendicular to the direction in which we are interested, and θ is the angle between Θ and the normal at point x.

A.1.5 Relationships between Radiometric Quantities

Given the definitions of the radiometric quantities above, the following relationships between these different terms can be derived:

$$\Phi = \int_A \int_\Omega L(x \rightarrow \Theta)\, cos(\theta) d\omega_\Theta dA_x \tag{A.4}$$

$$E(x) = \int_\Omega L(x \leftarrow \Theta)\, cos(\theta) d\omega_\theta \tag{A.5}$$

$$B(x) = \int_\Omega L(x \rightarrow \Theta)\, cos(\theta) d\omega_\theta \tag{A.6}$$

where A is the total surface area and Ω is the total hemispherical solid angle at each point on the surface. $L(x \rightarrow \Theta)$ represents radiance leaving point x in direction Θ. $L(x \leftarrow \Theta)$ represents radiance arriving at point x from direction Θ.

A.1.6 Diffuse emitter

By definition, a diffuse emitter emits equal radiance in all directions from all its surface points. Therefore, $L(x \rightarrow \Theta) = L$. The relationship between the power, radiance, and radiosity of a diffuse surface is:

$$\Phi = LA\pi = BA \tag{A.7}$$

A.1.7 Bidirectional reflectance distribution function (BRDF)

Ignoring effects such as fluorescence, phosphorescence, and subsurface scattering, the reflectance properties of a surface are described by a reflectance function called the bidirectional reflectance distribution function. The BRDF at a point x is defined as the ratio of the differential radiance reflected in an exitant direction (Θ), and the differential irradiance incident through a differential solid angle ($d\omega_\Psi$). The BRDF is denoted as $f_r(x, \Psi \rightarrow \Theta)$:

$$f_r(x, \Psi \rightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} \tag{A.8a}$$

$$= \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\, cos(N_x, \Psi)\, d\omega_\Psi} \tag{A.8b}$$

$$= \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\, N_x \cdot \Psi\, d\omega_\Psi} \tag{A.8c}$$

where $cos(N_x, \Psi)$ is the cosine of the angle formed by the normal vector at the point x, N_x, and the incident direction vector Ψ.

A.1.8 Albedo

The BRDF of a diffuse surface can be defined as:

$$f_r(x, \Psi \rightarrow \Theta) = \frac{\rho_d}{\pi} \tag{A.9}$$

The reflectance ρ_d represents the fraction of incident energy that is reflected at a surface. This term is also called albedo.

A.1.9 Diffuse Shading Model

A Lambertian shading model where Ψ represents the light source, and Θ is the direction of the viewer is defined as:

$$f_r(x, \Psi \rightarrow \Theta) = k_d = \frac{\rho_d}{\pi} \tag{A.10}$$

where k_d is the diffuse coefficient of reflection.

A.1.10 Phong Shading Model

The Phong shading model is defined as:

$$f_r(x, \Psi \rightarrow \Theta) = k_s \frac{(R \cdot \Theta)^n}{N_x \cdot \Psi} + k_d \tag{A.11}$$

where k_s is the specular coefficient of reflection, and n is the shininess coefficient. For a perfect reflector n equals infinity.

A.1.11 Ward Shading Model

The Ward model [124] is defined as:

$$f_r(x, \Psi \rightarrow \Theta) = \frac{\rho_d}{\pi} + \rho_s \frac{e^{\frac{-tan^2 \theta h}{\alpha^2}}}{4\pi\alpha^2\sqrt{(N_x \cdot \Psi)(N_x \cdot \Theta)}} \tag{A.12}$$

where θh is the angle between the half-vector and the normal, ρ_d, the diffuse reflectance, ρ_s the specular reflectance, and α, a measure of the surface roughness. This model is energy conserving unlike the Phong model.

