UNIVERSITY OF CALIFORNIA
Santa Barbara

Towards Anywhere Augmentation

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Stephen DiVerdi

Committee in Charge:

Tobias Höllerer, Ph. D., Chair

Matthew Turk, Ph. D.

Yuan-Fang Wang, Ph. D.

George Legrady, M. F. A.

September 2007

The Dissertation of
Stephen DiVerdi is approved:

Matthew Turk, Ph. D.

Yuan-Fang Wang, Ph. D.

George Legrady, M. F. A.

Tobias Höllerer, Ph. D., Committee Chairperson

June 2007

Towards Anywhere Augmentation

by

Stephen DiVerdi

Progress isn't made by early risers. It's made by lazy men trying to find easier ways to do something.

Robert Heinlein (1907-1988)

Acknowledgements

There are many people without whom this thesis would not be possible. I would like to thank my advisor, Tobias Höllerer for being a constant source of guidance and encouragement over the last five years – without him, I would have completely missed out on the world of augmented reality, as well as all the other interesting projects he put in front of me. Thank you to the rest of my committee, Matthew Turk, Yuan-Fang Wang, and George Legrady for your ideas, feedback and criticisms, which has all been instrumental in shaping this work into what it is today.

Thank you to the many collaborators in the lab over the years:

Jason Wither, for being there over the years to bounce ideas off of and keep me looking at things from a fresh perspective,

Mathias Kölsch, for his advice in the earlier years and for going through it all first,

Cha Lee for helping me keep an upbeat attitude at the end and carrying on the tradition,

My IGERT collaborators, Ethan Kaplan and MarkDavid Hosale, for helping me break out of my computer science mold a bit,

Everyone on the FogScreen project, Ismo Rakkolainen, Alex Olwal, Nicola Candussi, Marc Breisinger, and Thomas Klemmer, for two years of demoing the FogScreen and explaining how it works,

Everyone in the Four Eyes Lab and the IGERT lab over the last few years for your friendship and input, and most importantly for putting up with me for so long.

Finally, thank you to my parents, for supporting me even when you had no idea what I was working on, and even though it wasn't chemistry.

This research was funded in part by NSF grant #IIS-0635492, NSF IGERT grant #DGE-0221713 in Interactive Digital Multimedia, a research contract with the Korea Institute of Science and Technology (KIST) through the Tangible Space Initiative Project, and an equipment donation from Microsoft.

Curriculum Vitæ

Stephen DiVerdi

Education

2006	Master of Science in Computer Science, University of California, Santa Barbara.
2002	Bachelor of Science in Computer Science, Harvey Mudd College, Claremont, California.

Experience

2005 – 2007	Software Developer, WorldViz Inc..
1999 – 2007	Graduate Research Assistant, University of California, Santa Barbara.
1999 – 1999	Software Developer, Adobe Systems Inc..
1998 – 2006	Teaching Assistant, University of California, Santa Barbara.

Selected Publications

Stephen DiVerdi, Daniel Nurmi, Tobias Höllerer, "ARWin - A Desktop Augmented Reality Window Manager," In *Proc. Intl. Symp. on Mixed and Augmented Reality*, October 2003.

Stephen DiVerdi, Daniel Nurmi, Tobias Höllerer, "A Framework for Generic Inter-Application Interaction for 3D AR Environments," In *Proc. Intl. Augmented Reality Toolkit Workshop*, October 2003.

Stephen DiVerdi, Tobias Höllerer, Richard Schreyer, "Level of Detail Interfaces in an Augmented Reality Application Environment," In *Proc. Intl. Symp. on Mixed and Augmented Reality*, November 2004.

Stephen DiVerdi, Tobias Höllerer, "Image-space Correction of AR Registration Errors Using Graphics Hardware," In *Proc. IEEE Virtual Reality*, March 2006.

Jason Wither, Stephen DiVerdi, Tobias Höllerer, "Using Aerial Photographs for Improved Mobile AR Annotation," In *Proc. Intl. Symp. on Mixed and Augmented Reality*, October 2006.

Stephen DiVerdi, Tobias Höllerer, "GroundCam: A Tracking Modality for Mobile Mixed Reality," In *Proc. IEEE Virtual Reality*, March 2007.

Tobias Höllerer, Jason Wither, and Stephen DiVerdi, "Anywhere Augmentation: Towards Mobile Augmented Reality in Unprepared Environments," In G. Gartner, M.P. Peterson, and W. Cartwright (Eds.), *Location Based Services and TeleCartography, Series: Lecture Notes in Geoinformation and Cartography*, Springer Verlag, 2007.

Abstract

Towards Anywhere Augmentation

Stephen DiVerdi

For Augmented Reality (AR) technologies to experience wide-spread adoption, the barrier to entry must be reduced significantly. In particular, setup costs of new systems in new environments are prohibitive for high-fidelity augmentation. The goal of this thesis is to take first steps towards an overall reduction in these costs.

The main AR challenges that I focus on in this thesis are realistic lighting of virtual geometry, registration of polygons with the scene, tracking a user in a mobile context, creation of 3D annotations, and acquisition of the scene lighting environment. Each of these challenges is a fundamental component of many AR systems that commonly requires laborious setup in new environments. The contributions presented in this thesis directly address these challenges, lowering the startup costs.

My first contribution is to enable more realistic lighting in AR applications by dynamically measuring the lighting environment to accurately shade virtual objects, and to use the video image to facilitate application of virtual light sources to dynamic physical geometry. Realistic lighting also depends on accurate reg-

istration of polygons with physical objects. My second contribution reduces the impact of registration errors for these polygons by shifting polygon edges to match image intensity edges.

For mobile AR systems, high quality wide-area tracking is necessary. My third contribution is the GroundCam, which combines a camera pointed at the ground, used as an optical mouse, with a GPS unit for untethered, high quality position estimation. Good position estimates are needed to accurately create 3D annotations in the outdoor scene, which is what my fourth contribution focuses on. The aerial annotator system uses aerial photographs as a second viewpoint to enable easy annotation placement in outdoor scenes. Finally, my fifth contribution is Envisor, an application that allows users to quickly acquire environment maps of these scenes with just a head-worn or hand-held camera.

To facilitate the work in this thesis, I developed two frameworks for conducting AR research. ARWin is an indoor desktop system that acts as a 3D application window manager, and is rapidly-deployable in new environments. Outdoor, mobile research is conducted within ARagorn, a wearable computing platform built with inexpensive, commonly-available hardware.

Tobias Höllerer, Ph. D.

Dissertation Committee Chair

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Thesis Statement

> *This thesis work improves the usability of high-fidelity augmented reality in new environments: Novel applications of computer vision, computer graphics, and user interface techniques alleviate setup costs of desktop and mobile augmented reality applications.*

1.1 Document Outline

The introduction to this thesis describes the problem area of Anywhere Augmentation and gives a summary of the contributions of this work. Chapter 2 discusses in detail my approach to the domain of desktop Augmented Reality (AR), while Chapter 3 focuses on mobile AR. Finally, conclusions and thoughts on future directions for this work are discussed in Chapter 4.

1.2 Problem Description

The steady progress of AR research has resulted in many advances in basic, enabling technologies for AR applications. Improved tracking, better system design, and higher quality visuals have all increased the appeal of AR systems. However, such high-fidelity augmented reality is not without its own limitations. As system complexity increases, these solutions rely on significant setup costs to achieve their impressive results. Accurate tracking can require instrumenting the surrounding environment with fiducials or sensors [52, 81], which then may need position calibration. Accurate rendering of virtual objects in the physical lighting environment, called common illumination, often necessitates high dynamic range light probes to be measured before the system can be used [26, 40, 4]. It is very common for systems to require detailed geometric models of the scene, to improve the quality of tracking and rendering results [40, 86]. Finally, sophisticated systems increasingly depend on expensive, niche hardware in many cases for real-time performance.

The problem with these high initial costs is that they form a barrier to entry for AR applications. Potential users are frequently turned away when it becomes apparent that use of an AR system will require a few days of building, modeling, instrumenting, calibrating, and measuring, assuming all the required pieces of

hardware are on hand. To create an active community of potential developers, it is important to foster experimentation in AR technologies by making use as simple as possible. Augmented reality must become a casual technology that people who are not experts in the field can try out themselves before widespread adoption can be expected.

The concept behind Anywhere Augmentation, then, is to rephrase the goal of AR research – rather than ask, "What do we need to get high-quality results?", my thesis comes from the opposite direction and asks, "What quality of results can we achieve with what we already have?" Ideally, an AR system developed with this goal in mind would be able to be used "out of the box" in a new environment with no setup necessary. However, it is reasonable to expect some small amount of initial effort, so long as it does not interfere overall with the experience. Thus, I focus on systems that take no more than about half an hour – enough time for quick calibration or semi-automatic acquisition of environment data, but not enough for the careful measurement and setup work required by AR systems today.

Towards this goal, there are very many avenues of research that may be pursued. Basic technologies such as tracking, automatic modeling, and rendering algorithms are all important. At the same time, developing applications that need to be extremely mobile and operate under a wide variety of different condi-

tions is important from the user's side of the field. The breadth of this research area is explored and narrowed down in the next section.

1.3 Anywhere Augmentation

Certainly, the motivation behind Anywhere Augmentation is not a new one – experts in the augmented reality field are definitely aware of the limitations of current technologies, and research has progressed in many areas that have Anywhere Augmentation implications. However, this thesis represents the first work to unify technologies, methodologies and applications under the common banner of Anywhere Augmentation, and to examine the open research problems from this vantage point.

Figure 1.1 gives an overview of some of the pertinent technologies and how they inter-relate in this area. In this diagram, there are large groupings that combine related technologies into sub-fields of Anywhere Augmentation research. "Pose Tracking" contains methods for providing user position and orientation in indoor and outdoor environments without relying on environment instrumentation. Work such as Davison's SLAM tracking [25] fits in this category. "Lighting Acquisition" is concerned with measuring and constructing a model of the distribution of light in a new environment. Similarly, "Geometry Acquisition" deals with creating

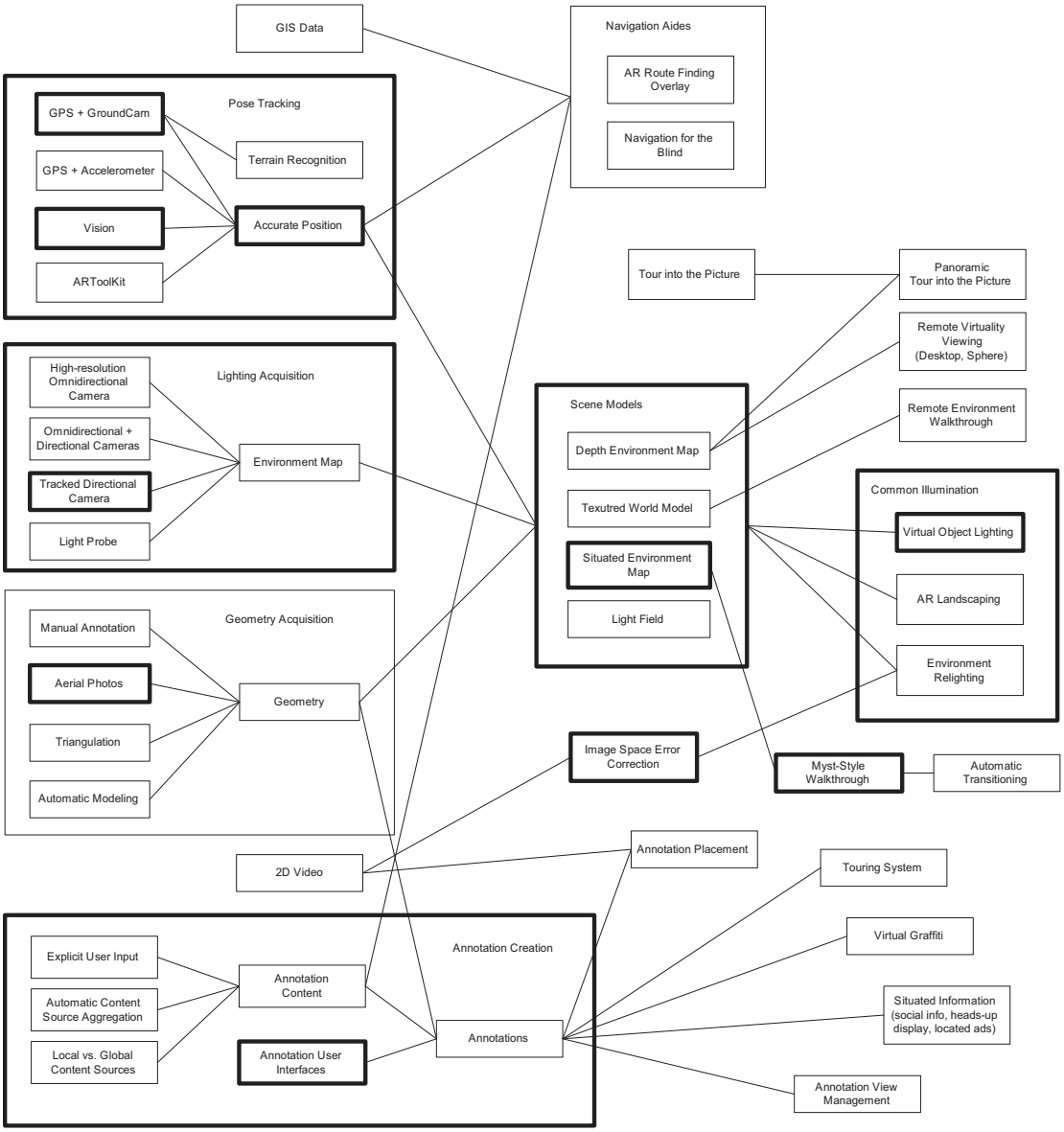

Figure 1.1: A brief overview of the research areas related to Anywhere Augmentation. The areas of my contributions have dark borders.

a model of the geometry of a scene. These two components combine to form a variety of different "Scene Model" representations, which can then enable a variety of different applications that need both lighting and geometry information. For example, "Common Illumination" techniques use scene models to seamlessly integrate the appearance of physical and virtual objects. "Annotation Creation" deals with the placement of AR content, in the form of situated metadata within the environment – a whole host of applications and techniques relates specifically to creating and consuming this type of content.

In order to make my thesis feasible, the scope of the Anywhere Augmentation research area must be carefully selected. In the next section, I outline the specific contributions that make up my thesis.

1.4 Thesis Overview

Overall, my thesis is a first step in a selection of the important Anywhere Augmentation technologies, towards a generally improved AR application experience. The work can be broken up roughly into contributions in the areas of desktop AR and mobile AR. While most of my work can be applied in both environments, applications that exist in both domains are exceedingly rare, so the dichotomy reflects how the underlying technologies will be developed and evalu-

ated. The contributions in desktop AR are presented in Chapter 2, and mobile AR in Chapter 3.

The aim of my work in desktop AR is to improve the quality of visuals experienced by users in a small, controlled space such as a user's office. This setting poses limited tracking and data acquisition challenges, but has significant interface and rendering problems which I focus on. In particular, I present a desktop AR application framework, in which I demonstrate an intuitive, tangible user interface for desktop work. This framework is also used for techniques that address the common illumination problem for dynamic scenes, and a technique to reduce the visual impact of registration errors. The system relies only on an inexpensive camera and head-worn display, and can quickly be set up in any office setting.

My contributions in mobile AR are focused on the challenges presented by the size of the environments applications operate in. Specifically, the large space makes online data acquisition an important component of a mobile system, for content creation applications as well as visualizations. I present an outdoor, wearable system framework that is used to develop techniques appropriate to this setting. Within this framework, my first contribution is an application that takes advantage of ubiquitously available aerial photographs to augment the user's interface for 3D annotation. As accurate, wide-area tracking is critical to mobile AR, my next contribution is a computer vision tracking modality that operates over larger

areas with less error than previously possible. The third contribution is a framework for remote exploration and mapping of an environment by a scout user with a wearable computer. Finally, the fourth contribution extends the abilities of such a remote explorer system with online environment map construction based on a user's tracked video stream, creating a light probe representation of the acquired scene to be used to shade virtual geometry placed in the scene.

Chapter 2

Desktop AR

The first component of my thesis work focuses on augmented reality in an indoor setting. I developed a framework for prototyping interfaces and techniques for desktop computer use enabled by augmented reality. The work consists of a tangible user interface, two techniques for the common illumination problem, and an algorithm for correcting registration errors. The result is a system that allows for rapid deployment of higher-fidelity desktop AR applications than previously possible. This chapter summarizes this result, and further information can be found in the related publications [31, 32, 30, 27, 28].

Within the desktop AR framework, my contributions are

- a rapidly-deployable application window manager that makes use of an existing tracking solution for its tangible interface,
- a technique for common illumination between physical and virtual worlds in dynamic environments, and

- a post-processing filter to reduce the visual impact of registration errors for impostor polygons.

Each of these contributions is directly relevant to the goals of Anywhere Augmentation, as outlined in the introduction, by improving the visual fidelity and interface usability for applications with low setup costs.

2.1 System

The framework I developed is called ARWin, which stands for Augmented Reality Window Manager. The goal was for ARWin to act as a standard desktop window manager, which is responsible for handling the placement, maintenance and rendering of application windows, as well as handling input event distribution. Traditionally, window managers are 2D WIMP-based (Windows, Icons, Menus, and Pointers) interfaces. There are a few exceptions (e.g., Project Looking Glass [82] and the Task Gallery [87]) that have tried to extend the windows metaphor into 3D environments with some success. ARWin extends the 3D window manager concept by using augmented reality to place applications in the space above a user's physical desk, where they can coexist with real objects. While there are other AR application environments, such as Studierstube [93], the novel aspects of ARWin are its transparent integration of pre-existing 2D X Windows applica-

tions, and its use of the ARToolKit [80] to create a one-to-one mapping between physical and virtual objects, creating a tangible user interface for the organization of desktop applications. This interface also allowed simple interactions between applications based on proximity of their physical markers.

Aside from the interface, another major decision in the ARWin design was whether to use video or optical see-through for the display technology. This has important implications to Anywhere Augmentation, and in the end video see-through was selected. While optical see-through provides a better unobstructed view of the real world, which can be a stronger advantage in some environments where peripheral vision is important, it also includes significant calibration costs per-use and has a difficult time operating in varying lighting conditions. Video see-through is more limited in terms of field of view and unencumbered operation, but nicely avoids regular calibration and is less effected by changes in illumination.

The use of the ARToolKit with video see-through display technology means that the ARWin system is rapidly deployable to be used in new environments without difficulty. One advantage of the ARToolKit's tracking is that the fiducials can be simply printed on a regular office ink jet or laser printer and are ready for use in a few minutes. No calibration of the camera / display system is necessary, so once the markers are printed, a user can start working with ARWin immediately.

Figure 2.1: A typical ARWin desktop, with clock, weather station and graphing calculator applications attached to markers in the environment.

Since tracking is accomplished entirely by the camera's view of the fiducials, there is no instrumentation or measurement of the environment necessary as well.

The ARWin framework was developed in conjunction with Daniel Nurmi, who was responsible for the mechanism to support unmodified X Windows applications within ARWin. My contribution was the resource, event, and display manager, the user interface, and the hardware system construction.

2.2 Common Illumination

Common illumination is the problem of rendering physical and virtual objects with a common lighting environment – that is, physical and virtual light sources will affect physical and virtual objects, as if they were seamlessly integrated. This is an important component of improving the visual fidelity of AR applications, as

Figure 2.2: A person wearing the ARWin HMD, a Sony LDI-D 100B with a Point Grey Dragonfly camera.

virtual objects do not look real unless they are properly lit the same as other real objects. While there has been work on common illumination previously [40, 60, 4, 36, 26], it has focused on techniques that require significant setup, including careful modeling of the entire scene, high dynamic range light probe acquisition, and global illumination computations that can hinder interactive framerates. Even less work has been done that allows virtual objects to affect the lighting of physical objects, generally focusing on casting shadows [104]. These techniques always require detailed scene geometry upfront.