A.1.12 Rendering

Define $L_e(x \rightarrow \Theta)$ as the radiance emitted by the surface at x and in the outgoing direction Θ, and $L_r(x \rightarrow \Theta)$ the reflected radiance. By conservation of energy:

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta) \tag{A.13}$$

Radiance is the sum of all of the emitted and reflected light in the scene. If we integrate light sources from direction Ψ over the hemisphere Ω_x at surface point x, radiance can be summarized by the following:

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \rightarrow \Theta) L(x \leftarrow \Psi)\, cos(N_x, \Psi) d\omega_\Psi \tag{A.14}$$

BIBLIOGRAPHY

[1] Adelson, E. H., and Bergen, J. R. The Plenoptic Function and the Elements of Early Vision. In *Computational Models of Visual Processing*, M. Landy and J. Movshon, Eds. Computational Models of Visual Processing, Oct. 1991, pp. 3–20. (Cited on page iv.)

[2] Agarwala, A. *Authoring effective depictions of reality by combining multiple samples of the plenoptic function.* PhD thesis, University of Washington, 2006. (Cited on page iv.)

[3] Arikan, M., Schwärzler, M., Flöry, S., Wimmer, M., and Maierhofer, S. O-Snap: Optimization-Based Snapping for Modeling Architecture. *ACM Transactions on Graphics (TOG) to appear* (Sept. 2012), (Cited on pages 10 and 114.)

[4] Artusi, A., Banterle, F., and Chetverikov, D. A Survey of Specularity Removal Methods. *Computer Graphics Forum 30*, 8 (Aug. 2011), 2208–2230. (Cited on page 23.)

[5] Autodesk. Autodesk Homestyler. (Cited on page 6.)

[6] Autodesk. Autodesk 123D Catch. *http://www.123dapp.com/catch* (2011). (Cited on page 2.)

[7] Autodesk. Autodesk Revit. *http://usa.autodesk.com/revit* (2011). (Cited on page 8.)

[8] Bahmutov, G., Popescu, V., and Mudure, M. Efficient Large Scale Acquisition of Building Interiors. *Computer Graphics Forum 25*, 3 (2006), 655–662. (Cited on pages 15 and 66.)

[9] Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. B. PatchMatch: a randomized correspondence algorithm for structural image editing. *SIGGRAPH '09: ACM Trans. Graph 28*, 3 (2009), 24. (Cited on page 29.)

[10] Barrow, H. G., and Tenenbaum, J. M. Recovering intrinsic scene characteristics from images. Tech. Rep. Technical Note 157, 1978. (Cited on page 22.)

[11] Belhumeur, P. N., Kriegman, D. J., and Yuille, A. L. The bas-relief ambiguity. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on* (1997). (Cited on page 21.)

[12] Bhat, P., Zitnick, C. L., Cohen, M. F., and Curless, B. GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering. In *Proceedings* (New York, NY, USA, 2009), ACM, pp. 10:1–10:14. (Cited on page 79.)

[13] Blake, A. Boundary conditions for lightness computation in Mondrian World. *Computer Vision, Graphics, and Image Processing 32*, 3 (Dec. 1985), 314–327. (Cited on page 22.)

[14] Bousseau, A., Paris, S., and Durand, F. User-assisted intrinsic images. *SIGGRAPH Asia '09: ACM Trans. Graph.* (Dec. 2009). (Cited on page 23.)

[15] Brelstaff, G., and Blake, A. Computing lightness. *Pattern Recognition Letters 5*, 2 (1987), 129–138. (Cited on page 22.)

[16] Buchholz, H., and Dollner, J. View-dependent rendering of multiresolution texture-atlases. *Visualization, 2005. VIS 05. IEEE 0* (2005), 215–222. (Cited on page 77.)

[17] Buehler, C., Bosse, M., McMillan, L., Gortler, S. J., and Cohen, M. F. Unstructured lumigraph rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, Aug. 2001), New York, NY, USA, ACM, pp. 425–432. (Cited on pages 8, 34, 35, 48, and 78.)

[18] Carroll, R., Agarwala, A., and Agrawala, M. Image warps for artistic perspective manipulation. In *SIGGRAPH '10: ACM Trans. Graph.* (New York, NY, USA, 2010), ACM, pp. 127:1–127:9. (Cited on page 67.)

[19] Catto, E. Box2d. *http://box2d.org* (2011). (Cited on page 29.)

[20] Chen, B., Ofek, E., Shum, H.-Y., and Levoy, M. Interactive deformation of light fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM, pp. 139–146. (Cited on pages 27 and 67.)

[21] Choudhury, B., Chandran, S., and Herder, J. A survey of image-based relighting techniques. *Journal of Virtual Reality and Broadcasting 4*, 7 (2007). (Cited on page 24.)

[22] Cipolla, R., and Robertson, D. 3D Models of Architectural Scenes from Uncalibrated Images and Vanishing Points. In *Proceedings of the 10th International Conference on Image Analysis and Processing* (Washington, DC, USA, Sept. 1999), IEEE Computer Society, pp. 824–829. (Cited on pages 9 and 67.)

[23] Cohen, M. F., and Greenberg, D. P. The hemi-cube: a radiosity solution for complex environments. *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques 19*, 3 (July 1985), 31–40. (Cited on page 92.)

[24] Colburn, A., Agarwala, A., Hertzmann, A., Curless, B., and Cohen, M. F. Image-Based Remodeling. *IEEE Transactions on Visualization and Computer Graphics 99* (2012). (Cited on pages 29, 63, and 114.)

[25] Cornelis, N., Leibe, B., Cornelis, K., and Van Gool, L. J. 3D Urban Scene Modeling Integrating Recognition and Reconstruction. *Int. J. Comput. Vision 78* (July 2008), 121–141. (Cited on pages 12 and 66.)

[26] Criminisi, A. Single-view metrology: Algorithms and applications. *LECTURE NOTES IN COMPUTER SCIENCE* (Jan 2002). (Cited on page 9.)

[27] Criminisi, A., Perez, P., and Toyama, K. Object removal by exemplar-based inpainting. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)* (Jun. 2003). (Cited on page 57.)

[28] Criminisi, A., Reid, I., and Zisserman, A. Single view metrology. *Int. J. Comput. Vision 40* (November 2000), 123–148. (Cited on page 67.)

[29] Curless, B., and Levoy, M. A Volumetric Method for Building Complex Models from Range Images. *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), 303–312. (Cited on page 12.)

[30] Debevec, P. E. 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* (July 1998), ACM, pp. 189–198. (Cited on pages 25, 96, and 104.)

[31] Debevec, P. E., Taylor, C., and Malik, J. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (Aug. 1996), 11–20. (Cited on pages 8, 67, and 73.)

[32] Demers, J. Depth of field a survey of techniques. In *GPU Gems 1*, R. Fernando, Ed. Mar. 2004, ch. 23, pp. 375–390. (Cited on page 54.)

[33] Dick, A., Torr, P. H. S., and Cipolla, R. Modelling and Interpretation of Architecture from Several Images. *Int. J. Comput. Vision 60* (Nov. 2004), 111–134. (Cited on pages 14 and 66.)

[34] Du, H., Henry, P., Ren, X., Cheng, M., Goldman, D. B., Seitz, S. M., and Fox, D. Interactive 3D modeling of indoor environments with a consumer depth camera. In *UbiComp '11: Proceedings of the 13th international conference on Ubiquitous computing* (Sept. 2011), ACM. (Cited on page 16.)

[35] Dutré, P., Bala, K., and Bekaert, P. *Advanced global illumination*, 2 ed. A K Peters Ltd, 2006. (Cited on page 116.)

[36] Eastman, C. M., and Weiss, S. F. Tree structures for high dimensionality nearest neighbor searching. *Information Systems 7*, 2 (1982), 115–122. (Cited on page 72.)