The proposed contribution of this portion of portion of my thesis is a technique to acquire and render virtual geometry with a dynamic physical lighting environment, and a technique to illuminate approximated physical geometry with virtual light sources, towards the goal of Anywhere Augmentation by drastically

Figure 2.3: A virtual teapot and spotlight. The teapot has a gold metal material and is lit by both the environment and a blue spotlight overhead. The spotlight also illuminates the physical table below.

reducing the initial setup required over previous systems. The results can be seen in Figure 2.3.

2.2.1 Rendering with Physical Light

Physical illumination of virtual objects is achieved by acquiring an image of a light probe of the scene, and then processing this light probe and applying it as a spherical texture map to geometry to create various material responses. The light probe consists of a small (2.75 inch diameter) silver sphere (a Christmas ornament) mounted on a marker (see Figure 2.5). The screen position of the sphere can be determined from the transformation matrix for the marker as reported by the ARToolKit, and these pixels are copied from the video stream into a texture. The texture then contains a typical environment map, which can be applied using

regular OpenGL sphere mapping for mirror reflections of the environment. For more general rendering of different materials, the environment map texture must be filtered.

In order to be able to dynamically update the filtered environment maps, I use a technique from Ashikhmin and Ghosh [6] that creates a rough estimate of Phong integrated environment maps by using OpenGL's built-in mipmap generation capabilities. Before the environment map texture is created, automatic mipmap generation is enabled to quickly create smaller, box filtered versions of the environment map. I can then select which level to render with by specifying the minimum mipmap level. This creates a rough approximation of a glossy material.

With a set of filtered environment maps, ranging from specular to diffuse, arbitrary material properties can be simulated by selecting the appropriate maps and blending them together. The material's shininess factor determines which level of glossy specular map is blended with the diffuse map. Samples from these textures modulated by the material's diffuse and specular responses represent the material's physical reflectance, which is then added to any virtual lighting calculations and material texture maps for the final reflectance value. While this technique is not physically accurate, it does produce convincing results (see Figure 2.4).

Figure 2.4: A virtual torus with an unpolished gold material, illuminated by the physical environment.

2.2.2 Rendering with Virtual Light

The obvious way to add lighting to physical geometry is to create a replica of the physical objects in the virtual world, calculate the lighting for this proxy, and then add the difference into the image of the environment. However, acquiring detailed scene geometry is a large setup cost , and it must be repeated whenever the environment changes. To address this issue, my proposed technique approximates lighting of the environment using a significantly simplified model of the physical objects present. For example, the surface of a desk is modeled as a flat plane, and low-profile objects on the desk such as a mouse or keyboard do not need additional modeling. A monitor can be approximated with a single scaled

Figure 2.5: An ARToolKit marker with a light probe attached. The bullseye shows the marker has been recognized.

cube. In fact, for most large objects, a simple scaled box is sufficient, and small objects generally do not need modeling. Given the low complexity of this sort of rough scene geometry, it is a much simpler task to create a proxy of the physical environment in short order.

Since flat polygons are used to approximate rough geometry, the virtual illumination must be adjusted to properly account for fluctuations in height. Simple OpenGL blending creates the appearance of illumination of a flat surface, and so is not sufficient. However, the image of the scene captured from the camera provides useful information when projected on top of the simple scene model. Each polygon in the model has a physically lit texture associated with it. If the physical

illumination is known then the pattern of highlights and shadows on the texture can be associated with the dominant illumination direction. This information represents the bumps along that particular direction, and the virtual illumination can be modified to take them into account. If the virtual light is placed near the area of brightest physical illumination, then it is clear that the virtual light will have the same highlights and shadows as are present in the projected video texture. Conversely, if the virtual light placed opposite the brightest physical illumination, the highlights and shadows will be opposite.

Therefore, a more accurate bump-mapped virtual illumination can be calculated using the projected video texture. The flat polygon shaded value is called the "additive factor", A, and the intensity of the projected video texture is the "multiplicative factor", M. Then the dot product of the virtual light vector, L, and the dominant lighting direction, E, determines the contribution of each of these factors to the final response, R, using linear interpolation / extrapolation:

$$\alpha = L \cdot E \tag{2.1}$$

$$R = (1.0 - \alpha)A + \alpha M \tag{2.2}$$

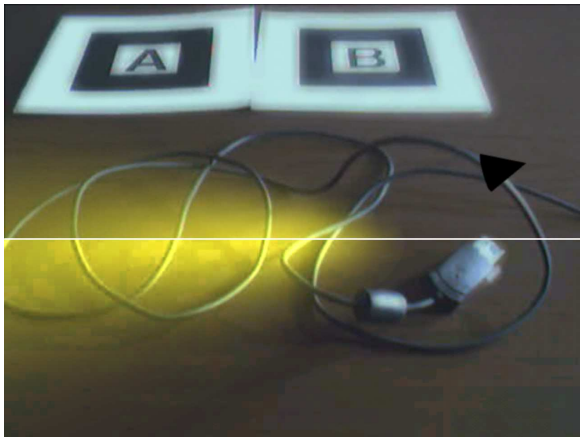

Figure 2.6: A physical table and wire illuminated with a virtual spotlight coincident with the physical illumination. The top half is using alpha blending, while the bottom half is using the custom shader. Note that with the shader, highlights are brighter and shadows are darker.

This computation is done in a fragment shader, taking advantage of programmable graphics hardware to avoid any significant performance hit. The results can be seen in Figure 2.6.

2.2.3 Common Illumination Results

As a result of these techniques, improved visual fidelity is possible over traditional AR application rendering with little upfront effort on the part of the user. The use of a tracked light probe for physical illumination of virtual objects allows for dynamic updates to changes in lighting, for virtual objects with a greater sense of presence within the physical environment. The use of filtered environment maps also improves the shading quality of virtual objects, giving them a distinctly

more realistic appearance. Using virtual lights to affect physical objects is a less common application task, but as augmented reality becomes more common, experimentation along these lines will develop unforeseen applications areas in which a complete common illumination model will be important, and virtual lighting of physical objects is a critical component of that model. The presented technique allows unmodeled geometry to be roughly lit, giving a more realistic impression than previously possible without carefully modeling and tracking all scene elements. In dynamic environments with large amounts of movable geometry such as a work desk, offline modeling is simply not feasible, so this technique fills an important role for these types of applications.

2.3 Registration Error Correction

One side effect of the goals of Anywhere Augmentation is that the lack of setup cost can mean high-quality tracking solutions must be forgone in favor of faster methods. The result is that registration errors may be more common than in current top of the line systems. Such errors are especially problematic when virtual and physical features should coincide, such as when virtual geometry is drawn directly on top of a physical object to affect its appearance, which may occur in applications such as highlighting of an object using wireframe outlines or

colored polygons, halo glow around objects, re-texturing of physical objects, and re-lighting with impostor polygons.

To compensate for errors introduced by faster tracking methods, I propose a post-processing image-space filter that can be applied to AR applications regardless of tracking technology, to consistently reduce the visual impact of these errors. The basic assumption of the technique is that geometric edges in an AR scene model correspond with edges in the physical world. Edges are detected in an acquired video of the scene. For each edge in the virtual model, the algorithm searches a region determined by a per-vertex tracking-error estimate (provided by the application) for strong nearby edges, and then smooths the detected edges. Finally, the original model polygons are rendered, clipped against the detected edges so they approximate the video features. The result is virtual objects that more closely match strong features in the physical scene, and experience less jitter in their positions.

It is important to note that the tracking result is not modified and no rigid transformation is performed. The technique is an image space warp that is performed as a post-processing step, which is fundamentally different from techniques which use image edges as part of the tracking computation. This is different from established tracking technologies that use image edge information to refine their pose estimates [55, 56, 25, 21]. The difference is my proposed technique is gener-

ally applicable to a wide variety of AR tracking technologies, with easier integration than would otherwise be possible by allowing loose coupling of image edge information with other tracking modalities.

The inputs to the algorithm are: a list of quad polygons, a list of per-vertex position error estimates, and an image of the physical scene. In order, the algorithm

1. performs edge detection on the scene image,
2. searches edge image within polygon error regions for strong, similar edges,
3. (optional) smooths individual detected edges, and
4. renders original polygons, clipped to detected edges,

and the output is a visual of the virtual scene with improved geometric registration:

Per-vertex estimates of tracking error must be provided. Tracker errors can be propagated through a series of transformations to provide the position and orientation error of a local coordinate system, which can then be applied to individual vertices to determine a region of the screen where a vertex may exist [19, 63]. These errors are used to determine the search regions for step 2. The underlying tracking technique is unimportant – all that is needed is an estimate of its error.

2.3.1 Edge Detection

I rely on established edge detection algorithms for the first step of the technique. The default algorithm is a GPU implementation of a 3x3 Sobel filter – a fragment shader samples the texture in the kernel and outputs a color encoding the gradient direction and magnitude.

I rely on established edge detection algorithms for the first step of our technique. The default algorithm is a GPU implementation of a 3x3 Sobel filter – a fragment shader samples the texture in the kernel and outputs a color encoding the gradient direction and magnitude. This is the standard choice for reasons of speed, but Sobel filtering suffers from the lack of any edge continuity enforcement, so edges can become fragmented easily.

An alternate choice is OpenCV's Canny function [70], which is a standard CPU implementation of the Canny edge detection algorithm [15] (GPU implementations are now available as well [39]). While Canny edge detection does do a better job in general of finding exact edges by enforcing an edge continuity constraint, the output is only a binary value, lacking any information about edge strength or orientation. The results are also very sensitive to the appropriate threshold values, which require tuning from scene to scene, and possibly even within a scene, if there is a significant change in lighting. Because of the sensitivity to

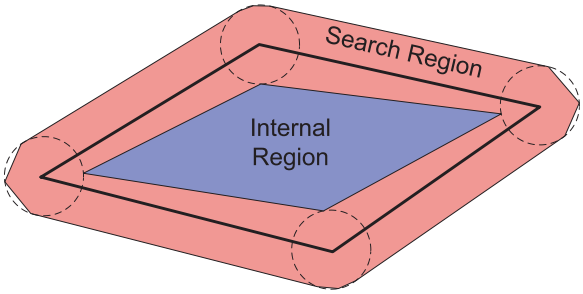

Figure 2.7: An input polygon is drawn in thick black. The dotted circles around each vertex shows the tracking error estimates. The blue region is guaranteed to be part of the polygon, while the red region is uncertain.

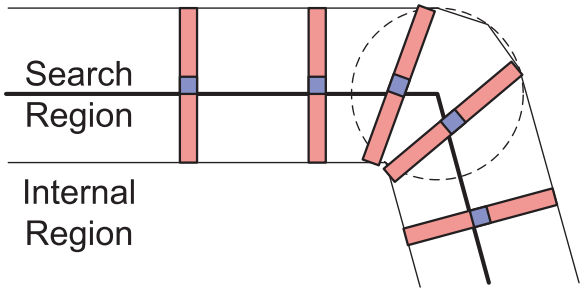

Figure 2.8: For each pixel along the input polygon edge (shown in blue), a perpendicular line of pixels (shown in red) is searched for edges in the image. At the corner, these search lines extend radially from the internal corner.

tuning and binary-only output of Canny edge detection, the Sobel operator is preferable in general.

2.3.2 Edge Searching

The second step is to search the edge image for strong edges near each polygon edge. First, back-facing polygons are culled. Per polygon, I project the vertices to screen coordinates and use the vertex error estimate to determine a region of

the screen where the vertex may lie (the dotted circles in Figure 2.7). Adjacent vertices define a search region (roughly the convex hull of the two vertex regions, shown in red in Figure 2.7) in which the edge can be found. The interior of the search regions is guaranteed to be part of the polygon regardless of the vertices' actual positions within the error estimates (the blue region in Figure 2.7).

Once the search regions have been determined, each edge is rendered as a line, with the *search shader* activated and the edge image as a texture input. Search regions are defined in texture coordinates. The search shader walks perpendicular to the edge, or near corners, radially from the internal region's corner (see Figure 2.8). It samples the edge image along this path, retrieving that pixel's edge magnitude and orientation. A weighting function is applied to these samples, and the position of the maximum weighted sample is encoded as an offset vector in the red and green channels of the output color.

$$w = s * \frac{d}{d_{max}} * |v_g \cdot v_s| \tag{2.3}$$

Sample weights are the product of the edge strength s, a distance term, and an orientation term (Equation 2.3). The distance term linearly weights the sample based on distance from the input polygon edge. The orientation term is the absolute value of the dot product of v_g, the gradient vector, and v_s, the search direction vector – if the two vectors are parallel or antiparallel, then the detected

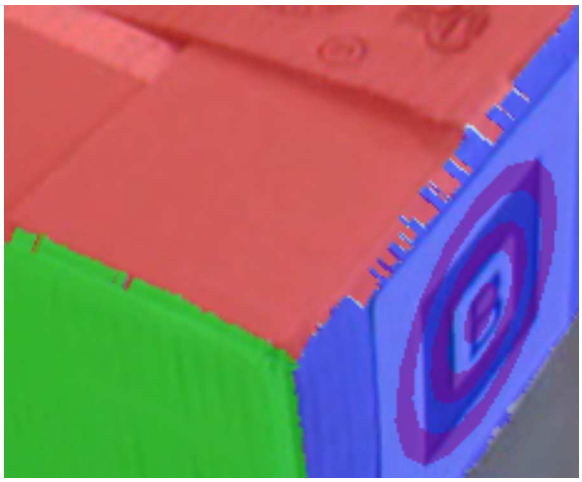

Figure 2.9: An example of the detected edge results, before smoothing. The red-blue edge shows severe fraying due to the low contrast image edge – global smoothing is necessary. The red-green edge has much more isolated noise and would benefit from localized smoothing. The blue-green edge is an ideal result of the detection.

edge and the polygon edge are similarly oriented. The final result is a weight value between 0 and 1, representing the likelihood of the particular sample.

2.3.3 Edge Smoothing

The detected edges are often quite noisy – that is, there are frequent discontinuous jumps between adjacent offsets, which create a "frayed" appearance (see Figure 2.9). This noise flickers rapidly among frames, creating a displeasing visual artifact. One possible way to address this problem would be to improve the edge searching algorithm. Instead, I chose to implement a separate edge smoothing

step. This is faster than more robust edge searching and allows the smoothing filter to be modified or removed depending on application needs.

The smoothing step is applied by rendering the polygon edges as lines again, with the *smoothing shader* activated and the detected edge offsets from step 2 as a texture input. For each polygon edge, the shader takes regular samples of the detected edge offset. A running sum of offsets is kept, to determine the mean offset at the end of the pass. Additionally, a measure of the "noisiness" of the edge is calculated for each sample and accumulated. Once the walk is complete, the noisiness measure is examined – if it exceeds a user-specified threshold, the edge is too noisy and each offset value is replaced by the mean offset. Otherwise, the original offsets are kept. The mean value is used instead of a more appropriate measure such as the median due to the lack of efficient n-element median algorithms for graphics hardware.

The noisiness measure is based on the second derivative of the detected edge offsets. At each sample, a 1x3 Laplace filter ($[-1, 2, -1]$) is applied. The mean of the absolute values of the second derivatives is the "noisiness" of the detected edge.

If the smoothing operator were implemented in a fragment shader, for each fragment, it would need to sample the entire edge and recalculate the mean and noisiness. For an edge of 100 pixels, this means 10,000 textures samples, which is

clearly a waste of processor cycles. I avoid this level of redundancy by implementing the smoothing filter in a vertex shader, which is then executed per-vertex, or twice per edge (since vertex shader outputs are interpolated across the line, the same result must be computed at both vertices). The results are passed to the fragment shader via the vertices' output colors, which are shared among the fragments. This is made possible by using the programmable shader capabilities made available in NVIDIA's 6000 series GPUs – specifically, support for vertex texture sampling. In absence of these graphics hardware capabilities, the smoothing step can be omitted, or it can be done more slowly in a fragment shader as discussed.

2.3.4 Rendering

After the previous three steps, the result is a smoothed detected edge image which encodes the per-pixel offset of the detected edge from each polygon's original edges. To render the newly deformed polygon, first, the internal region is rendered normally. Then, the unknown border regions are rendered with the *clipping shader* and the detected edge texture as input. In this final pass, each fragment samples the detected edge image and compares its position to the sample – if it is outside (determined by the inside and outside boundaries of the search region), its alpha value is set to zero, but if it is inside, its alpha is unchanged. Near the detected

edge, alpha values drop off linearly for a smooth polygon border. Since each rendered pixel does only one texture sample, this pass is very fast.

Alternately, this final pass can be slightly altered to accommodate wireframe rendering. The polygons' internal regions are omitted, and the clipping shader sets pixels' alpha values based on the distance from the detected edge, for an antialiased line.

Some filtering of the detected edge image can also be done in this step. Rather than sampling one pixel for the edge offset, its neighbors can be sampled as well, and the multiple values are combined to compute the final offset. I implemented support for 1x3, 1x5 and 1x7 block filters, as well as a 1x3 median filter. As graphics hardware is designed for this type of filtering, it does not significantly affect performance.

2.3.5 Registration Error Correction Results

For testing, the post-processing filter was applied within the ARWin framework to overlay a virtual model of a box on top of a physical, tracked box. The ARToolKit is prone to small errors in orientation estimates, which propagate to large translational errors for vertices that are far from the marker's center. With an NVIDIA GeForce FX 6800GT graphics card, there is a modest 8.2% drop in framerate, from 61 to 56 frames per second, when processing a pre-recorded

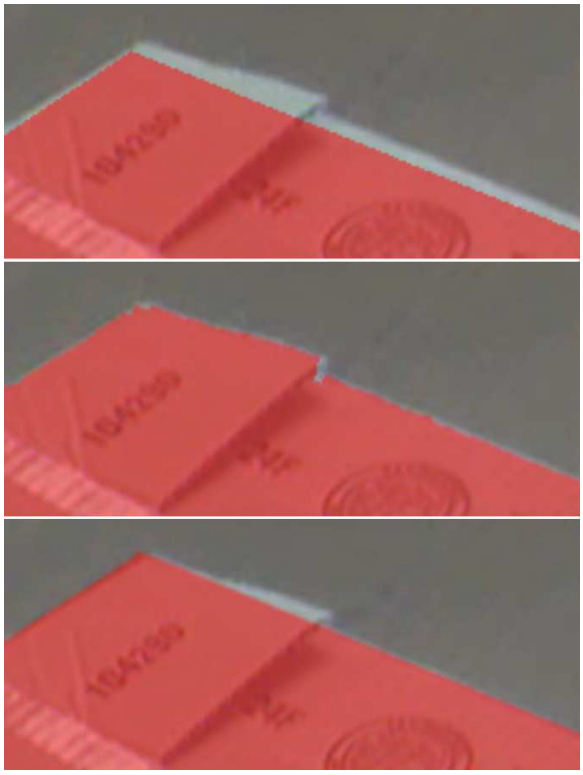

Figure 2.10: Comparison of results. *Top to bottom:* The original polygon edge, the detected edge, and the smoothed detected edge.

640x480 video stream, using all four steps of the technique, with a 1x3 block filter in the fourth step.