[37] El-Hakim, S., Whiting, E., and Gonzo, L. 3d modeling with reusable and integrated building blocks. In *The 7th Conference on Optical 3-D Measurement Techniques* (Jan. 2005), pp. 3–5. (Cited on pages 10 and 67.)

[38] Felzenszwalb, P. F., and Huttenlocher, D. P. Distance transforms of sampled functions. Tech. Rep. TR2004-1963, Cornell University, 2004. (Cited on page 40.)

[39] Furukawa, Y., Curless, B., Seitz, S. M., and Szeliski, R. Manhattan-world stereo. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), 1422–1429. (Cited on pages 11, 13, and 70.)

[40] Furukawa, Y., Curless, B., Seitz, S. M., and Szeliski, R. Reconstructing building interiors from images. In *Computer Vision, 2009 IEEE 12th International Conference on* (2009), pp. 80–87. (Cited on pages 13, 63, 66, and 82.)

[41] Furukawa, Y., and Ponce, J. Accurate, Dense, and Robust Multi-View Stereopsis. *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on* (June 2007), 1–8. (Cited on page 12.)

[42] Furukawa, Y., and Ponce, J. Patch-based Multi-View Stereo Software. *http://grail.cs.washington.edu/software/pmvs/* (2009). (Cited on page 70.)

[43] Gal, R., Sorkine, O., Mitra, N. J., and Cohen-Or, D. iWIRES: an analyze-and-edit approach to shape manipulation. *SIGGRAPH '09: ACM Trans. Graph 28*, 3 (2009), 33:1–33:10. (Cited on page 29.)

[44] Georghiades, A. S. Recovering 3-D shape and reflectance from a small number of photographs. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (June 2003), Eurographics Association. (Cited on page 21.)

[45] Georgiev, T., Zheng, C., Nayar, S., Salesin, D., Curless, B., and Intwala, C. Spatio-angular resolution trade-offs in integral photography. *Proc. of Eurographics Symposium on Rendering* (2006). (Cited on pages 33 and 59.)

[46] Goesele, M., Snavely, N., Curless, B., Hoppe, H., and Seitz, S. M. Multi-View Stereo for Community Photo Collections. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* (2007), pp. 1–8. (Cited on pages 12 and 66.)

[47] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (Jan. 1984), ACM. (Cited on page 91.)

[48] Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 43–54. (Cited on pages 9, 32, 33, 34, and 35.)

[49] Haber, T., Fuchs, C., Bekaer, P., Seidel, H.-P., Goesele, M., and Lensch, H. P. A. Relighting objects from image collections. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (June 2009), pp. 627–634. (Cited on page 21.)

[50] Hammon, E. Practical post-process depth of field. In *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley, July 2007, ch. 28. (Cited on page 54.)

[51] Hartley, R., and Zisserman, A. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, 2004. (Cited on pages iv and 33.)

[52] He, L.-W., Cohen, M. F., and Salesin, D. H. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *Proc. of SIGGRAPH 96* (Aug. 1996), Computer Graphics Proceedings, Annual Conference Series, pp. 217–224. (Cited on page 35.)

[53] Hedau, V., Hoiem, D., and Forsyth, D. Recovering the Spatial Layout of Cluttered Rooms. *Computer Vision, 2009. ICCV 2009. IEEE International Conference on* (2009), 1849–1856. (Cited on pages 13 and 66.)

[54] Heigl, B., Koch, R., Pollefeys, M., Denzler, J., and Gool, L. J. V. Plenoptic modeling and rendering from image sequences taken by hand-held camera. In *DAGM-Symposium* (1999), pp. 94–101. (Cited on page 34.)

[55] Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research 31*, 5 (2012), 647–663. (Cited on page 16.)

[56] Hert, S., and Seel, M. dD convex hulls and Delaunay triangulations. In *CGAL User and Reference Manual*, 4.3 ed. CGAL Editorial Board, 2013. (Cited on page 106.)

[57] Hoiem, D., Efros, A. A., and Hebert, M. Automatic photo pop-up. *ACM Trans. Graph. 24*, 3 (August 2005), 577–584. (Cited on page 34.)

[58] Hormann, K., Lévy, B., and Sheffer, A. ACM SIGGRAPH 2007 courses on - SIGGRAPH '07. In *ACM SIGGRAPH 2007 courses* (New York, New York, USA, 2007), ACM Press, p. 1. (Cited on page 15.)

[59] Horn, D. R., and Chen, B. LightShop: interactive light field manipulation and rendering. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), ACM, pp. 121–128. (Cited on pages 28 and 67.)

[60] Horry, Y., Anjyo, K.-I., and Arai, K. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proc. of SIGGRAPH 97* (August 1997), pp. 225–232. (Cited on page 34.)

[61] Hughes, C., Denny, P., Jones, E., and Glavin, M. Accuracy of fish-eye lens models. *Applied optics 49*, 17 (2010), 3338–3347. (Cited on page 70.)

[62] Hugo Elias. Radiosity Tutorial. *http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm* (2003). (Cited on page 91.)

[63] Isaksen, A., McMillan, L., and Gortler, S. J. Dynamically reparameterized light fields. (Cited on page 59.)

[64] Johnson, D. S. Approximation algorithms for combinatorial problems*. *Journal of Computer and System Sciences* (1974), 38–49. (Cited on page 74.)

[65] Ju, T., and Ju, T. Robust repair of polygonal models. *ACM Transactions on Graphics (TOG) 23*, 3 (Aug. 2004), 888–895. (Cited on page 14.)

[66] Kang, S. B., and Shum, H.-Y. A review of image-based rendering techniques. In *IEEE/SPIE Visual Communications and Image Processing 2000* (2002), pp. 2–13. (Cited on page 34.)

[67] Karsch, K., Hedau, V., Forsyth, D., and Hoiem, D. Rendering synthetic objects into legacy photographs. In *SIGGRAPH Asia '11: ACM Trans. Graph.* (Dec. 2011), ACM. (Cited on pages 26, 86, and 89.)

[68] Kass, M., Lefohn, A., and Owens, J. D. Interactive depth of field using simulated diffusion. Tech. Rep. 06-01, Pixar Animation Studios, Jan. 2006. (Cited on page 54.)

[69] Kelly, T., and Wonka, P. Interactive architectural modeling with procedural extrusions. *ACM Transactions on Graphics (TOG) 30*, 2 (Apr. 2011), 1–15. (Cited on page 14.)

[70] Klaus, A., Sormann, M., and Karner, K. F. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *International Conference on Pattern Recognition (ICPR)* (2006), pp. 15–18. (Cited on page 45.)

[71] Kushal, A., Self, B., Furukawa, Y., Gallup, D., Hernandez, C., Curless, B., and Seitz, S. M. Photo Tours. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on* (2012). (Cited on page 18.)

[72] Laffont, P.-Y., Bousseau, A., and Drettakis, G. Rich Intrinsic Image Decomposition of Outdoor Scenes from Multiple Views. *IEEE Transactions on Visualization and Computer Graphics* (2012), 1–1. (Cited on page 23.)

[73] Land, E. H., and McCann, J. J. Lightness and retinex theory. *Journal of the Optical society of America 61*, 1 (1971), 1–11. (Cited on page 22.)

[74] Lee, D. C., Gupta, A., Hebert, M., and Kanade, T. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. *Advances in Neural Information Processing Systems (NIPS) 24* (2010), 1288–1296. (Cited on pages 13 and 66.)

[75] Levin, A., Fergus, R., Durand, F., and Freeman, W. Image and depth from a conventional camera with a coded aperture. *ACM Trans. Graph. 26*, 3 (July 2007), 70:1–70:9. (Cited on page 33.)

[76] Levoy, M. Light fields and computational imaging. *IEEE Computer 39*, 8 (2006), 46–55. (Cited on pages 32 and 35.)