Figure 2.10 shows a comparison of the original polygon edge, the unsmoothed detected edge, and the smoothed detected edge. The original edge clearly shows registration errors – because of tracking error, the polygon edge is moved away from the physical edge, and because of modeling error, the complex shape of the physical edge is represented by the straight edge of a polygon. Since this is a high contrast image region, the detected edge matches the physical edge nearly

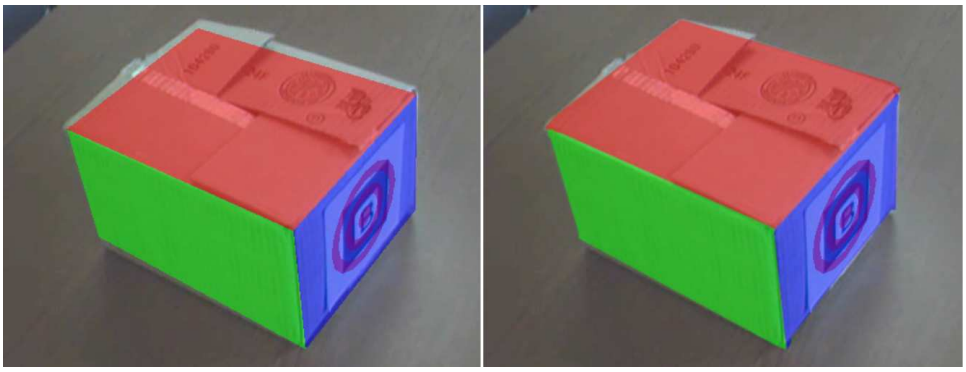

Figure 2.11: Comparison of results. *Left to right:* The original overlaid polygons, and the corrected result.

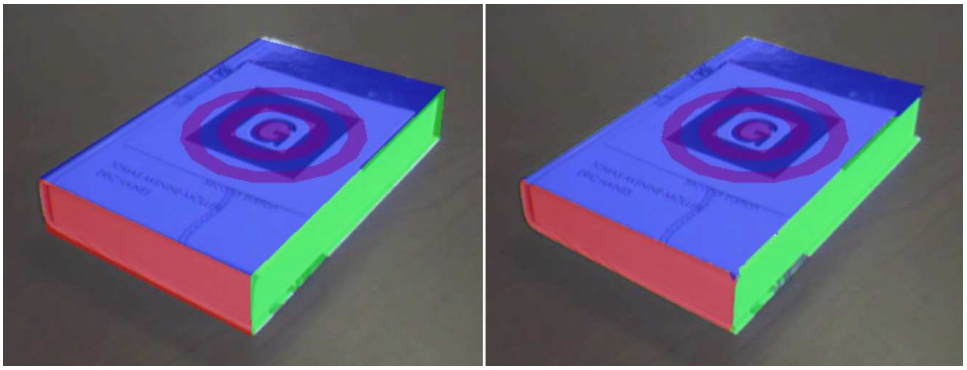

Figure 2.12: Comparison of results. *Left to right:* The original overlaid polygons, and the corrected result.

perfectly, correcting both the tracking and modeling error. The smoothed edge loses the ability to correct the modeling error, but is still a marked improvement over the original edge, due to the tracking correction.

The quality of the unsmoothed edge depends heavily on the edge detection result. If the edge detection step finds a clear, strong edge within the search region of the polygon, then the detected edge will be match the image's edge very closely,

with very little noise. Unfortunately, in real environments with complex lighting and low dynamic range video acquisition, low contrast images are commonplace, making good edge detection difficult. In these cases, the unsmoothed edge result will be noisy and will jitter from frame to frame, so the smoothed edge result will be more appropriate. While the smoothed edge may not match the physical edge as closely, it will be jitter less between frames and be a closer match than the original polygon edge, improving the visual quality.

2.3.6 Registration Error Correction Discussion

The nature of the design choices we made to implement this technique leads to some limitations which are important to examine. Foremost is the fact that I do not modify the tracking result, just the rendering of the polygons. This means that the technique is not a hybrid vision-based tracking system and cannot correct drift from other tracking technologies. It also means that polygons are clipped, rather than moved, so decal textures will not be shifted with the polygon's position. If text is texture mapped onto a polygon, the edge of the text will be clipped off, rather than all of the text being warped slightly. On the other hand, effects such as per-pixel lighting will not suffer from this effect, as they are calculated uniquely for each pixel.

Currently, the per-vertex tracking error estimation is a single float value, which represents a screen pixel radius. This is a simplistic assumption, as errors are more likely to be ellipses in screen space. The downside of this simplification is that the search regions may be too wide, decreasing performance unnecessarily and possibly getting distracted by edges that are farther away (though that error is minimized by penalizing distant detected edges in the search shader's weighting function). Of course, I also assume the reliability of the tracking error estimate. In the event that the error is over-estimated, time will be wasted searching larger regions, and edges may become distracted more easily (though weighting detected edges by distance reduces this effect). Conversely, if the error is under-estimated, as the edge search step may not find an edge in the search region, in which case the original input polygon edge is used. This way the result is ensured to always be at least as good as the input.

The global nature of our detected edge smoothing can be limiting as well. In the case that a detected edge is determined to be too noisy and is replaced by the mean offset edge, the result is a straight line. This means any per-pixel corrections of modeling error will be lost to avoid the noise. Clearly, it would be preferable to locally smooth the edge in a feature-preserving way, to keep the important detected edge features, but lose the distracting noise. One way this issue is addressed in the current implementation is by allowing the noisiness threshold

to be specified per-edge by the user. This way the user can judge the modeling error for an edge and determine about how much noise should be tolerated before resorting to a straight-line approximation. In my experiences , the edge noise is a distracting enough artifact that the noisiness threshold is always set to zero, forcing smoothing on every edge.

Finally, in cases where polygons are viewed from a shallow angle, or when the tracking error is very large, our algorithm will fail because there will be no known internal region of the polygon, and the search regions for opposite edges will overlap. This can cause confusion with detected edges and can result in non-convex polygons. To avoid this problem, when the internal polygon has negative area (by calculating area assuming counter-clockwise vertex ordering), our technique bails out and the original, unmodified polygon is drawn instead. This avoids visual artifacts and assures that our result is as least as good as the unmodified input.

2.4 Desktop AR Summary

The ARWin framework demonstrates the power of Anywhere Augmentation methodologies within a desktop environment. Each of the base technologies and design decisions focus on quick setup and low overhead, enabling high-quality results in a rapidly-deployable system. The application of desktop window man-

aging is a good fit for Anywhere Augmentation goals as well, since it allows users to immediately work within a familiar environment at any location, similar to how a laptop computer allows users to carry a known workspace with them to new locations. ARWin allows the same flexibility and mobility, while adding high fidelity augmented reality features as well.

Chapter 3

Mobile AR

The second component of my thesis work is focused on outdoor, mobile augmented reality. I developed a wearable computing platform designed for rapid prototyping and implementation of AR applications in an outdoor environment. Mobile AR directly embodies the spirit of Anywhere Augmentation, by enabling systems that users can wear in new environments for exploration (e.g., scouting hazardous locations), content creation (e.g., urban modeling), and content consumption (e.g., navigation directions, situated metadata). The wearable platform, called ARagorn, enables the development of the mobile AR applications and supporting technologies that are summarized in this chapter. Further details on this work can be found in the relevant publications [114, 29, 47].

Within this framework, my contributions are:

- an application that uses commonly available aerial photographs as an additional data source for placement of 3D annotations,

- a vision-based tracking modality and hybrid wide-area person-tracker with improved performance,
- a remote exploration system that uses an instrumented scout to acquire information about an environment for offline viewing, and
- an application that constructs environment maps online and uses them to shade virtual geometry.

These contributions are all relevant to Anywhere Augmentation, as they either directly lower the barrier to use of outdoor AR applications, or they are outdoor applications that demonstrate Anywhere Augmentation principles in useful domains.

3.1 System

The basic ARagorn wearable system can be see in Figure 3.1. At the core is an Alienware Area-51 m5500 laptop, which is worn on the user's back. The display is an SVGA Sony Glasstron PLM-S700E hanging from the front of the helmet, used in video see-through mode. Mounted directly above the display are a Point Grey Firefly firewire camera and an InterSense InertiaCube2 orientation tracker, and on top of the helmet is a Garmin GPS 18 position tracker. User input is through a hand-held ErgoTouch RocketMouse. All of these devices are

Figure 3.1: The ARagorn base wearable system hardware. An Alienware Area-51 m5500 laptop (worn in a backpack), an SVGA Sony Glasstron PLM-S700E display, a Point Grey Firefly firewire camera, An InterSense InertiaCube2 orientation tracker, a Garmin GPS 18 receiver, and an ErgoTouch RocketMouse.

relatively inexpensive, off-the-shelf components. This system was developed in collaboration with Jason Wither. I was responsible for the camera, orientation tracker, and display mounting as well as the data acquisition software, while Jason made the helmet and backpack system and handled network communication.

3.2 Aerial Augmentation

In this application, the focus is on annotating an outdoor scene from within the wearable system, providing an appropriate interface to allow accurate markup in a mobile context. To reduce the amount of manual work that must be done by the user, the ARagorn system is modified to use aerial photographs of the region

Figure 3.2: A user wearing the ARagorn system to annotate an outdoor scene.

in conjunction with the wearable's acquired data. In the current implementation, 0.5m resolution aerial photographs are acquired offline from Google Maps and stitched together into a single large view of the University of California, Santa Barbara campus. However, it is possible with a wireless internet connection to download map data on the fly based on the wearable's reported GPS coordinates, enabling the system to work in new environments covered by map services without an initial setup cost. This allows the user to accurately place 3D annotations from a single position by providing a means of accurately gauging depth, reducing the amount of motion necessary by the user to annotate a scene.

With orientation tracking, from a static position a user can easily cast a ray to select a visible feature in the scene, but setting the depth of that feature is more difficult. Previous work in this area requires the user annotate the same feature

from multiple viewpoints to triangulate a position [9, 79], or estimate depth from a static viewpoint using artificial depth cues [115]. However, commonly available aerial photographs [41, 116] can be used to allow accurate 3D position input from a single location. After a user has cast a ray, the system presents the user with an aerial view of the scene and the cast ray and allows the user to adjust the ray and set a distance. The result is a significant improvement in the accuracy of 3D positions over previous AR distance estimation work [115], as well as the ability to annotate features that may not be directly visible from the user's location, such as the opposite side of a building. Automatic feature extraction from the aerial photographs allows the system to intelligently recommend salient features along the cast ray, so the user needs only to choose from the detected features and possibly refine the result.

3.2.1 Annotation Interface

The intuition behind our use of aerial photographs to assist annotation is that they can fill the role of the second point of view necessary for triangulating 3D positions. Rather than having to walk to a different location and view the point again to find its depth, the user can instead find the point on an aerial photograph to provide the necessary information to calculate the distance to that point. This also provides a unified way of making many different types of annotations

by marking up an aerial photograph – for example, specifying a corner, an edge or a region all correspond to well-understood 2D drawing tasks on an aerial photograph. Conversely, an aerial photograph alone only allows annotations to be made in 2D, whereas the combination of the aerial photograph with the first person viewpoint allows the specification of 3D positions. The usefulness of aerial photographs is also greatly increased when the user is situated in the environment being annotated. For example, it may be difficult to distinguish features in an aerial photograph alone, but when a user can stand in the scene and look at the buildings from a ground-level viewpoint, these ambiguities can be more easily resolved. Having both the aerial photograph and the first person view is analogous to having a perspective view and a top-down view in a CAD modeler – while many actions can be executed in either view independently, having both views is often faster and more powerful. The implementation of the manual annotation interface was Jason Wither's contribution in this work.

3.2.2 Feature Extraction

In addition to providing a useful viewpoint for users to manually annotate an outdoor scene, aerial photographs also provide a great deal of information that can be automatically segmented with appropriate image processing. The system leverages this information by attempting to automatically detect which feature

the user is annotating. If the feature is detected correctly, the user only needs to confirm it in the overhead view – otherwise, the same selection interface can be used to correct any errors in the detected feature. Thus, the semiautomatic approach does not significantly add complexity to the interface, but does frequently reduce the amount of input necessary, significantly reducing the burden on the user.

Small errors in the tracking and the imperfectness the feature detection necessitate a certain amount of flexibility in the set of detected features returned. To address orientation tracker error, an angular epsilon term is used to define a search cone around the cast ray in which valid features may be found. For the case that the best detected feature is not the intended feature, the best n features are returned. The user can then easily scroll through the options in the order of distance from the user with a click-and-drag motion.

Features are extracted using simple, ad-hoc filters that roughly segment out desired feature types from aerial imagery. Algorithms are largely based on the implementations available as part of the OpenCV image processing library [70].

Corner Extraction

A multiscale approach to corner detection provides results more robust to distractions for all sizes of corners a user may want to annotate, from large buildings

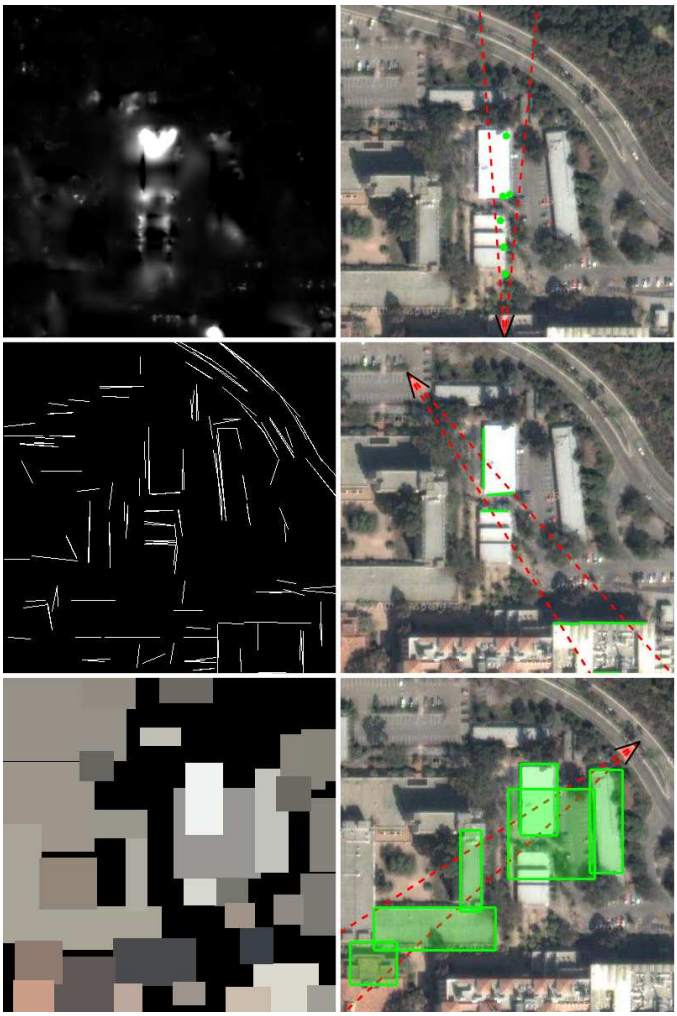

Figure 3.3: Outputs of each of the automatic feature detectors. *Left to right, top to bottom:* (a) The Harris corner transform is applied at multiple scales and summed together. (b) Corner features are selected from the local maxima of the continuous function. (c) Coherent line segments extracted from the output of Canny edge detection. (d) Edge features are selected from these line segments, weighted by their gradient magnitude. (e) After segmenting uniform color regions, components are merged and represented by their bounding boxes. (f) Region features are selected from components by size, intersection and distance from the center.

to smaller aerial features. Therefore, corner extraction uses the Harris corner detector in the search region at multiple scales and sums the results to create a likelihood for each pixel (see Figure 3.3a). Local maxima of the smooth corner function are extracted by a sliding 5x5 window. Then, the region along the user's cast ray is searched for the maximum weight pixels (from the set of local maxima). The weighting function is

$$w = w_s * w_a * w_d \tag{3.1}$$

where w_s is the strength of the corner sampled from the corner image, w_a is an angular term, measuring the angular distance between the cast ray and corner feature, and w_d is a distance term, determined by the distance to the pixel, scaled between 0 and 1. An example set of detected corner features can be seen in Figure 3.3b.

Edge Extraction

To extract edges, the Canny edge detector is first used, connected contours are extracted and simplified using a simple polygonal approximation, and edges shorter than a minimum threshold are discarded. Once the final set of edges is determined (see Figure 3.3c), the weight of each edge is calculated and the maximum weight edges are returned. The weighting function is

$$w = w_i * w_o * w_d * w_g \tag{3.2}$$

where w_i is an intersection term, measuring how much the edge intersects the cast ray, w_o is an orientation term, weighting edges perpendicular to the cast ray higher, w_d is the same distance term as used for the corner detection, and w_g is a strength of the edge, determined by sampling the magnitude of the image gradient at the edge's midpoint, computed with a Sobel filter. An example of the detected edges in a region can be seen in Figure 3.3d.

Region Extraction

Region features in aerial photographs tend to appear as regions of mostly uniform color surrounded by a boundary, so to find regions, the first step is to reduce local texture by doing morphological closure and then use the Canny edge detector to find region borders. To connect the boundaries, morphological closure is performed on the edge image, followed by thinning to restore single-pixel wide edges. The resulting image segments the aerial photograph into regions of similar color, which are extracted using a flood fill algorithm. The output components are culled based on HSV profile (vegetation areas are not the focus here), and then combined by computing their percentage overlap and their color similarity, calculated as the euclidean distance in CIE L*a*b* color space. The final list of components (see

Figure 3.3e) is then weighted and the maximum weight components are returned. The weight function is

$$w = w_i * w_a * w_p * w_d \tag{3.3}$$

where w_i is a binary intersection term, w_a is an area weight term, w_p is a perpendicular term, calculated as the perpendicular distance between the center of the region and the cast ray, as a percentage of the region's diagonal length, and w_d is the same distance weight term as the corner and edge detectors. A typical set of detected region features can be seen in Figure 3.3f.

3.2.3 Aerial Augmentation Results

Informal testing shows that the use of aerial photographs allows users to annotate scene features in 3D from a static viewpoint with much greater precision than was previously possible [115]. The longitude and latitude accuracy of an annotation position is limited only by the accuracy of the map and the ability of the user to manipulate the position accurately. Google Maps provides data at 0.5m per pixel resolution for Santa Barbara [41], and user input is generally accurate within a few pixels, so our final annotation precision is ±1.5m. Since height information is computed from the ray cast by the user, its accuracy is dependent on the quality of the orientation tracking.

While the accuracy of our approach is good, its greatest advantage is speed. The ability to create an annotation with only a few simple interactions is significantly faster than having to walk to different locations to create the same annotation by triangulation.

The performance of the automatic feature extraction was informally tested in an offline environment. For each type of annotation, 7 user positions were selected from a large area aerial photograph, and for each location, multiple visible features were targeted to annotate (57 corners, 34 edges and 31 regions). The detected features were inspected, and if any were close enough to the intended feature that manual correction would not be necessary, it was recorded as a success. The results of these tests were that corner detection was successful approximately 65% of the time, as was edge detection, while region detection had a success rate of approximately 40%. Since the inclusion of feature extraction at worst only adds an additional mouse-click to the user interface, even a low detection rate still speeds up the overall use case, as bad results are trivially ignored, while good results make further interaction unnecessary.

3.3 GroundCam

The prevailing solutions to the tracking problem in mobile AR work are to use differential GPS if it is available locally [46], use regular GPS only and be limited to low resolution (appropriate for applications such as navigation aides found in cars), or to couple a *global* tracker such as GPS, which provides wide area, absolute, low resolution data, with a *local* tracker such as an inertial navigation system (integration of linear accelerometer and rotational gyroscope data), which provides high resolution, relative and drift prone positioning [38, 33]. I present the GroundCam (consisting of a camera and an orientation tracker - see Figure 3.4) as a new local tracking modality for wide-area applications. The low cost (just a camera and a gyroscope unit) and ease of setup of the GroundCam make it appropriate for the goal of Anywhere Augmentation.

While similar to inertial tracking, the GroundCam is a significant improvement because it accumulates error much more slowly, maintaining similar small-scale accuracy for a longer period of time. The accumulated error resulting in drift over long runs makes the GroundCam unsuitable for a wide area tracking application by itself. To address this issue, we use a complementary Kalman filter to combine the GroundCam with a wide area GPS receiver (see Figure 3.4), providing better accuracy over long term use in large, outdoor environments.