[77] Levoy, M., and Hanrahan, P. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (Aug. 1996), ACM. (Cited on pages 9, 32, 34, 35, and 59.)

[78] Liang, C.-K., Lin, T.-H., Wong, B.-Y., Liu, C., and Chen, H. Programmable aperture photography: Multiplexed light field acquisition. *ACM Trans. Graph. 27*, 3 (2008). (Cited on page 33.)

[79] Lipp, M., Wonka, P., and Wimmer, M. Interactive visual editing of grammars for procedural architecture. *SIGGRAPH '08: ACM Trans. Graph.* (Aug. 2008). (Cited on page 14.)

[80] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* (1987). (Cited on page 12.)

[81] Lowe, D. G. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* (1999), pp. 1150–1157. (Cited on page 9.)

[82] Marquardt, D. W. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics 11*, 2 (June 1963), 431–441. (Cited on page 103.)

[83] Marschner, S. R., Westin, S. H., Lafortune, E. P. F., and Torrance, K. E. Image-Based Bidirectional Reflectance Distribution Function Measurement. *Appl. Opt. 39*, 16 (June 2000), 2592–2600. (Cited on page 22.)

[84] Microsoft. Photosynth. *http://www.photosynth.net* (2011). (Cited on page 17.)

[85] Moreno-Noguer, F., Belhumeur, P. N., and Nayar, S. K. Active refocusing of images and videos. *ACM Trans. Graph. 26* (2007), 671–679. (Cited on pages 35 and 58.)

[86] Müller, P., Zeng, G., Wonka, P., and Van Gool, L. J. Image-based procedural modeling of facades. *SIGGRAPH '07: ACM Trans. Graph.* (Aug. 2007). (Cited on pages 14 and 66.)

[87] Nan, L., Sharf, A., Zhang, H., Cohen-Or, D., and Chen, B. SmartBoxes for interactive urban reconstruction. *SIGGRAPH '10: ACM Trans. Graph.* (July 2010). (Cited on pages 11 and 67.)

[88] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. KinectFusion: Real-time dense surface mapping and tracking. *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on* (2011). (Cited on page 16.)

[89] Ng, R. Fourier slice photography. *ACM Trans. Graph. 24*, 3 (Aug. 2005), 735–744. (Cited on page 35.)

[90] Ng, R., Levoy, M., Bredif, M., Duval, G., Horowitz, M., and Hanrahan, P. Light field photography with a hand-held plenoptic camera. Tech. Rep. CSTR 2005-02, Stanford University, 2005. (Cited on page 33.)

[91] Nimeroff, J. S., Simoncelli, E., and Dorsey, J. Efficient re-rendering of naturally illuminated environments. *Eurographics Workshop on Rendering* (1994), 106. (Cited on page 24.)

[92] Nishino, K., Ikeuchi, K., and Zhang, Z. Re-rendering from a sparse set of images. Tech. Rep. Technical Report DU-CS-05-12, 2005. (Cited on page 21.)

[93] Nishino, K., Zhang, Z., and Ikeuchi, K. Determining reflectance parameters and illumination distribution from a sparse set of images for view-dependent image synthesis. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on* (2001), pp. 599–606. (Cited on page 21.)

[94] Oh, B. M., Chen, M., Dorsey, J., and Durand, F. Image-based modeling and photo editing. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (Aug. 2001), ACM. (Cited on pages 28, 34, and 67.)

[95] Perez, P., Gangnet, M., and Blake, A. Poisson image editing. *ACM Transactions on Graphics (TOG)* (2003), 313–318. (Cited on pages 78 and 79.)

[96] Pfister, H., Zwicker, M., van Baar, J., and Gross, M. Surfels: surface elements as rendering primitives. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (July 2000), ACM Press/Addison-Wesley Publishing Co. (Cited on page 16.)

[97] Phong, B. T. Illumination for computer generated pictures. *Communications of the ACM* (1975). (Cited on page 102.)

[98] Pollefeys, M., Van Gool, L. J., Vergauwen, M., Verbiest, F., Cornelis, K., Tops, J., and Koch, R. Visual Modeling with a Hand-Held Camera. *Int. J. Comput. Vision 59*, 3 (Sept. 2004), 207–232. (Cited on pages 12, 33, and 66.)

[99] Pulli, K., Cohen, M. F., Duchamp, T., Hoppe, H., Shapiro, L., and Stuetzle, W. View-based rendering: Visualizing real objects from scanned range and color data. In *Eurographics Rendering Workshop 1997* (June 1997), pp. 23–34. (Cited on pages 33, 35, and 48.)

[100] Ramamoorthi, R., and Hanrahan, P. A signal-processing framework for inverse rendering. *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), 117–128. (Cited on page 20.)

[101] Ramamoorthi, R., and Hanrahan, P. A signal-processing framework for inverse rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 117–128. (Cited on page 86.)

[102] Redfin. Redfin . *http://www.redfin.com* (2011). (Cited on page 1.)

[103] Romeiro, F., and Zickler, T. Blind reflectometry. In *ECCV'10: Proceedings of the 11th European conference on Computer vision: Part I* (Sept. 2010), F. Romeiro and T. Zickler, Eds., Harvard University, 33 Oxford St., Cambridge, MA, USA 02138, Springer-Verlag, pp. 45–58. (Cited on page 86.)

[104] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. Slam++: Simultaneous localisation and mapping at the level of objects. 1352–1359. (Cited on page 16.)

[105] Scharstein, D., and Szeliski, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision 47*, 1-3 (2002), 7–42. (Cited on page 45.)

[106] Seitz, S. M., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)* (June 2006), pp. 519–528. (Cited on page 33.)

[107] Seitz, S. M., and Kutulakos, K. N. Plenoptic image editing. *International Journal of Computer Vision 48*, 2 (2002), 115–129. (Cited on pages 27 and 67.)

[108] Shade, J., Gortler, S. J., He, L.-w., and Szeliski, R. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (July 1998), ACM. (Cited on page 34.)

[109] Shewchuk, J. R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1996, pp. 203–222. (Cited on page 48.)

[110] Siegel, R., and Howell, J. R. *Thermal radiation heat transfer.* Hemisphere Publishing Corp, 1972. (Cited on page 92.)

[111] Sinha, S. N., Steedly, D., Szeliski, R., Agrawala, M., and Pollefeys, M. Interactive 3D architectural modeling from unordered photo collections. *SIGGRAPH Asia '08: ACM Trans. Graph. 27*, 5 (2008), 159. (Cited on pages 10 and 67.)

[112] Snavely, N. Bundler: Structure from Motion for Unordered Image Collections. *http://phototour.cs.washington.edu* (2008). (Cited on pages 9 and 70.)

[113] Snavely, N., Garg, R., Seitz, S. M., and Szeliski, R. Finding paths through the world's photos. *SIGGRAPH '08: ACM Trans. Graph.* (Aug. 2008). (Cited on pages 17 and 82.)

[114] Snavely, N., Seitz, S. M., and Szeliski, R. Photo tourism: exploring photo collections in 3D. *SIGGRAPH '06: ACM Trans. Graph.* (July 2006), 835–846. (Cited on pages 9, 11, 17, 45, 63, 66, and 73.)

[115] Snavely, N., Seitz, S. M., and Szeliski, R. Modeling the world from internet photo collections. *Int. J. Comput. Vision 80*, 2 (2008), 189–210. (Cited on page 9.)

[116] Sutherland, I. E. Sketchpad: a man-machine graphical communication system. In *AFIPS '63 (Spring): Proceedings of the May 21-23, 1963, spring joint computer conference* (May 1963), ACM. (Cited on page 5.)