Figure 3.4: The modified ARagorn wearable setup for GroundCam tracking. We use a Unibrain Fire-i 400 camera, an InterSense InertiaCube2 orientation tracker, and a Garmin GPS 18 receiver.

3.3.1 Optical Flow-Based Tracking

The inspiration for the GroundCam is a desktop optical mouse. A camera is pointed directly at the ground from just above waist height, and the video of the ground moving in front of the camera can be used to determine how the camera is moving in the plane of the ground. The result is a 2D position tracker.

Offline intrinsic camera calibration is done using Zhang's procedure [118]. The distortion coefficients from this process are used to correct the resulting artifacts in the video frames by creating a corresponding undistortion image warp that is

applied to each frame – this allows us to use image distances as direct measurements of distances in the scene.

Features are tracked frame to frame using the image pyramid-based optical flow algorithm of Lucas and Kanade [62]. A hierarchy of images at different resolutions are used to efficiently match texture features from one frame with the most similar region in another frame. If the similarity between these two regions is below a threshold, the feature is considered lost and is removed from the set.

Coherent motion must be extracted from the set of features successfully found in consecutive frames, discarding the influence of outliers. We implemented the RANSAC algorithm [34] to accomplish this task. Only one sample is necessary to estimate the image's 2D translation. Other samples are tested against this estimate by separately thresholding the differences in magnitude and orientation. Once the final set of inliers is found, the image motion estimate is computed by taking the average of all the good samples. In the event that a consensus is not reached, a fallback estimate is computed as the average of all the samples.

The computation to get world motion in real units from the image motion in pixels is straightforward. The camera is assumed to be perpendicular to the ground at some uniform height (measured offline). For a known height in meters H, camera horizontal field of view F, and camera width in pixels W, the conversion factor from pixels to meters can be found with

$$\frac{2H}{W}\tan\left(\frac{F}{2}\right) \tag{3.4}$$

In the ARagorn system, a 640x480 image from the camera with a field of view of 12.2 degrees, mounted at 1.1m (just above waist high), yields a factor of 0.37mm per pixel. See Figure 3.5.

The motion estimate is computed in the camera's frame of reference. In order to convert it to the world's coordinate system, the absolute orientation of the camera must be known. An InterSense InertiaCube2 orientation tracker is used to obtain this information. A quick offline calibration is done to orient the InertiaCube2's output by obtaining angles for north, east, south and west. During operation, the detected angle is linearly interpolated between these computed values to get the world stabilized camera orientation. The motion vector is then transformed by this orientation to yield the final, world stabilized motion estimate.

3.3.2 GPS Coupling

Our complementary Kalman filter design is inspired by Foxlin's work on orientation tracker filtering [37]. The underlying concept is to filter the error signal between two sensors, rather than filtering the actual position estimate. See Figure 3.6 for a diagram of the filter design.

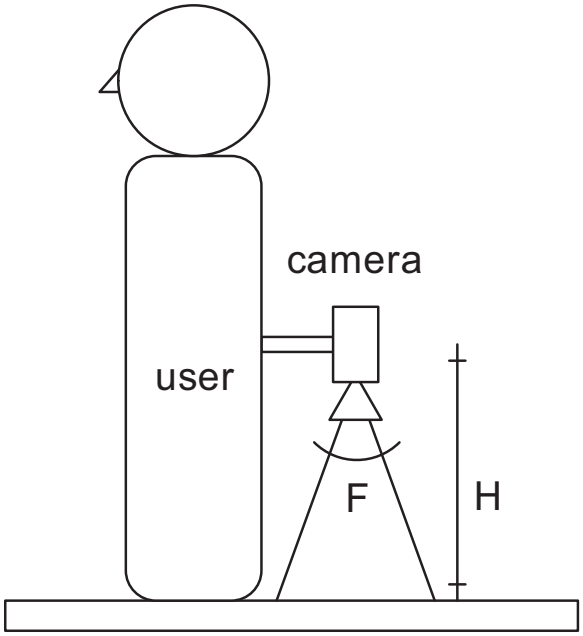

Figure 3.5: Model of the camera setup. F is the camera's horizontal field of view, and H is the height of the CCD from the ground. It is assumed the camera is oriented perpendicular to the ground.

The signal from the GroundCam is high frequency (30Hz), high resolution (1mm), and includes small random errors (10mm) and large systematic errors (drift is unbounded over time). The signal from a GPS receiver is low frequency (1Hz), medium resolution (10cm), and includes medium random and systematic errors (5m). We can model the error between the two signals as a smoothly varying random process with a Kalman filter, and then use the filtered error signal to correct the GroundCam signal on the fly, making the filtered output available at the high frequency and high resolution.

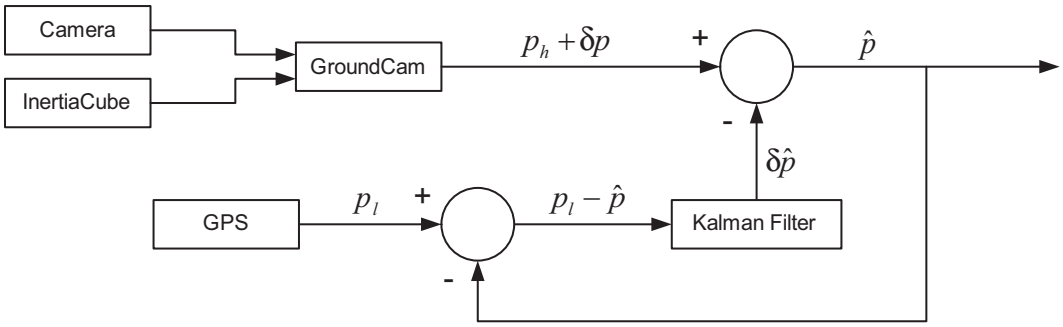

Figure 3.6: System diagram of the complementary Kalman filter. The difference between the GPS signal and the current estimate is used by the Kalman filter to create an error estimate. A camera and orientation tracker are combined in the GroundCam, which outputs a position that includes some systematic error (drift). The Kalman filter's error estimate is subtracted from the GroundCam's position to create a new position estimate. While the Kalman filter is updated infrequently (1Hz), a new position estimate is generated for each GroundCam update (30Hz).

Let p_h be the high frequency signal from the GroundCam, and p_l the low frequency signal from a GPS receiver. p is the ground truth position, $\hat{p}$ is the estimated position, $\delta p = p - p_h$ and $\delta\hat{p}$ is the estimated error signal. Since our filter operates on 2D data, p is actually the vector $[x, y]^T$. Within the Kalman filter, there are 6 process dimensions and 2 measurement dimensions. Filter variable names are standard as used in [113].

$$\mathbf{x} = \begin{bmatrix} \delta p \\ \delta\dot{p} \\ \delta\ddot{p} \end{bmatrix} \tag{3.5}$$

$$\mathbf{z} = \begin{bmatrix} \delta p \end{bmatrix} \tag{3.6}$$

$$A = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$H = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.8}$$

B and **u** are both not used, and thus zero. Q and R are empirically determined depending on the particular sensor being coupled with the GroundCam, and P is initially set so measurements are preferred at startup.

The result of a complementary filter setup such as this is that for each new high frequency update, only a prediction and then subtraction is necessary, making the processor load very low for the frequent step. The correction step is computed once per low frequency update.

3.3.3 Slip Compensation

The problem of RANSAC not reaching a consensus is analogous to the problem of slipping wheels in odometry of wheeled vehicles, and results in estimated paths that are much shorter than ground truth. Certain types of terrain are more prone to this sort of error. Figure 3.7 shows examples of the ground textures that we

encountered in our trials. Low-contrast terrains like concrete, asphalt, and carpet were much more prone to slipping than high-contrast terrains such as grass, gravel, and wood.

We made a simple attempt to compensate for some of this error, which we call *slip compensation*. The error is proportional to the rate at which RANSAC does not produce a coherent estimate, or the *slip rate*. Based on the slip rate over a short window of time, a successful coherent estimate is scaled to compensate for the missed estimates (e.g., if the slip rate is $s = 0.8$, then a coherent estimate is scaled by $(1 - s)^{-1} = 5.0$). The results of applying this compensation can be seen in Figure 3.8. While it does not correct for drift due to the integration of random errors in the GroundCam signal, it does help achieve the appropriate scale, which improves the quality of the filtered hybrid signal between GPS updates.

3.3.4 GroundCam Results

Figure 3.9 shows a typical run using the GroundCam and GPS hybrid tracking system for approximately 90 seconds. The path walked includes avoiding obstacles and going up and down steps, with wood, gravel and concrete terrain. As expected, the GroundCam exhibits some drift, partially from random errors in the motion estimate, but also from updates where a coherent estimate cannot be generated. These errors cause different effects in the GroundCam path – random

Figure 3.7: Different types of terrain with example slip rates. *Left to right, top to bottom*: asphalt (65%), concrete (80%), grass (20%), gravel (32%), wood (24%), carpet (48%). Slip rates depend on speed, jitter, lighting and debris, in addition to texture contrast.

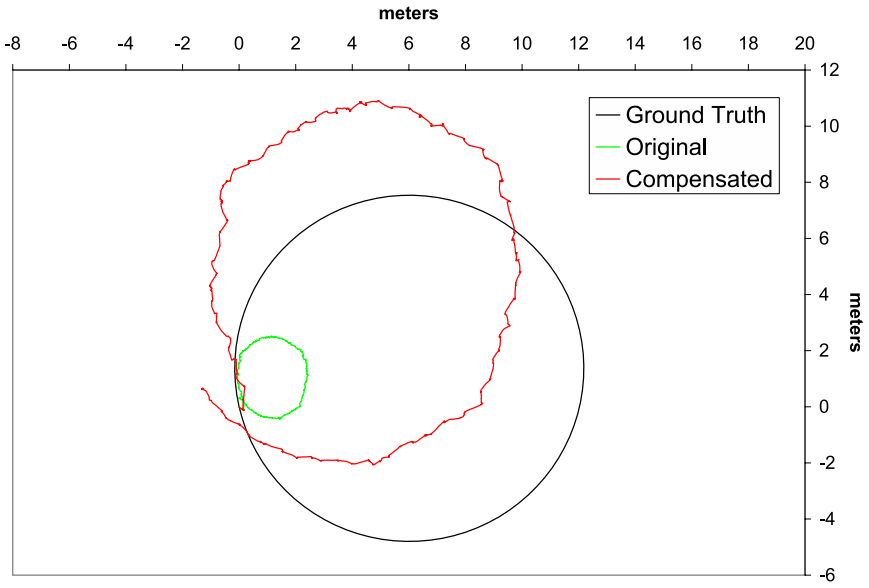

Figure 3.8: A trial run of the GroundCam with and without slip compensation, with hand-labeled ground truth. The trial was 72 seconds in duration over asphalt, and had a slip rate of 63%. Originally, the RMS error was 7.0m; with slip compensation, the RMS error is 4.8m.

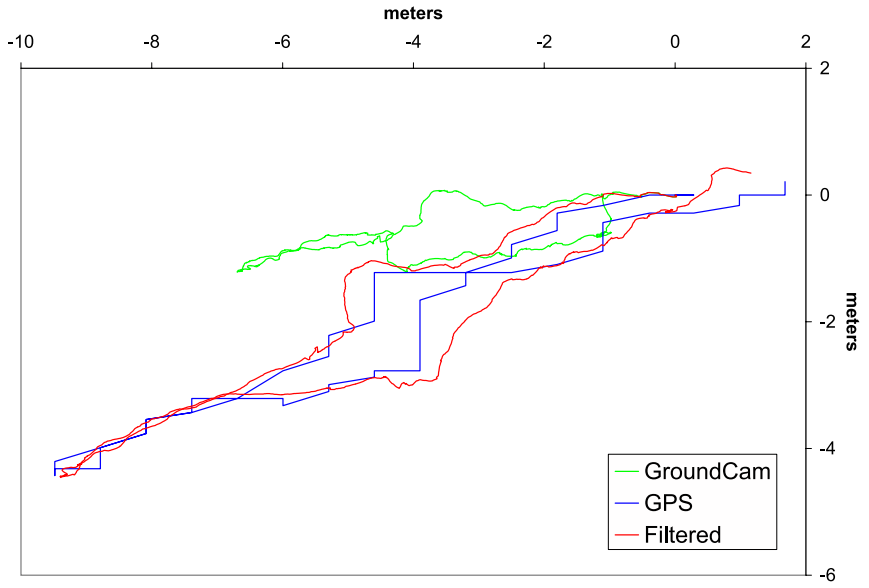

Figure 3.9: A trial run of the GroundCam coupled with GPS. The run was 90 seconds in duration, over wood, gravel and concrete terrain, and included avoiding obstacles and going up and down stairs. The slip rate was 30%.

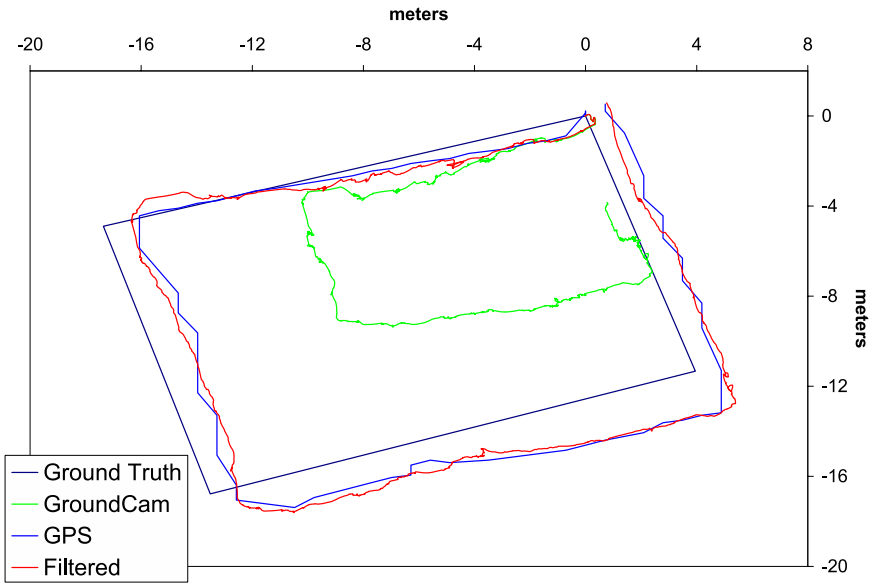

Figure 3.10: A trial run of the GroundCam coupled with GPS, with hand-labeled ground truth. The run was 81 seconds long, over concrete and asphalt, along a rectangle 18m long by 12m wide. The slip rate was 80% and RMS errors for the GroundCam, GPS, and filtered signals are 5.5m, 1.9m, and 1.9m respectively.

errors make the path less smooth, while missing coherent estimates create a shortening effect. However, the coupling with the GPS signal eliminates the effect of the GroundCam drift. Of particular importance is the much smoother quality of the filtered signal than the raw GPS signal, from the high frequency, high resolution GroundCam data, which makes the hybrid tracker very appropriate for mixed reality applications.

For comparison purposes, the run in Figure 3.10 includes a hand-labeled ground truth – a rectangular path of approximately 18m x 12m over 81 seconds on a residential street. The terrain is concrete and asphalt, which have lower contrast

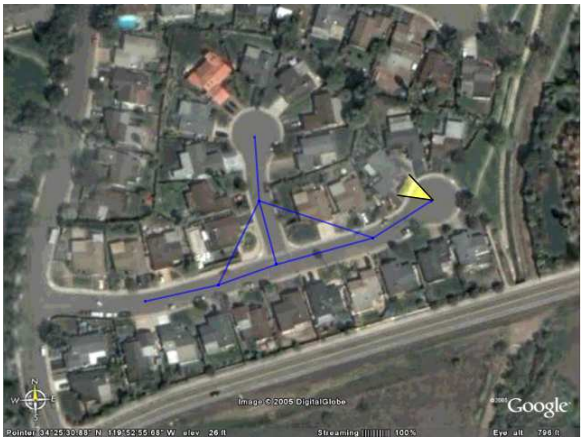

Figure 3.11: The basic topology of the world model constructed by the world building preliminary implementation, with the user's field of view represented by the yellow triangle. Discrete nodes are connected based on spatial and temporal distance, and may not reflect the actual visibility of one position from another, as around sharp corners.

textures and are more prone to noise in the error estimates. For this particular trial, our GPS receiver experienced very little random noise, but did exhibit a significant drift overall, due to our GPS unit not receiving a WAAS signal at our location. While our filtered path stays close to the GPS signal, we cannot correct for errors in the GPS position so any systematic GPS error is propagated into our tracking result. In most US locations, the presence of a WAAS signal will improve the quality of the GPS data, and subsequently improve the filtered data as well.

Figure 3.12: From the first person view, the user can see the aerial view in the inset in the lower-left corner of the screen. The adjacent locations the user can navigate to are indicated by blue arrows emanating from the bottom of the screen.

3.4 Remote Explorer

Creation of environment models for rendering and remote exploration is a complex task that often requires a significant amount of effort. Recent approaches have been developed to automate the procedure [107, 110], and large-scale acquisition projects are underway. However, these techniques require custom hardware and significant resources that are not available to most researchers, much less non-experts. Therefore, an Anywhere Augmentation effort is desirable to bring easy outdoor modeling within reach.

The concept behind this contribution is to enable a single, instrumented scout user to explore a remote location and transmit data back to a base station, where processing can be done to allow other users to explore the environment. Online

processing in the scout's wearable can also allow for use of the acquired model to enable applications such as realistic rendering of virtual geometry in these environments.

The implemented system is a prototype that tackles the world building problem in a lower fidelity manner, focusing on the scout and base interface and data acquisition. The main simplifying assumption is that the scout is an expert user whose behavior can be expected to fit a certain pre-determined pattern. In particular, the scout moves between points of interest as desired, and then while stationary, rotates 360 degrees to obtain a panoramic view of the environment at that location.

Given this input, in the form of a position and orientation tracked video stream, the preliminary implementation builds a very simple image-based world model, styled after the mid-90s video game Myst [67]. In this style of world, the user can view the surrounding environment from individual discrete locations by turning around in place. Then, the user may select a new location visible from the current one to jump to and view the scene from there. The world model that must be built then stores the images associated with the surrounding view of the environment at each of these discrete locations, plus a visibility graph that links locations to one another.

To build the set of view images, individual frames from the tracked video stream are stored to represent the view at regular angular increments. Then when the user looks around their current location, the image closest to the target view direction is displayed. The visibility graph is built first by connecting locations that are temporally adjacent, and then based on distance thresholding. Unfortunately, this does not take into account the actual visibility between two points, as positions around a corner from one another may be blocked by a building but will tend to be connected with this approach. Additional reasoning is necessary based on the video stream.

This implementation was done in conjunction with Jason Wither, Ingrid Skei and John Roberts. My contribution to the project is the world building component that takes the tracked video stream as input and generates the world model, exporting an API for a client application.

3.5 Panorama Building

To extend the fidelity of the remote explorer's world model, I developed an online environment map construction application called Envisor that allows a user to capture a full surround view of a scene by panning a camera around. There are many uses such easily acquired panoramas could fulfill. For example, within the

context of the remote explorer, they allow for a complete, compact representation of the environment surrounding a single location. Even if not integrated into a wearable computer, a home user may want to create full environment maps of their home or office environments for sharing with friends or colleagues, or for use in applications such as video conferencing or telepresence systems [45, 110] as a simple way of representing a remote environment, e.g., as a backdrop in a tele-collaboration system, or in low-bandwidth first-person interfaces such as QuickTime VR models [83] or the game Myst [67]. Environment maps are also an important component of realistic, real-time shading of geometry [94] – easy acquisition of environment maps for new scenes enables visualization of virtual geometry with the correct physical shading, as discussed in Section 2.2.