[117] Talton, J. O., Lou, Y., Lesser, S., Duke, J., Měch, R., and Koltun, V. Metropolis procedural modeling. *ACM Transactions on Graphics (TOG) 30*, 2 (Apr. 2011), 1–14. (Cited on page 14.)

[118] Taylor, D. Virtual camera movement: The way of the future? *American Cinematographer 77*, 9 (Sept. 1996), 93–100. (Cited on page 35.)

[119] Trimble. Sketchup. *http://www.sketchup.com* (2011). (Cited on page 6.)

[120] Unger, J., Gustavson, S., Kronander, J., Larsson, P., Bonnet, G., and Kaiser, G. Next generation image based lighting using HDR video. *SIGGRAPH '11: SIGGRAPH 2011 Talks* (Aug. 2011). (Cited on pages 25 and 89.)

[121] Unger, J., Gustavson, S., Larsson, P., and Ynnerman, A. Free Form Incident Light Fields. *Computer Graphics Forum 27*, 4 (2008), 1293–1301. (Cited on pages 25, 86, and 89.)

[122] van den Hengel, A., Dick, A., Thormählen, T., Ward, B., and Torr, P. H. S. VideoTrace: rapid interactive scene modelling from video. *SIGGRAPH '07: ACM Trans. Graph.* (Aug. 2007). (Cited on pages 9 and 67.)

[123] Viola, P., and Jones, M. Robust real-time object detection. *International Journal of Computer Vision 57* (2004), 137–154. (Cited on page 38.)

[124] Ward, G. J. Measuring and modeling anisotropic reflection. *ACM SIGGRAPH Computer Graphics* (1992). (Cited on pages 20, 102, and 119.)

[125] Warn, D. R. Lighting controls for synthetic images. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (July 1983), ACM. (Cited on page 98.)

[126] Xiao, J., and Furukawa, Y. Reconstructing the World's Museums. *Computer Vision–ECCV 2012 7572*, Chapter 48 (2012), 668–681. (Cited on page 14.)

[127] Yang, Y.-L., Yang, Y.-J., Pottmann, H., and Mitra, N. J. Shape space exploration of constrained meshes. *SIGGRAPH Asia '11: ACM Trans. Graph.* (Dec. 2011). (Cited on page 29.)

[128] Yu, Y., Debevec, P. E., Malik, J., and Hawkins, T. Inverse global illumination: recovering reflectance models of real scenes from photographs. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (July 1999), ACM Press/Addison-Wesley Publishing Co. Request Permissions. (Cited on pages 19, 22, 86, 102, and 114.)

[129] Zheng, K. C., Colburn, A., Agarwala, A., Agrawala, M., Curless, B., Salesin, D., and Cohen, M. A consistent segmentation approach to image-based rendering. Tech. Rep. CSE-09-03-02, University of Washington, 2009. (Cited on pages 34 and 45.)

[130] Zheng, K. C., Colburn, A., Agarwala, A., Agrawala, M., Salesin, D. H., Curless, B., and Cohen, M. F. Parallax photography: creating 3D cinematic effects from stills. *GI '09: Proceedings of Graphics Interface 2009* (May 2009). (Cited on page 31.)

[131] Zheng, Y., Chen, X., Cheng, M.-M., Zhou, K., Hu, S.-M., and Mitra, N. J. Interactive images: cuboid proxies for smart image manipulation. *SIGGRAPH '12: ACM Trans. Graph. 31*, 4 (July 2012). (Cited on page 28.)

[132] Zheng, Y., Fu, H., Cohen-Or, D., Au, O. K.-C., and Tai, C.-L. Component-wise Controllers for Structure-Preserving Shape Manipulation. *Computer Graphics Forum 30*, 2 (Apr. 2011), 563–572. (Cited on page 29.)

[133] Zitnick, C. L., Kang, S. B., Uyttendaele, M., Winder, S., and Szeliski, R. High-quality video view interpolation using a layered representation. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 600–608. (Cited on page 35.)