The environment map construction process is implemented as a series of component algorithms. First, the camera's 3DOF orientation is tracked using vision-based tracking, both frame to frame updates and landmark features, which may be fused with a gyroscope unit for increased robustness. The tracked video feed is then projected into a cubemap, taking care to avoid projecting dynamic scene elements to make sure the environment map only contains the static scene. Since it may be difficult for the user to make sure to cover the entire scene on their own, feedback is provided in the user interface to direct the user which directions need further acquisition, and remaining gaps are filled with a texture diffusion tech-

nique. Finally, the resulting environment map is used to shade virtual geometry placed in the physical environment.

3.5.1 Background

In general, acquisition of an environment map, or "light probe" is done by carefully photographing a reflective sphere [26, 4] and reprojecting the image into the appropriate format. Alternately, a camera with a fish-eye lens [50] or an omnidirectional camera [110, 51, 68] can be used. Whichever technique is used, the process is slow and detailed, requiring specialized hardware and offline processing. Automatic acquisition of environment maps with just a camera is a significant improvement in usability for the casual AR user.

High quality panorama construction has been extensively researched over the last decade and a half. Panoramas can be and often are used as lightprobes for many of the same applications as environment maps, such as remote presence systems. For spherical panoramas (that cover 360° pan and 180° tilt), the equivalence to environment maps is clear (simply different representations of the same data), while cylindrical panoramas (360° pan, $<$ 180° tilt) and partial panoramas ($<$ 360° pan, $<$ 180° tilt) are subsets of full environment maps. 1D and 2D scanned image mosaics (where the camera is translated, not rotated) are distinct and not explicitly discussed in this work.

Most of the early work in panorama and mosaic stitching focused on the task of aligning two adjacent images through feature- or correlation-based algorithms [18, 65, 44, 105]. Feature-based algorithms use sparse corresponding points in each image, whereas correlation-based algorithms compute similarity based on all overlapping pixels. Szeliski's approach to acquire a cylindrical panorama [105] is to warp each image according to a cylindrical projection model, where alignment can be done via pure translation. Hansen et al. create scanned image mosaics by computing affine transforms between successive images from a video sequence with a hardware-implemented Laplacian pyramid technique for real-time operation [44]. McMillan and Bishop focus on computing the transform between images during panning-only camera rotation for cylindrical panoramas [65]. The main limitation of these local-alignment only approaches is that the accumulation of error over multiple successive alignment computations will result in large errors when the camera's path revisits a previous pose, such as in closed-loop panoramas.

To address this problem, a global solution across the set of acquired images being stitched together has been used in the past. Szeliski and Shum [106, 100] proposed using a motion model that accounts for three rotational degrees of freedom, allowing arbitrary camera orientations. After computing local alignments between each image pair, a global solution is computed that simultaneously minimizes (using gradient descent) the error for corresponding feature pairs across all

overlapping images. Sawhney, Hsu and Kumar's VideoBrush [49, 91, 92] solves the same problem differently by alternating between computing pairwise image alignments, then inferring the topology of the sequence (which images overlap that are not temporally adjacent), and finally computing a global solution by simultaneously minimizing the alignment error across all topological neighbors. By iterating through these steps multiple times, arbitrary camera orientation paths can be stitched. Both of these approaches (Szeliski and Shum's, and Sawhney, Hsu and Kumar's) are capable of high-quality, full spherical panoramas. However, their solutions require all images to be available for stitching, and need multiple iterations to compute global solutions over all images. These limitations hinder the techniques' application in real-time panorama stitching for online systems, and suggest that scaling to large sets of images would be inefficient. Steedly et al. [102] presents an optimization that accelerates the performance of stitching video sequences into panoramas by only matching keyframes from within the video sequence, using a global solution that takes $O(n^2)$ for n images (as opposed to $O(n^3)$ for bundle adjustment). Aggregation of feature correspondences between images further increases performance. Capel and Zisserman [16] present an alternative that is similar to my use of landmark features, that reuses image features to compute consistent homographies across many images that may not be temporally adjacent. However, both these approach still requires all images available at

the beginning of stitching (requiring storage of the entire video sequence during acquisition) and does not provide real-time performance.

A more unique way of creating mosaics is the manifold formulation presented by Peleg et al. [76, 75, 89]. The difference in this formulation is to project the mosaic onto different manifolds other than just cylinders, spheres and planes, providing support for arbitrary camera motion (including translation) while maintaining quality. However, while the results focus on camera translation, particularly in the forward direction, only 1D panning rotation is discussed, potentially with some static roll and tilt. Indeed, it is not clear how the proposed manifolds could be adapted to support simultaneous panning and tilting.

MIT's ongoing City Scanning project [22, 57, 107] aims to acquire geometry and lightmap data for large urban environments. Towards this end, they have constructed a robotic camera rig with a camera on a pan-tilt head that automatically acquires images for spherical panorama stitching. Due to the large number of images and global optimization formulation, the offline processing can take hours. Other approaches utilizing unusual hardware include Nayar's omnidirectional camera [68], the omnidirectional camera used by Uyttendaele et al. [110], the proposed omnidirectional camera of Coleshill and Ferworn [20], and Kim et al.'s [54] stereo camera setup. The concentric mosaic work by Shum et

al. [98, 97, 17, 99] also falls into this category, requiring an acquisition rig with multiple cameras moving through concentric circular paths.

While most work in panorama stitching uses individual images as input, there is also work on stitching frames from a video sequence [44, 120, 119, 105, 64, 92, 53, 102]. Mann and Picard [64] conducted some of the earliest work, using a novel formulation of the alignment problem, but only demonstrate partial rotational panoramas. Zhu et al.'s [119] original work focuses on videos that contain large 1D translational motion, acquiring a panorama and depth information simultaneously. Later, the same group presented results on 1D panning rotation to acquire full cylindrical panoramas [120]. Kang and Shin [53] use a segmented video to create a background panorama in their Tour into the Video project. The offline approach registers images using a pair-wise least squares minimization of a rotation and zoom model, but does not handle the accumulation of error over time and only creates partial panoramas. Finally, the VideoBrush application from Sawhney et al.'s work [92, 91] can create 2D scanned mosaics as well as spherical panoramas from key frames selected out of a video sequence.

The most recent panorama work focused on automatic detection of panoramas. Brown and Lowe [14] presented a method for identifying panoramas from sets of images and automatically stitching them, up to full spherical panoramas. Along a similar vein, Snavely et al.'s Photo Tourism project [101] registers arbitrary

images in a persistent 3D model, without explicitly creating panoramas. These techniques are very powerful for processing sets of images, but that power comes at the price of speed as their performance does not scale well to large sets / video sequences.

A few systems have been developed that attempt to provide online feedback to allow users to direct their panorama acquisition. The VideoBrush application [92] computes a quick panorama approximation by doing only local refinement of a simple 2D translation model. While this enables quick previews, the oversimplified model creates very large distortions. The Panoramic Viewfinder [10] uses an ultramobile PC to present a user interface for panorama acquisition that shows the estimated panorama at that time. The system achieves 4Hz on a desktop PC, so processes only individual images rather than video sequences, and once all the images are acquired, the final panorama still requires offline global optimization.

Finally, there are many commercial products that do various forms of panorama stitching. They can be roughly grouped based on a few features. Many of these products support automatic stitching [84, 7, 1, 3, 112, 2, 109, 71, 5, 73, 77, 78, 103], though often with the opportunity for user adjustment if necessary. A few products exist that can handle 180° fisheye images [103, 74, 112, 2], while others can create full spherical panoramas from regular images [84, 7, 72, 112, 2, 24, 59, 74, 103]. In general, all the available products work by offline processing of a set or

sets of individual images acquired by a camera on a tripod. There are currently no products that operate on video, or provide real-time feedback.

In the context of augmented reality, there is ample previous work on landmark vision-based and hybrid tracking systems. The basic approach in general is to use vision-based methods for landmark feature recognition, combined with gyroscopes for robustness. An earlier system that uses a silhouette of the horizon as a stable landmark for vision only orientation tracking was presented by Behringer [12]. More recently, Satoh et al. [90] presented an outdoor orientation tracking system that uses user-specified patches of image texture as landmarks, fused with a gyroscope. You and Neumann [117] demonstrated a position and orientation tracker that uses offline acquired landmark features and a gyroscope in an Extended Kalman Filter framework. Most recently, Reitmayr and Drummond [86] introduced a robust 6DOF outdoor hybrid tracking system that matches video frames to a pre-acquired scene model. The limitation of each of these systems it that they depend on offline measurement of the scene to be tracked before they can be used. This requirement sets up a barrier to entry that hinders casual use of these tracking solutions.

Most similar to Envisor is the work of Montiel and Davison in visual compassing [66]. They extend previous work on single camera simultaneous localization and mapping (SLAM) [25], using a complex Extended Kalman Filter formulation

of the tracking problem to compute orientation from dynamically acquired landmark features. This work is significantly different from the algorithm I present, primarily in that my approach utilizes a more modular design, combining two configurable tracking modalities to achieve similar tracking performance. Additionally, the use of RANSAC and a larger number of simple features per frame suggests that Envisor will exhibit greater robustness to dynamic scene elements, though a direct comparison is not available.

In conclusion, Envisor is a novel application for the online creation of environment maps with a hand-held camera. The vision-based orientation tracking is a novel formulation of frame to frame and landmark tracking techniques with competitive performance characteristics to established approaches. The environment map construction differs significantly from existing panorama stitchers by registering full video sequences into spherical panoramas, in linear time complexity and constant space complexity, and without the need for post-processing or user input.

3.5.2 Vision-based Tracking

Similar to the tracking done in the GroundCam, the vision-based tracking for camera orientation uses the optical flow of a sparse set of features. Initial features are found using Shi and Tomasi's [95] good features operator, which

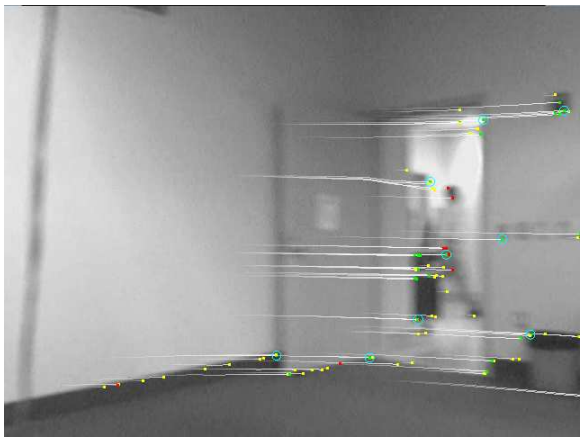

Figure 3.13: A video frame with the features overlaid. Green features have been inliers for many frames, yellow have been inliers for a few frames, and red are outliers. Features with circles around them are landmarks. The white lines show each features' recent history.

greedily selects a set of "corner" features, where corners are defined as image patches with strong gradients in two directions. The motion of these features between consecutive frames is determined using a pyramidal version of Lucas and Kanade's optical flow algorithm [62], which uses a hierarchy of different resolution images to efficiently match sparse image patches between two frames. See Figure 3.13 for a visualization of these features. As features are lost (moved out of the field of view, or could not be tracked), new features are added incrementally when the number of features drops below a threshold.

One time offline camera calibration is done using Zhang's algorithm [118], which measures the camera's focal length, center point, skew, and radial distortion. The distortion parameters are used to correct the position of features in

the image, as well as to undistort each frame on the GPU creating an idealized pinhole projection where the straightness of lines is maintained. The advantages of this model are that an OpenGL camera can be made to match the real camera's properties, and that it makes it trivially easy to unproject 2D points on the image plane into 3D points on the viewing sphere [35]. To do this unprojection, given a 2D point $(x, y)^T$ in normalized coordinates ($[-1..1]$),

$$(X', Y', Z')^T = ((C_r - C_l)x, (C_t - C_b)y, -C_n)^T \tag{3.9}$$

$$(X, Y, Z)^T = \frac{(X', Y', Z')^T}{||(X', Y', Z')||} \tag{3.10}$$

where C is the camera's intrinsic parameters, as a frustum defined in the OpenGL style (C_r, C_l, C_t, C_b, C_n are the right, left, top, bottom and near parameters respectively).

Given a set of 2D image feature correspondences between two frames, there are a variety of ways to recover camera rotation. The most straightforward is to state the solution as an energy minimization problem. The function to be minimized has three variables, the three degrees of freedom of the camera, and the error is the distance between the updated points from the previous frame with the new points from the current frame:

$$f(\theta, \phi, \psi) = \sum_i ||(x'_i, y'_i, z'_i)^T - M(\theta, \phi, \psi)(x_i, y_i, z_i)^T|| \tag{3.11}$$

where θ is yaw (rotation about the up or y axis), ϕ is pitch (rotation about the right or x axis), and ψ is roll (rotation about the z axis, the negative of the viewing direction in a right-handed coordinate system). $(x_i, y_i, z_i)^T$ is the i-th point in the previous frame, and $(x'_i, y'_i, z'_i)^T$ is from the current frame. Finally, $M(\theta, \phi, \psi)$ is the 3x3 rotation matrix.

This error function can be minimized using a standard solver such as gradient descent, using the previous frame's solution as an initial guess. The downsides to this approach are that it is iterative and may be distracted by nearby local minima. Also, an Euler angle representation such as this necessitates specifying a particular order of rotations, and can become confused around the singularity points. This can be avoided by always minimizing the relative orientation, which should always be close to zero, and then integrating these rotations.

Another standard approach is to construct a system of linear equations representing the rotation applied to the points, then solving the system using a technique like singular value decomposition. In this approach, since rotations are not linear operations, the system must be constructed to solve for the 9 elements of the 3x3 rotation matrix. The system of equations is setup as

$$x'_i = M_{1,1}x_i + M_{1,2}y_i + M_{1,3}z_i \quad (3.12)$$

$$y'_i = M_{2,1}x_i + M_{2,2}y_i + M_{2,3}z_i \quad (3.13)$$

$$z'_i = M_{3,1}x_i + M_{3,2}y_i + M_{3,3}z_i \quad (3.14)$$

Systems of hundreds of equations are created in standard cases. Once a solution is acquired through SVD, an iterative minimizer is often used to refine the result as a final step. Also, because the solution is generated for the elements of a 3x3 matrix, it is not actually guaranteed that the resulting matrix will be a valid rotation matrix, as the column vectors are likely to not be orthogonal. Therefore, explicit orthogonalization of the output matrix is required.

None of these techniques for computing orientation fit in with my goals for a tracking framework. Easy integration within a RANSAC [34] implementation is critical, as outliers are very likely in dynamic, real-world environments. Therefore, Envisor employs a technique that can be quickly used to generate RANSAC estimates and then used to refine the estimate from a large set of inliers, and it needs to be fast to evaluate many times per frame. Horn's formulation of the absolute orientation problem [48] matches this requirement, and in addition does not require further refinement or normalization. The algorithm takes two sets of points and finds the optimal quaternion rotation that maps the first set to the

second, in a single step. It accomplishes this by finding the first eigenvector of the following matrix,

$$M = \sum_i M_i \tag{3.15}$$

$$M_i = \begin{bmatrix} a_i & b_i & c_i & d_i \\ e_i & f_i & g_i & h_i \\ i_i & j_i & k_i & l_i \\ m_i & n_i & o_i & p_i \end{bmatrix} \tag{3.16}$$

$$a_i = x_i x'_i + y_i y'_i + z_i z'_i \tag{3.17}$$

$$b_i = z_i y'_i - y_i z'_i \tag{3.18}$$

$$c_i = x_i z'_i - z_i x'_i \tag{3.19}$$

$$d_i = y_i x'_i - x_i y'_i \tag{3.20}$$

$$e_i = z_i y'_i - y_i z'_i \tag{3.21}$$

$$f_i = x_i x'_i - y_i y'_i - z_i z'_i \tag{3.22}$$

$$g_i = x_i y'_i + y_i x'_i \tag{3.23}$$

$$h_i = z_i x'_i + x_i z'_i \tag{3.24}$$

$$i_i = x_i z'_i - z_i x'_i \tag{3.25}$$

$$j_i = x_i y'_i + y_i x'_i \tag{3.26}$$

$$k_i = y_i y'_i - z_i z'_i - x_i x'_i \tag{3.27}$$

$$l_i = y_i z'_i + z_i y'_i \tag{3.28}$$

$$m_i = y_i x'_i - x_i y'_i \tag{3.29}$$

$$n_i = z_i x'_i + x_i z'_i \tag{3.30}$$

$$o_i = y_i z'_i + z_i y'_i \tag{3.31}$$

$$p_i = z_i z'_i - x_i x'_i - y_i y'_i \tag{3.32}$$

The resulting eigenvector is the quaternion that optimally rotates the first set of points into the second, which in the context of orientation tracking, is the delta rotation between the two frames.

This technique has many attractive properties. First of all, the additional contribution of each point is part of the sum of a 4x4 matrix, so for any number of points the solution requires the same amount of work to compute, as compared to the energy minimization or linear system approaches, which scale linearly with the number of points. This also means it can be easily integrated into a RANSAC implementation as both the estimation step on a small number of points and the refinement step for a large set of points. Since this algorithm always yields the optimal orientation, local minima are not a concern and a good initial guess is not necessary. Finally, as the function returns a quaternion, order of rotations and

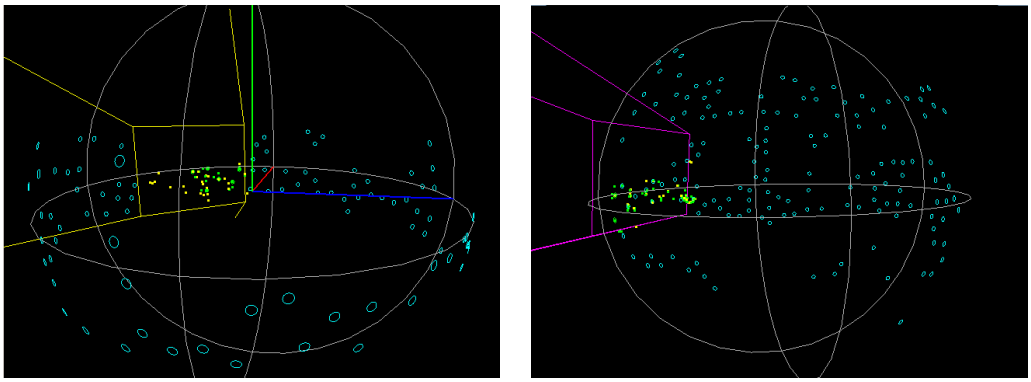

Figure 3.14: Two examples maps of landmarks distributed around the camera's position. Landmarks are blue circles, and the yellow or purple frustum represents the camera's current field of view. In the camera's field of view, the current set of tracked features can be seen in green, yellow and red.

singularities are not a concern and normalization, while generally unnecessary, is a simple matter of dividing by the magnitude of the quaternion.

3.5.3 Landmark Features

Augmentation of the frame to frame tracking with landmark features is important to combat drift during long tracking runs. Drift is a problem because the frame to frame tracking actually measures the camera's angular velocity over a short time, which is then integrated to get the orientation. This integration accumulates small errors in each update, so over time the difference between the actual orientation and the estimated orientation can become unbounded. Identifying and reusing absolutely positioned landmark features can combat this problem

by providing a periodic direct measurement of absolute orientation, rather than angular velocity.

Existing landmark based tracking systems [25] use some sort of uniquely identifiable feature such as large image patches, or SIFT [61] or SURF [11] features for landmarks. These heavyweight features are used to recognize when the tracker is revisiting previously seen regions, as well as during the frame to frame update of currently visible features. This approach can be simplified, as the uniquely identifiable nature of landmarks are not necessary for frame to frame updates – since landmarks will agree with the motion of the rest of the scene, they will be inliers in lightweight frame to frame tracking result discussed earlier. Therefore, optical flow based tracking is sufficient for the feature update step, and the utility of landmarks is only to uniquely identify features. Additionally, since a landmark feature does not change from when it starts being tracked to when it leaves the field of view, the landmark identification only needs to happen during feature reinitialization.

The manner in which landmark tracking is integrated into the vision based orientation tracking is as follows. Frame to frame updates are unchanged from how they are described in the previous section. Features that are inliers for a certain number of consecutive frames are promoted to landmarks if they are far enough apart from pre-existing landmarks. A landmark feature is added to a set

of landmarks called the map (see Figure 3.14). Each landmark has its associated world coordinate direction vector, and a feature descriptor. When a landmark is created, the patch around it in the image is used to create a SURF feature, using the code provided by Bay, Tuytelaars, and Gool [11]. The result is a 64 float descriptor that can be used to uniquely identify that patch of image texture.

Once landmarks are in the map, they must be reacquired when they come back into the camera's field of view. If a landmark is expected to be in the field of view (by projecting known landmark locations to the camera's estimated orientation), the landmark feature is searched for in a small search region about its expected location. To do this, Shi and Tomasi's good features operator is used to find candidate points inside a small search region, and then SURF descriptors are computed for each of those features. These descriptors are compared to the landmark descriptor and if a match is found, the feature is linked to the landmark and entered into the tracker. After a certain number of times attempting to reacquire a landmark feature and failing, the landmark is determined to be lost and is removed from the map. Matching descriptors are determined by normalizing the two descriptors to have a magnitude of one and then computing the dot product, which is thresholded.

During tracking, the landmark features are used twice, once as part of the full set of frame to frame features, and separately to find an orientation estimate from

just the landmarks. This separate estimate uses the same algorithm as the frame to frame tracking, but instead of computing the rotation between each landmark's position in the previous and current frame, the rotation from the landmark's world stabilized position to the current frame position is generated. RANSAC is still applied, because while landmarks are assumed to be static features, they may still change – for example, a landmark feature may be on a parked car, but after some time the car may drive away and the landmark will have changed. Landmarks that are outliers a certain number of times will eventually be discarded.

It is important to note that landmarks still drift in a certain sense, but the implications are different than the drift from the frame to frame tracking. In particular, new features are still initialized with a position relative to previously acquired features, which means that new features will include any error from the current position estimate in their positions. Over time, as new features are created and then used for tracking to initialize more new features, this error will accumulate and create a map that diverges from the real physical layout of features. This is less of a problem for orientation tracking as the distance that can be traveled before old features are reacquired is limited. Regardless, the difference between this drift and the drift in frame to frame tracking is that the landmark orientation estimate will always be the same when viewing a set of previously

acquired features, whereas the integrated frame to frame estimate may give very different estimates each time an orientation is revisited.

3.5.4 Hybrid Tracking

For robust tracking, there are three tracking modalities that must be combined – the frame to frame vision tracking, the landmark vision tracking, and a gyroscope / compass orientation tracker (the InertiaCube 2). The combination of these trackers will retain their respective benefits and result in a higher quality final tracked result. The landmark vision tracking provides an absolute orientation, but is not available each frame as enough landmarks may not be visible, and may exhibit some random error due to the smaller number of features used to compute the orientation estimate. The frame to frame vision tracking provides an angular velocity estimate with a low amount of error assuming slow camera motion, but integration over time creates drift. Both vision based modalities depend on good image data, which can be lost under fast motion (due to motion blur), significant occlusion, or regions of no texture (e.g., the sky). The use of an InertiaCube 2 provides an additional absolute orientation measurement that is always available, but exhibits significant systematic error due to the influence of ambient magnetic fields.

Envisor uses a heuristic approach to combine these tracking inputs. In the case that a landmark measurement is available, the current orientation is set to that measurement. When insufficient landmark data is available, the frame to frame relative measurement is added to the previous orientation. In the case that neither is available, the relative rotation measured by the InertiaCube 2 is used in place of the frame to frame measurement. To detect the case that the camera is completely occluded or distracted by large dynamic motion, the vision update (landmarks or frame to frame) is compared to the measurement of the InertiaCube 2. If the difference between the two is above a threshold for multiple frames (adjusted to account for the systematic errors in the InertiaCube 2 measurements), then the vision update is determined to be bad and the InertiaCube 2's update is used.

I also experimented with an implementation of an Extended Kalman Filter [113] designed for orientation tracking, which uses the measurements from all sensors as they are available and fuses them continuously. The EKF design was based on the work presented by Azuma [8]. However, an EKF and other similar filters are not particularly well-suited to Envisor's environment map construction goal. Errors in tracking cause visible discontinuities in registration of images in the final environment map, and the particular formulation of an EKF ensures that errors will be smoothed out over multiple frames, lengthening their effect and increasing the span of their impact in the final output. Additionally, the significant error

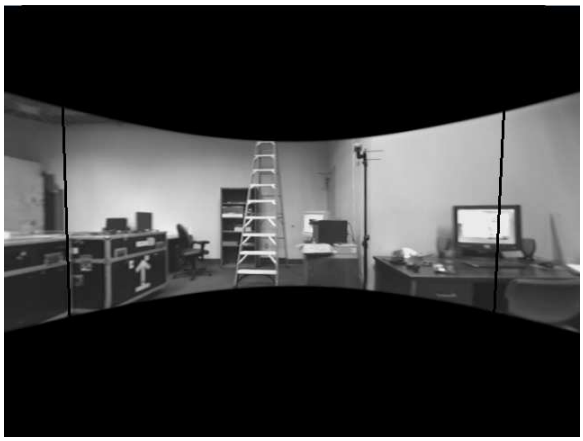

Figure 3.15: A portion of the cubemap after the camera has been rotated to complete a circle. The black lines near the left and right sides of the image mark the edges of the cubemap face.

of the InertiaCube 2 will exert some influence in every frame, further increasing registration discontinuities. Finally, the lag in the response of an EKF creates visible errors every frame, as the EKF does not directly reflect the most recent relative rotation measurement, which (under good tracking conditions) is the best estimate of the relative alignment of two consecutive frames.

3.5.5 Cubemap Projection

Since creation of an environment map is the goal of this work, a blank cubemap is the starting point. Image data from the tracked video stream is then projected into this cubemap, creating an environment map. Accomplishing the projection is a straightforward procedure. First, the cubemap is made renderable by attaching each face to separate color attachment points of a framebuffer object (FBO). This

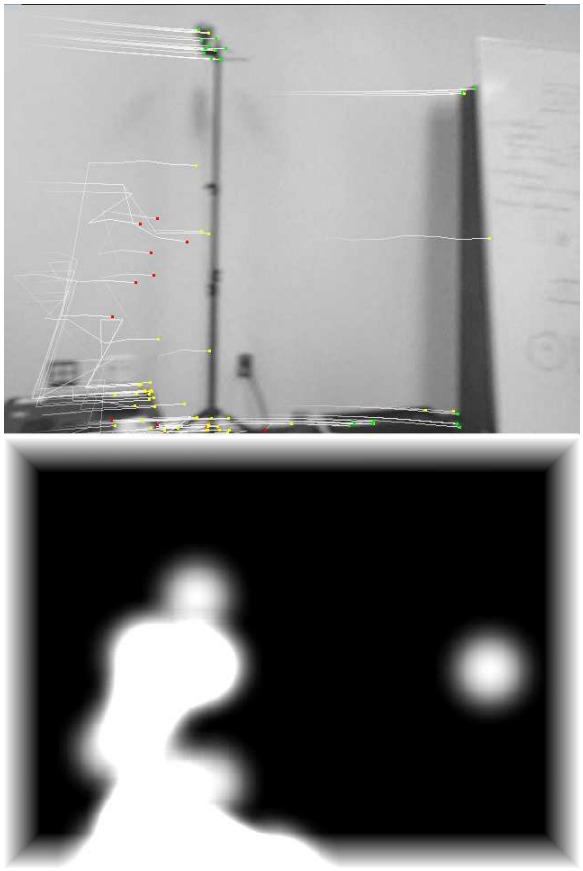

Figure 3.16: An example video frame with inlier (yellow and green) and outlier (red) features marked, and the associated alpha mask used for projection. As features are masked out until they have been inliers for multiple consecutive frames, some of the yellow features are still masked out.

allows projection of the video data to happen by making the FBO the current rendering context and drawing geometry texture mapped with the video image into the cubemap face directly. It is necessary then to figure out where to draw the geometry. As was mentioned earlier, because of the information gained from the offline camera calibration, an OpenGL camera can be constructed that has the same intrinsic parameters as the real camera. Given a camera intrinsic matrix of

$$\begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.33}$$

(assuming zero skew) then the corresponding OpenGL projection matrix is (modified from Rehman [85])

$$\begin{bmatrix} \frac{2f_x}{w} & 0 & \frac{2u_0}{w-1} & 0 \\ 0 & \frac{2f_y}{h} & \frac{2v_0}{h-1} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.34}$$

for a user-specified near and far plane (n and f respectively) and given image width and height in pixels (w and h).

By orienting this camera with the estimated orientation, the portions of the scene covered by the video image can easily be realized. Computing the directions of the four corners of the camera's field of view gives the coordinates of the quad that needs to be drawn to project the video image into the cubemap, as a cubemap is indexed by direction vectors. Therefore, the four corners are tested to see which cubemap faces they fall on, and then each face that is affected is attached to the OpenGL context (via the FBO) and a quad is drawn with the corner coordinates equal to the computed world direction vectors (see Figure 3.15).

To reduce edge artifacts and allow regions of the image to be selectively projected, an alpha mask texture is applied to the quad being drawn into the cubemap as well (see Figure 3.16). Initially the mask is set entirely to black, and the borders are faded to white at the very edges. When rendering, blending is turned on with a blend equation that makes black mask pixels opaque and white pixels transparent.

The mask is further used to selectively project only portions of the video image that are determined to be static background as opposed to dynamic foreground objects (e.g., people, cars, etc.). This determination is made based on the features from the frame to frame tracking. Because of the way features are added to the scene, any portion of the image that is tracked by features has significant texture. If a feature is an outlier, then it is tracking some scene element that is moving

contrary to the camera motion, and is labeled as a dynamic object which should not be projected. Inlier features move with the camera and so are labeled as static elements of the scene which are projected. Even though regions of the image with uniform texture will not have features tracking them, they should be projected as well. It is better to be slightly conservative about classification, as if a portion of the image is erroneously determined to be dynamic, it likely will not be for long and can be filled in shortly thereafter.

To remove dynamic regions from the projected image, the mask is modified by drawing a Gaussian splats for each outlier feature. These splats are blended together, so in regions with lots of outliers, the entire region is masked out. Then when the video image is drawn into the cubemap, these regions will be left out.

3.5.6 Gap Avoidance and Filling

Creating full environment maps means dealing with gaps and there are two components to effectively do so. First is to enable the user to avoid gaps in the environment acquisition process, and the second is to fill in the gaps that remain.

Gap Avoidance

Enabling user avoidance of gaps means giving the user useful feedback during the acquisition process that allows him or her to more intelligently direct the

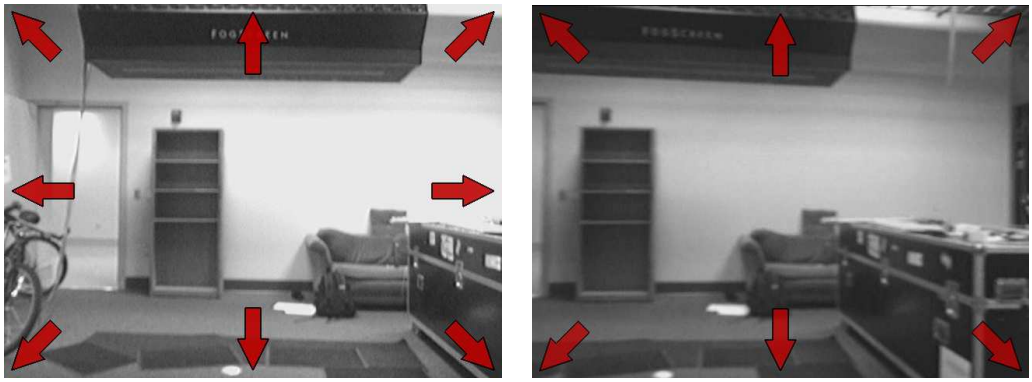

Figure 3.17: On-screen feedback in the form of arrows around the video image direct the user's acquisition. *Left to right:* (a) Before panning the camera, all directions need to be acquired still. (b) After the camera has completed a circular path, the left and right arrows are gone.

camera. In a wearable context, a user interface that simply presents the user with the cubemap and allows panning around the view is too complex, requiring significant cognitive load and manipulation of the wearable input device. Instead, the feedback should be more tailored to low cognitive load with no interaction requirement. Envisor presents a passive display of a set of arrows around the current view that indicate which directions gaps are present along. See Figure 3.17 for an example image. As there are fewer unfilled pixels along a certain direction, that arrow will become more transparent until it disappears when all the gaps are filled.

Creating these arrows requires efficiently sampling the pixels along each direction and testing for gaps (to make this determination easy, projected pixels have alpha values of 0, while gap pixels are greater than 0) To sample the cubemap

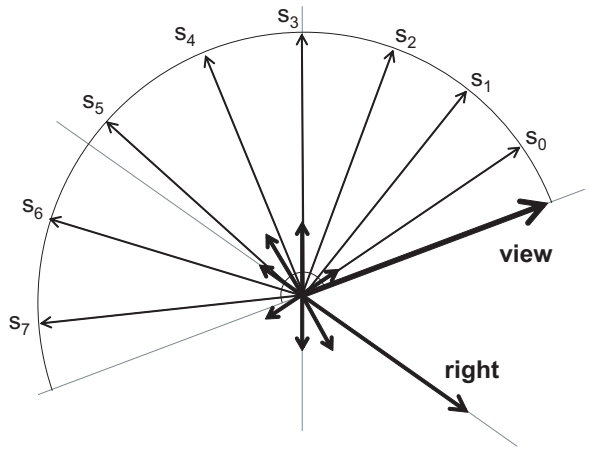

Figure 3.18: Gap searching proceeds by rotating the view vector around each of 8 evenly-spaced vectors perpendicular to the view direction. Here, the view vector is rotated about the right vector. Along each sample vector s_i, the cubemap is sampled to see if there is a gap.

along each direction, from the camera's extrinsic pose the camera's view, right and up vectors (d, r and u respectively) can be extracted. The right and up vectors can be used to create 8 cardinal directions around the viewing direction. Then for each of these axes, the view vector is rotated about the axis incrementally in the range of [0..180] degrees (see Figure 3.18).

For the set of direction vectors about the view vector, D,

$$D = \begin{bmatrix} -u \\ r-u \\ r \\ r+u \\ u \\ u-r \\ -r \\ -r-u \end{bmatrix} \tag{3.35}$$

S_i is the set of sample vectors for each direction $D_i \in D$,

$$S_i = \{s_i^j : s_i^j = M_R(j, D_i)d, j \in [0..180]\} \tag{3.36}$$

where $M_R(\theta, \vec{v})$ is the rotation matrix corresponding to a rotation of θ degrees about the vector $\vec{v}$.

For each direction i, to determine the prevalence of gaps along that direction, the cubemap is sampled at each sample vector in S_i, and the average of these samples is taken,

$$w_i = \frac{1}{n} \sum_j \text{gap}(s_i^j) \tag{3.37}$$

where each gap is a function that samples the cubemap at the direction and returns 1 if it is a gap and 0 if it is filled.

An straightforward way to implement this would be to read back the pixels from the cubemap along the sample vector directions, but this pixel readback is extremely slow as it breaks GPU pipelining by introducing a stall. Since the panorama construction has heavy CPU and GPU use, good pipelining is critical for performance, so keeping the computation on the GPU is important. To accomplish this, the cubemap is sampled by drawing a point with the sample vector as its texture coordinate. The series of samples are combined by blending with an additive blend function, and a simple fragment shader is used to test the sampled values to see if they are gaps or not and output 1 or 0 appropriately. The result is 8 pixels in an offscreen buffer, each with an alpha value that represents the weight for the corresponding direction. When drawing the arrows in the user interface, the geometry simply has this texture applied with the appropriate pixel's coordinate passed as the texture coordinates for the entire arrow.

Gap Filling

Even with user feedback, gaps are still likely to occur, though they may be smaller. It is important to do something to fill these gaps, as they create a very distraction visual artifact (the appearance of obvious "holes" in the environment

Figure 3.19: Examples of the texture diffusion for incomplete environment maps. The black lines show the edges of the cubemap faces.

map and shaded geometry). There are a number of established image inpainting algorithms [13, 23] that are suited to this task. However, most are offline algorithms that require extensive computation and often take seconds to minutes per image, depending on the size of the gaps. That sort of performance is not acceptable in an online solution like this. The fast image inpainting algorithm of Oliveira et al. [69] is better, but it still requires user input to place diffusion boundaries. The technique can be adapted to be fully automatic however, with just the diffusion component and no boundaries. The advantages of this approach are numerous. First, it can easily be implemented on the GPU, which is inline with the goal of avoiding readback of GPU data to the CPU. Second, it is an incremental algorithm, so a little work can be done each frame without impacting the overall performance. And finally, with some modifications it can be made to handle well the case that some gap filling is done and then some of the gap

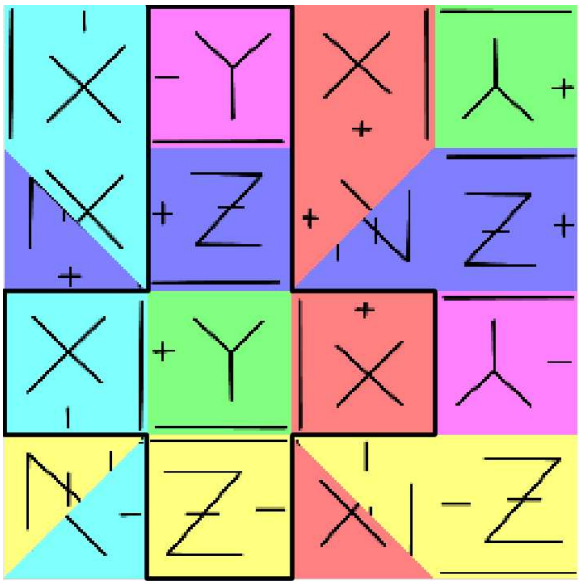

Figure 3.20: Layout of the cubemap faces in the atlas texture. The black outline marks where the cubemap faces are sampled from when applying the texture diffusion.

pixels are filled in from projected video images. As this will happen often, it is important for the gap filling to be able to respond to the new data immediately.

The basic approach is to use a fragment shader that implements a diffusion function. Pseudocode for the shader is as follows:

```
center = sample texture at center of kernel
avg = 0
count = 0
dist = MAX
for each pixel in kernel
{
  samp = sample texture at kernel pixel
  dist = min( dist, samp.a )
  if( samp.a <= center.a )
  {
    avg += samp
    count += 1
```

```
  }
}
avg /= count;
avg.a = dist + 1
if( count == 0 || center.a < avg.a )
  output = center
else
  output = avg
```

This shader operates in the following way. First, all pixels in the cubemap are initialized with an alpha value of 1, while projected pixels are set to have an alpha value of 0. During diffusion, the alpha value is used to encode the distance from a filled pixel – pixels with alpha 0 are filled, pixels with alpha 1 have not been diffused into yet, and alpha values in between designate the distance that pixel is from the nearest filled pixel. The shader computes an RGB value that is the average of all the surrounding pixels that have a distance less than or equal to the distance of the center pixel. It also finds the minimum distance to the center pixel. The new output value is then the average RGB and the minimum distance plus one for the alpha value.

One advantage of this implementation is that it does not repeatedly recompute pixels that have already been diffused, which would result in their colors slowly fading due to rounding errors. Also, by using the distance as part of the criteria for deciding to update a pixel, when new regions of the image are filled, the new smaller distances of nearby pixels will insure they get updated. The result of this

sort of propagation is the generation of a rough Voronoi-style diagram, except that the colors from the regions are extensively blurred together. See Figure 3.19 for examples.

The remaining difficulty is to implement the diffusion on a cubemap, with correct diffusion between cubemap faces. The solution is to do the diffusion in a regular 2D texture that has been filled in with the cubemap faces laid out so the boundaries of the embedded faces meet as they do on the actual cube. The particular layout used is from Gu [43], which can be seen in Figure 3.20. Before a diffusion step is computed, the cubemap is drawn into the atlas texture as shown. Then the diffusion is computed by drawing each face back into the cubemap with the diffusion shader enabled, sampling from the atlas texture. This way, pixels at the boundary of one face will correctly sample from the abutting faces as needed.

To improve performance, the texture diffusion is actually computed on a sub-sampled cubemap with faces sized 64x64 pixels (while the full resolution cubemap is 512x512 or greater). As the diffused texture is very low frequency, this does not impact the visual quality.

3.5.7 Application to Visualization

The final step of this technique is to use the acquired environment maps to render virtual geometry as if it were illuminated by the surrounding physical scene.

This is accomplished with the same basic technique as outlined in Section 2.2.1. A directly applied environment map simulates the response of a completely specular material. To simulate the response of a diffuse material, the environment map is blurred, so each pixel is the averaged contribution of many pixels in the original environment map. Different from the description in Section 2.2.1 is the fact that a cubemap texture is being applied in this case, as opposed to a 2D texture that contains a spheremap. A cubemap texture also supports mipmap generation in OpenGL, but the algorithm does not blur across the cubemap faces, so the result of a heavily blurred environment map is a cube with each face a different color.

To compute a correct cubemap blurring effect, the texture atlas approach described in Section 3.5.5 is used again. The cubemap is drawn into the texture atlas, and then a 2-pass blur shader is applied. The blur shader works by computing a 1D blur in each pass, first horizontal and then vertical, resulting in a 2D triangle blur. The advantage of this approach is that the number of texture samples needed for an $N \times N$ filter is $2N$ instead of N^2. A 1D blur shader is very simple – it samples the texture multiple times along one axis and averages the result. To further reduce the number of texture samples needed, each sample is made on the boundary between two texels and the hardware's texture filtering is turned on, thereby returning the average of two pixels with one sample. This way, a 15×15 filter is implemented very efficiently, with only 7 samples. These

Figure 3.21: Virtual geometry shaded using the acquired environment map from Figure 3.25(a). The environment map is filtered first to create the appearance of a silver material with a glossy finish. The teapot is superimposed over the acquired environment map.

two passes are applied to the atlas, which is then put back into a cubemap for application as a reflectance map.

To adjust the appearance of the virtual geometry, the OpenGL diffuse color response is used to modulate the intensity and hue of the applied reflectance map. Different levels of shininess are created by adjusting the amount of blur applied to the environment map. An example of a glossy virtual object being rendered inside the scene acquired in Figure 3.25(a) can be seen in Figure 3.21.

3.5.8 Panorama Building Results

See Figure 3.25(a) for an example of an environment map constructed by Envisor from a camera on a tripod. While a few misregistrations are evident,

particularly in regions that are close to the camera, they are minor. Since the camera was on a tripod, it was unable to acquire the scene directly above or below its position, so the texture diffusion process has filled in those gaps.

Performance

This work was tested on three machines: a desktop with a 2.1GHz AMD Athlon CPU and an NVIDIA GeForce FX 6200 graphics card, another desktop with a 3.0GHz Intel Xeon and an NVIDIA GeForce 7800 GS, and a laptop with a 2.0GHz Intel Pentium M CPU and an NVIDIA GeForce Go 6600. The camera used was a Unibrain Fire-i400 camera with a 4mm lens. In general, we experience around 15 frames per second in the testing application, which, for testing convenience, runs off of a pre-recorded MPEG encoded video and accompanying metadata file. See Table 3.1 for more detailed timing data. The GPU implemented steps of the technique are not accurately represented in the timing data because of the difficulty in accurately measuring the stages separately given the GPU's heavy pipelining. Because of the way the GPU stages are implemented however, they are able to completely overlap the CPU portions of the algorithm, and so do not impact the final per-frame running time. This was confirmed by comparing timing data with and without the GPU components of the application, which were not significantly different.

stage	**Athlon**	**Xeon**	**Pentium**
video decoding	13.0	8.5	11.2
undistortion	0.3	0.3	0.3
preprocessing total	13.3	8.8	11.5
KLT tracking	24.5	15.7	28.6
RANSAC	0.3	0.5	0.7
landmarks	40.1	33.9	24.5
tracking total	65.2	50.5	54.2
cubemap update	2.7	2.3	3.7
total	81.2	61.6	69.5

Table 3.1: Average times (in ms) of the various stages of Envisor, on three computers. The preprocessing and tracking are broken up into their component stages, and timings are presented for each stage as well as the frame total. The final total is the start to finish for each frame of the test application.

Unfortunately, some of the steps that take the most time are fixed costs. When SURF features are used for landmarks, there is a step to compute the integral image of the video frame. This expensive operation is performed even if only one SURF descriptor is needed. However, better performance can be achieved by not doing every step of the algorithm every frame. For example, only looking for new landmarks every 3 frames yields a large improvement to the average framerate. Similarly, one of the slowest components of the KLT tracking is initialization of new features, which can also be done every few frames. By distributing periodic workloads across frames (e.g., interleaving KLT feature initialization and landmark searching) the performance can be increased. How aggressively this can be done depends on the expected speed of camera motion and dynamic nature of the scene. For faster camera motion, features will be in the field of view for fewer

frames, and so all tracking operations must happen frequently. More dynamic scenes will need better robustness to outliers, which requires more features processed more frequently. These considerations are important on a per-application basis.

The ability to tune the performance of the frame to frame and landmark feature tracking separately is an additional advantage of the approach to tracking used in this work. The level of configurability to particular application needs is very high – for example, in a scene with consistent strong texture and very few dynamic elements, landmark tracking updates by themselves may be sufficient, without relying on frame to frame measurements to fill gaps. Alternately, applications such as fully immersive VR or games that only need angular velocity input could use only the frame to frame updates without the landmark corrections. This advantage is in contrast to black box tracking solutions that only rely on one tracking modality.

Tracking

For testing purposes, we used a selection of different cameras: a Unibrain Fire-i, a Unibrain Fire-i400, and a Point Grey FireFlyMV. They cover a range from consumer level to mid-range lab cameras. Mostly, we used the Fire-i400 because it has the widest field of view, at 51°. A wider field of view means the

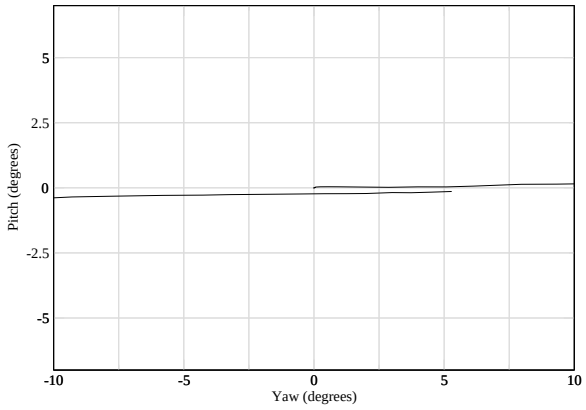

Figure 3.22: Accuracy of the frame to frame tracking. After rotating the camera in a full circle on a tripod, the estimated orientation from the integration of the relative rotation measurements has error less than 0.2°.

feature tracking is less likely to get distracted by large occluders such as a person walking by, and that potentially faster motion can be tracked. It also improves the robustness of the tracking against regions of uniform texture, as such regions will have to be much larger to fill the camera's field of view. However, the most significant impact of the wider field of view was that it made environment map construction much faster as fewer sweeps around the scene were needed, which definitely improved the usability of Envisor.

The accuracy of the tracking was tested by moving the camera through a circular sweep at roughly 20° per second, on a tripod in a large room under favorable tracking conditions. Figure 3.22 shows what the estimated orientation of the camera was at the start and end of the sweep. The tracking is accurate enough that after completing the loop, the camera is off the original orientation by 0.2°.

This is significant, as it means that the characteristic discontinuity at the end of a closed-loop in panorama stitching, which generally requires a global refinement to the stitching, is not as important for Envisor. This accuracy is also sufficient for the landmarks from the beginning of the sweep to be reclaimed as they come back into view, as shown in Figure 3.23. This means that as long as good tracking conditions are maintained, Envisor is capable of long-term drift-free orientation tracking. However, because the landmark features are originally initialized off the frame to frame tracking results, if the relative orientation gets distracted the landmarks will incorporate that error into their positions. If this error is too large, the reacquisition of old landmark features will fail, causing them to be discarded and new landmarks acquired in their place. Because of these limitations, the vision based tracking alone is not robust to poor tracking conditions such as total occlusion. Under good conditions, the tracking is successful indefinitely (see Figure 3.24).

Theoretically, the maximum rate of rotation that can be tracked is limited by the camera's field of view and the framerate. For our testing setup, we used a 51° field of view camera and had a framerate of 10Hz. If half the image needs to remain in the field of view between consecutive frames for tracking to succeed, then that results in a theoretical maximum angular velocity of 255° per second. Realistically, motion blur causes optical flow to fail at much lower speeds. In practice, we find

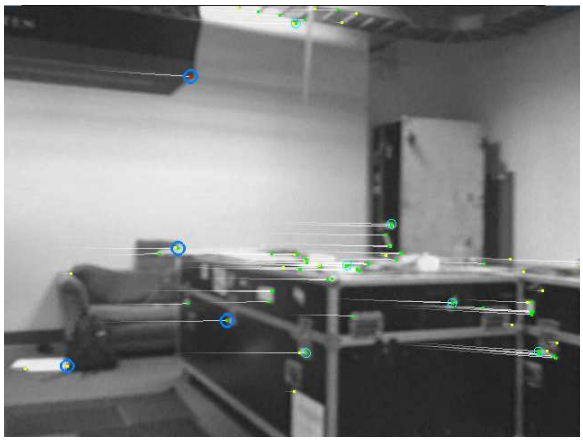

Figure 3.23: The tracking accuracy is sufficient to close circular paths without the characteristic discontinuity. Here, a circular sweep is coming to an end, and the landmarks put in the map at the start are successfully being reinitialized as they come back into the field of view (indicated by the darker, heavier circles).

that angular velocities of up to 60° per second can be tracked by the frame to frame tracking, while landmark tracking is successful at angular velocities of up to 30° per second. The reasons for the slower maximum for landmark tracking are that the landmarks require features to be tracked successfully for a number of consecutive frames before they can be promoted to landmark status, and that the blurring decreases the quality of the computed SURF descriptor, which interferes with reinitialization. The most effective way to increase the maximum trackable angular velocity is to lower the camera's exposure time, reducing the motion blur effect. High speed cameras and brightly illuminated scenes will both improve this result.

As Envisor is solely an orientation tracker, it makes the implicit assumption that the camera experiences no translation – i.e., that it rotates about the camera's optical center. For a head-worn or hand-held camera, this assumption will not be exactly correct. The impact of small translations on the quality of the tracking depends on the distance to the objects being tracked. For an indoor environment such as a lab or office with objects within 5 to 10 feet of the camera, the small translations from a hand-held camera will cause visible discontinuities when closing circle sweeps. Locally the resulting environment map appears smooth, but global errors become a problem (see Figure 3.25(c)). For this reason, our indoor tests use a tripod – unfortunately, high quality indoor environment map construction either requires a tripod or a tracking model that takes translation into account. Outdoors, where objects are often 20 feet or more away, the translations from hand-held camera motion have a much smaller impact (see Figure 3.26).

Environment Mapping

The quality of the resulting environment map from Envisor depends heavily on the quality of the tracking data obtained. People are very sensitive to small registration errors when tracking results can be compared directly, side-by-side, as they are in an environment map at the borders between projected frames.

Figure 3.24: Snapshots of the feature tracking for hand-held camera motion over five minutes. *Left to right, top to bottom:* (a) Initial feature set. (b,c,d,e) Tracking maintained through various speeds and orientations, with landmark reacquisition. (f) The resulting partial environment map shows the quality of the tracking.

Visible gaps and jumps negatively affect the appearance of a panorama. While the tracking presented here is able to rely on a variety of different modalities, only the frame to frame relative updates create seamless blending within the environment map, as they directly compute the optimal transform between two frames. If there are errors in the tracking from bad video data or random noise in the landmark orientation measurement, discontinuities in the environment map will arise. However, depending on the target application, these discontinuities may not be a problem. For example, applications that use environment maps as a backdrop may find small errors acceptable. Shading of virtual geometry that is

Figure 3.25: Cylindrical projections of acquired environment maps. *Top to bottom:* (a) Using a tripod. (b) With automatic exposure and white balance enabled, creating visible discontinuities due to intensity and hue differences. (c) Carefully constructed with a hand-held camera in approximately 3 minutes. Small translations result in errors.

Figure 3.26: A panorama constructed outdoors with a hand-held camera.

not completely specular and smooth will also not be adversely affected by these errors.

One of the problems facing environment map construction is the changing exposure and white balance of automatically adjusting cameras. As a camera moves from a bright region to a darker one, or vice-versa, it takes some amount of time to adjust to the new illumination, which means that revisiting the same portion of a scene may result in different pixel values than were previously acquired. This problem is evident when the camera is re-swept over a region, and the border between the old and new data is clearly visible due to brightness and hue differences (see Figure 3.25(b)). In low dynamic range environments, such as an office with fluorescent lighting that creates significant ambient illumination, the camera's automatic adjustment can be turned off with no adverse effects (see Figure 3.25(a)). However, in high dynamic range environments such as outdoors or indoors with very localized light sources, the automatic exposure adjustment

is important for tracking because it ensures that the image features always have good contrast. Over or under exposure will reduce the quality of the computed optical flow, hurting tracking performance. Alternately, if a camera supports reading the adjusted parameters per-frame, a color model can be fit to these parameters that would allow manual normalization of the images on the CPU or GPU, so the tracking can always have an optimal exposure image, while the environment map always has normalized intensities. Unfortunately, our cameras do not support this feature.

The quality of the masking of dynamic portions of the scene in the environment map warrants examination as well. The assumption made is that dynamic objects will be tracked by outlier features which are then used to create the mask. However, the number of features definitely impacts the quality of this masking. If there are too few features being tracked, moving elements may only have one or two features on them, or possibly none at all. Also, a fast moving object will have more motion blur than the rest of the scene, and therefore will be less likely to be selected for new feature points. The result is that dynamic objects (or portions thereof) will still occasionally end up in the final environment map. Increasing the number of tracked features will reduce this effect, while a more robust solution would be to use a dense optical flow algorithm that could classify each pixel as inlier or outlier.

3.5.9 Error Analysis

It is important to examine the sources of error and failure modes of the orientation tracking. There are five ways error can be introduced: translation of the camera (violation of the rotation-only assumption), image noise (affecting the feature tracking), motion blur (also affecting feature tracking), lack of texture (causing poorly distributed features), and significant distractions (overwhelming RANSAC classification). Extremes of each of these conditions can cause tracking to completely fail due to error – too much translation and orientation estimate will diverge from real orientation, too much noise or motion blur and the feature tracking will produce random data, too little texture and there will be no features to track, too many / large distractions and the coherent estimate will not reflect the camera motion. How each source will degrade performance is worth examining more closely.

Translation

It is possible to approximate what the measured rotation will be for a translation of the camera. Let p be a 3D point at the center of the camera image plane at distance n from the camera, corresponding to a 3D point P that is distance D from the camera. If the camera undergoes a translation of distance T, perpendicular to the viewing direction, then the point P will appear to move T in

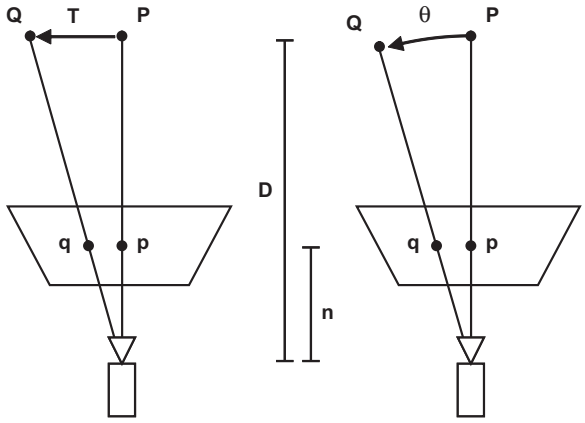

Figure 3.27: Comparison of camera translation and rotation on scene features. *Left to right:* (a) As the camera undergoes translation T, point P in the scene moves the same distance. The corresponding projected points p and q show the motion of the feature in the image. (b) Point q can also be generated by rotating the camera through angle θ.

the opposite direction (in camera coordinates), becoming Q, with corresponding image plane point q. See Figure 3.27 for an illustration. The distance the point in the image plane will have appeared to move is

$$\overline{pq} = \frac{nT}{D} \tag{3.38}$$

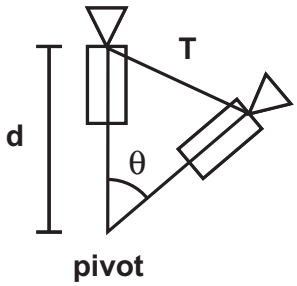

Figure 3.28: Induced translation from rotation of the camera about a pivot point that is offset from the optical center.

If the camera had instead rotated by θ degrees about an axis perpendicular to the viewing direction, the distance between p and q would instead be

$$\overline{pq} = n \tan \theta \tag{3.39}$$

Therefore, for a translation T, the apparent rotation θ can be computed as

$$\theta = \tan^{-1} \frac{T}{D} \tag{3.40}$$

While this only computes the apparent rotation of a point at the center of the image, it is a reasonable approximation of the resulting measured rotation for the entire image. The motion of features elsewhere in the image will be symmetric about the center in orientation, but the further points are from the center, the smaller their apparent rotation will be. Therefore, this approximation will slightly over-estimate the actual measurement under translation.

Pure translation is unlikely, however. More likely is that the camera will rotate about a point that is not the optical center, which will cause both rotation and translation simultaneously. Assume the pivot about which the camera rotates is distance d in front of the camera's optical center (negative values of d mean the pivot is behind the optical center). Then for a rotation of θ, the translation of the camera T will be

$$T = 2d \sin \frac{\theta}{2} \tag{3.41}$$

For an illustration, see Figure 3.28. This translation T will cause an additional rotation measurement of $\delta\theta$,

$$\delta\theta = \tan^{-1} \frac{2d \sin \frac{\theta}{2}}{D} \tag{3.42}$$

Thus, the total measured rotation θ_m will be

$$\theta_m = \theta + \delta\theta \tag{3.43}$$

$$\theta_m = \theta + \tan^{-1} \frac{2d \sin \frac{\theta}{2}}{D} \tag{3.44}$$

$$\tag{3.45}$$

This equation is important as it lets us predict the error due to translation of rotation of the camera about a point other than the optical center. For small values of θ, $\sin \theta \approx 1$ and for small values of x, $\tan^{-1} x \approx x$. Therefore, for small rotations,

$$\delta\theta \approx \frac{2d}{D} \tag{3.46}$$

$$\theta_m \approx \theta + \frac{2d}{D} \tag{3.47}$$

The significance of this approximation is that the error is only based on the ratio between the distance to the 3D feature point and the distance between the optical center and the pivot point. Therefore, smaller scenes can be effectively tracked only when the pivot point is closer to the optical center than is necessary to achieve similar tracking quality in large scenes. Also apparent from the equation is that if the optical center is in front of the pivot point, the measured rotation will be larger than the actual rotation, and if it is behind, the measured rotation will be smaller.

Tests with synthetic data were used to measure this effect directly, by rotating an off-center virtual camera inside a synthetic scene through a 360° circle. The synthetic scene consists of features regularly spaced on a 10m sphere every 5° in pan and tilt, resulting in 2592 distinct features. The camera's offset was varied from 5mm to 5m, for both positive and negative offsets, and the camera underwent 360 1° rotations for each offset. Results can be seen in Figure 3.29, which roughly agrees with the predicted model.

The results show that for a ratio of pivot offset to scene distance of 0.1, after a full circle at 1° increments, the error will be approximately 36°. However, for tripod or hand-held camera rotation, a pivot offset of 20cm or less is reasonable, in which case, the error is very small per frame, on the order of 0.002° in a 10m scene. After a full circle, the accumulated error in this case is expected to be on

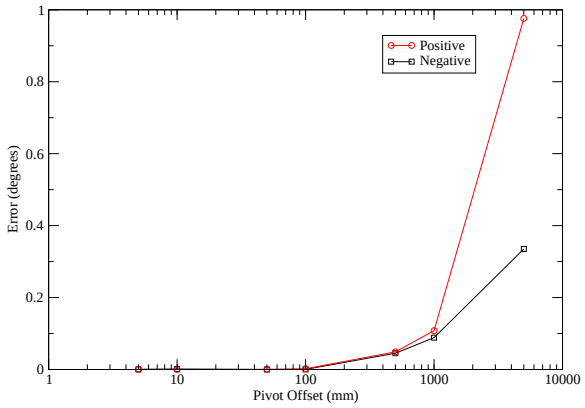

Figure 3.29: Error in rotation measurements in synthetic test of off-center rotation. Error is the average over 360 1° rotations.

the order of a few tenths of a degree. In real world scenarios, indoor scenes tend to range between 2m and 10m from the camera, versus outdoor scenes which range between 5m and 100m. A pivot offset of approximately 20cm (for a hand-held camera) results in a per-degree error range of 0.002° to 0.01° indoors, or 0.0002° to 0.004° outdoors. After a 360 degree panning rotation, the expected error indoors is on the order of 1°, versus 0.1° degrees outdoors. A camera on a tripod can expect a much smaller pivot offset of 5cm or less, which results in an expected total error of the order of 0.1° indoors or 0.01° outdoors.

To further confirm these results, controlled scenes were created with the camera pivoting about different points to get direct measurements with the actual Envisor software. In these tests, the camera was set on a tripod, surrounded by real objects (walls and chairs) a particular distance away. The camera was then slowly rotated

test	**pivot** (cm)	**scene** (m)	**measured error** (deg)	**predicted error** (deg)
A	3.5	1	6	12
B	12.3	1	33	44
C	3.5	2	2	6

Table 3.2: Measured and estimated errors for real tests with a camera rotating through a full circle about a pivot point offset from its optical center, in a nearby scene. The error is the yaw value of the orientation (pitch and tilt were negligible) when the camera is returned to the initial orientation.

through a full circle and the final error was measured. To determine the error, the camera was first lined up with an alignment mark on the tripod before rotating, and the rotation stopped when the alignment mark was reached again, resulting in a final orientation that was as close as possible to the original orientation. The computed orientation of the final camera pose was measured from Envisor, which was then interpreted as the error (as the orientation should have been the identity). This method gives a reasonable estimate of the orientation error. In test A, the camera pivot offset was 3.5cm and the scene was 1m away. In test B, the offset was 12.3cm in a 1m scene, and test C had a 3.5cm offset in a 2m scene. Results can be seen in Table 3.2.

Aside from the fact that the translation error model over-predicts the error, the measured errors are partially smaller than expected due to the difficulty in ensuring that features are only chosen from near-scene objects. In portions of the constructed scene, far field objects were partially visible and some features

ended up in those regions, meaning they did not exhibit the expected parallax of the near field objects. This has the effect of lessening the measurement error per-frame.

Image Noise

Noisy image data and motion blur both have the effect of randomly perturbing the detected positions of each image feature. To simulate these effects, white noise was added into feature positions before being input to different components of the tracking. First, the absolute orientation computation was tested by itself, followed by the full RANSAC / absolute orientation computation component (as described in Section 3.5.2). The purpose of these two tests was first, to see how the orientation computation itself degrades with noise, and second, to see how integration with RANSAC affects this degradation. In each test, the camera was first held stationary for 360 frames and then rotated by 360 1° increments in the same synthetic scene as before (features on a 10m sphere at 5° pan and tilt positions). The average error for each of these stages was measured versus the amount of noise added.

Figure 3.30 shows how the absolute orientation computation performs with increasing noise. The error increases roughly linearly with the magnitude of the noise in the feature tracking. This is the expected behavior, as the orientation

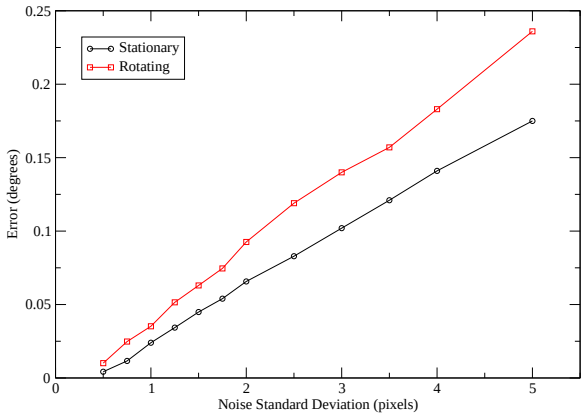

Figure 3.30: Average error in the computed absolute orientation during 1° rotations versus pixel noise added in to feature positions.

computation generates the optimal rotation for the entire set of features, and random noise in the input points will directly translate to random noise in the output rotation. Because these errors are random, they will not systematically affect the orientation estimate (as translation will), but they will cause it to drift over time.

The second test evaluated RANSAC's performance at finding a coherent estimate in spite of increased image noise. Figure 3.31 shows the relationship, which is again roughly linear, though with a much steeper slope in this case. The reason why performance with RANSAC degrades faster is that since feature motion does not match as closely, RANSAC is likely to find a subset of the features that do move coherently (in some random direction) and use them exclusively for the estimate. Therefore, in the case where increased image noise is expected, the

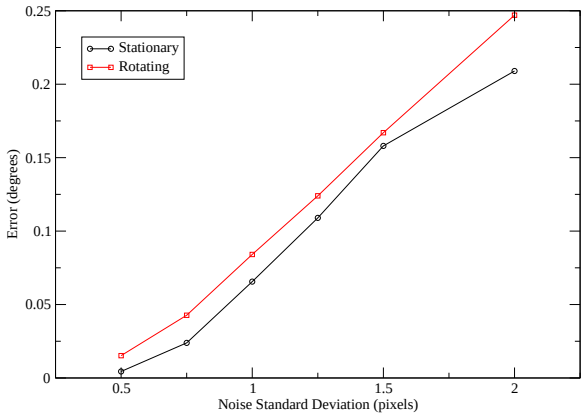

Figure 3.31: Average error in the coherent RANSAC rotation estimate during 1° rotations versus pixel noise added in to feature positions, simulating the effect of noisy or motion blurred images.

RANSAC match threshold should be adjusted to allow for the expected random motion of features. Since in general the feature tracking provides subpixel accuracy, the default match threshold is set to the number of degrees spanned by one pixel, which has performed well in my experiences.

Image noise also affects the reacquisition of landmark features, by increasing the difference between the landmark's stored SURF descriptor and the descriptors of the candidate features during the reacquisition step. To quantify this effect, I computed SURF descriptors for a set of points in an image multiple times with varying amounts of noise added in to each pixel intensity. The difference between the noisy descriptors versus the descriptors under ideal conditions were measured and are presented in Figure 3.32. The similarity computed is the dot product of two unit-length descriptors. As the results show, the similarity drops off linearly

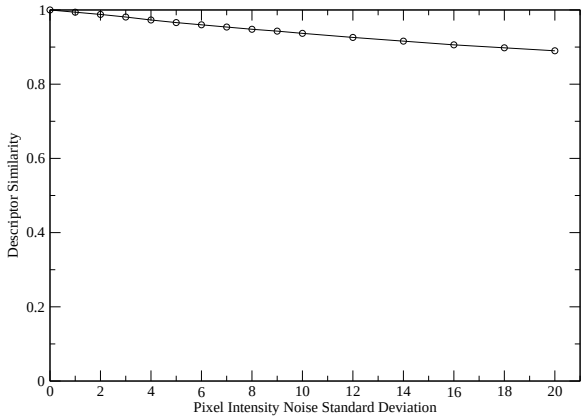

Figure 3.32: Average similarity of between original descriptors and descriptors computed with different amounts of pixel intensity noise added, simulating the effect of image noise on landmark reacquisition.

with increased image noise. Landmark reacquisition looks for a feature with a descriptor that is similar above some threshold (set to 0.95 under normal conditions). Therefore, in the case of increased expected image noise, the landmark threshold should be lowered accordingly, to ensure that landmark descriptors can still be reacquired.

Distractions

Large distractions that move in the image independently of the camera motion (people, cars, etc.) can interfere with the orientation computation if they overwhelm RANSAC's ability to ignore outliers. To test the performance with large distractions, significant coherent noise (standard deviation of 10 pixels, randomly

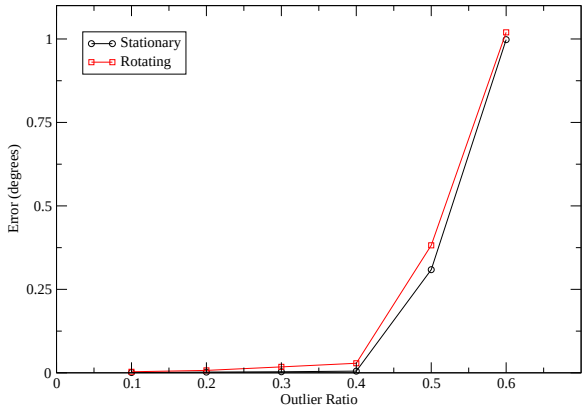

Figure 3.33: Average error in the measured rotation during 1° rotations versus portion of features with significant coherent noise added, simulating the effect of large distractions in the scene.

chosen per-frame) was added to a random portion of the features in each frame before the coherent rotation computation step.

The results of this test can be seen in Figure 3.33. Up until a certain portion, the effect of outliers is nearly negligible, but then the error increases dramatically. This is the expected behavior for the RANSAC algorithm. Outliers can be discarded without affecting the estimate, up until the point that there are so many outliers that they appear to be another set of inliers. Once that threshold is reached, it is likely that the RANSAC estimate will converge on the incorrect set of coherent motion, causing total failure.

This possible failure mode will occur when a single object, such as a near-standing person or a moving car, occludes most of the camera's field of view. More likely in dynamic scenes is that a large number of small distractions will

be present in the scene, each with their own motion. In this case, so long as the portion of inlier features is still above the RANSAC threshold (currently set to 0.33), the correct estimate is still likely to be found, as other potential candidates will be discarded since there will not be a consensus among the features. As the RANSAC algorithm is probabilistic, there is always the possibility of failure – this probability can be set as part of the RANSAC parameter tuning, to trade likelihood of failure with performance. In general, we set the failure probability to 0.01, which the RANSAC algorithm uses to determine the number of iterations it executes before giving up. Given accurate estimates of all the RANSAC parameters (probability that a feature is an inlier and the number of inliers needed to reach a consensus), the failure probability dictates how often RANSAC will fail to find a consensus for good data. In reality, conservative estimates are used for the RANSAC parameters, and so failure on good feature data is extremely uncommon.

Insufficient Texture

The last failure mode is insufficient texture for feature tracking. This can happen outdoors, for example, if the camera is pointed at a clear sky, or indoors looking at an unadorned wall. In these cases, the tracking fails completely as there is not enough information to compute orientation. More likely is that the

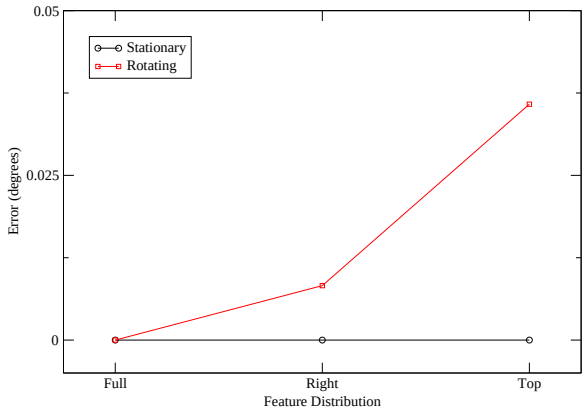

Figure 3.34: Average error in the measured rotation during 1° rotations versus region of the image features are restricted to, simulating the effect of insufficient texture creating lopsided feature distributions.

majority of the image is featureless, but there is some texture on one side that can be tracked – for example, the camera is pointed above the top of a building, so the top half of the image is featureless sky, while the bottom half is textured building. This creates an asymmetric distribution of features in the image that has the potential to systematically influence the orientation computation by including coherent (due to its asymmetry around the image center) random noise in the process.

To test the orientation computation performance under these conditions, we used the same synthetic testing setup but only allowed features in the top or right portions of the image (during a 360° panning motion). The measured rotations in these cases were compared to the known rotation to see if the scenario created a systematic error. Results are presented in Figure 3.34. While there is some

increased error for asymmetric feature configurations, the effect is exceedingly small.

Error Analysis Summary

In practice, the largest overall source of error is from translation of the camera, especially in the case of hand-held camera motion. Adequate illumination and slow camera motion make image noise insignificant, and with a wide field of view camera (50° or greater), lack of texture is rarely a problem. In crowded areas, distractions can be a problem if there is not enough static imagery visible in each frame.

Tracking error impacts the ability of the landmark features to correct for drift in long acquisitions. Landmark reacquisition searches for features within a region around the expected position of the landmark. The search region I use has a radius of 15 pixels, which roughly corresponds to an error of 1°. If the accumulated tracking error is less than this region, then landmarks can be reacquired and will correct the error, enabling long-term tracking without drift. If the expected tracking error is greater, due to poor scene conditions, then the search radius should be increased to ensure that landmarks can still be reacquired.

The impact of tracking error on the acquired environment maps is that it creates visible discontinuities between frames that are spatially adjacent but not

temporally adjacent. In my experiments, registration errors on the order of 1° are large enough to detract from quality. Therefore, the sources of error for a scene need to be mitigated to the point that the total accumulated error is less than 1°. For indoor scenes, that level of accuracy requires the use of a tripod (or extremely careful hand-held rotation). The RANSAC parameters need to be adjusted to accommodate the expected levels of image noise and distractions. With these considerations, we have been able to achieve sufficient accuracy for environment acquisition in both indoor and outdoor scenes (see Figures 3.5.8 and 3.26).

3.6 Mobile AR Summary

The goal of Anywhere Augmentation is embodied in the techniques and applications developed within the ARagorn framework. The aerial annotation application augments the standard wearable data sources with readily available aerial photographs to aid the common task of annotation placement, easing content creation in a wearable context. The remote explorer application also allows the easy creation of content by automatically creating a basic model of a remote environment from the tracked video stream. The GroundCam and panorama construction techniques both extend the capabilities of ARagorn to operate more effectively in new environments, assisting it with accurate tracking and online, automatic envi-

ronment acquisition. These two algorithms will prove useful for a wide variety of both indoor and outdoor AR applications. Finally, using the environment maps from the panorama constructor to illuminate virtual geometry with the surrounding physical scene demonstrates improved visual fidelity in an established AR application domain. Each of these contributions provides the capabilities of advanced augmented reality technologies and exhibits the low setup cost and ease of use associated with Anywhere Augmentation.

Chapter 4

Conclusions

To conclude this thesis, I will revisit the concept of Anywhere Augmentation and view the completed work from that perspective. The goal of Anywhere Augmentation is to achieve the highest level of quality in an AR application with the minimum amount of setup work necessary by the end user. It is the traditionally high startup costs that form a barrier to casual experimentation with AR technologies, hindering widespread acceptance of AR as a usable paradigm in day to day actions. By focusing on faster setup, Anywhere Augmentation aims to bring high quality AR applications to the average computer user.

In the arena of desktop AR, the ARWin system makes using AR interface concepts on desktop computers quick and easy by using printed paper markers as a tangible UI device. Starting to use the ARWin desktop simply requires printing new markers, and the workspace can be moved with the user to a new office by moving the computer and markers, with no additional setup required. Within

ARWin, using the common illumination support is an improvement over previous results – the user must assemble the tracked lightprobe, but once that step is complete, fully dynamic illumination of virtual geometry is possible, and with the same or lower quality scene model as other techniques, complex physical geometry can be lit more accurately by virtual light sources. Finally, the image space error correction technique automatically improves the visual quality of appropriate AR applications, reducing the necessary level of tracking quality, adding more, easier to setup tracking techniques to the Anywhere Augmentation toolbox.

For mobile AR, the ARagorn system does not require any custom hardware or mounting assemblies and can easily be put together from standard AR components. The first application presented in ARagorn, the aerial augmenter, uses Anywhere Augmentation data sources to vastly simplify the creation of 3D annotations in large environments. The GroundCam offers significantly improved position tracking to the ARagorn system with the addition of a second camera, which, when combined with a standard GPS unit, yields high frequency absolute position updates of improved quality over standard GPS alone. Using ARagorn in my second application, the remote explorer, makes quick acquisition of simple models of remote environments possible, and finally, adding the ability to construct environment maps within this system online, by simply panning a camera

around the scene with live, incremental feedback, is a much easier and user-friendly task than previous panorama construction methods.

Overall, each of these techniques and applications successfully further the goals of Anywhere Augmentation. While no single contribution is a silver bullet, together and with other AR techniques, they form a powerful set of tools for the creation of easy to use, high quality AR applications.

4.1 Contributions

To summarize, the specific contributions of this thesis are

- a rapidly-deployable application window manager that makes use of an existing tracking solution for its tangible interface,
- a technique for common illumination between physical and virtual worlds in dynamic environments,
- a post-processing filter to reduce the visual impact of registration errors for impostor polygons.
- an application that uses commonly available aerial photographs as an additional data source for placement of 3D annotations,

- a vision-based tracking modality and hybrid wide-area person-tracker with improved performance,
- a remote exploration system that uses an instrumented scout to acquire information about an environment for offline viewing, and
- an application that constructs environment maps online and uses them to shade virtual geometry.

4.2 Future Work

Anywhere Augmentation is a wide and challenging area within the Augmented Reality field, and there are many avenues along which this work can be continued. Most interesting to me is the idea to extend the panorama construction techniques into an automatic world building system that outputs a hybrid light map and geometry representation of the captured environment. The hybrid model could then be used for the realistic visualization of virtual geometry in the physical scene. This proposed contribution is outlined here as a potential direction for future work.

4.2.1 Hybrid Light Map and Geometry

Based on the results of the panorama construction, the next goal would be to extend the data stored at individual locations to include some positional information as well, resulting in a hybrid lighting and geometry dataset that can be explored and used in more interesting ways than just environment maps. To do this in a feasible fashion, rather than construct complete 3D geometry for arbitrary scenes and arbitrary camera motion, which is exceedingly complex, the technique can take advantage of the scout's behavior which is to move directly between discrete locations. This way the tracked video stream will show the scene geometry along a single direction gradually coming closer – in image space, this motion presents itself as gradual increase in scale and the motion of features away from the center of the image, proportionally to their distance from the camera. It is this proportional motion parallax that can be exploited for geometry information.

There is a significant amount of research dealing with the acquisition of image-based scene models. Plenoptic modeling [65] is a technique to make models from panoramas at different locations. The Lumigraph [42] and lightfield rendering [58] extend the idea to create a full representation of 4D light fields for rendering novel views. These techniques produce very nice results, but the size of datasets and complexity of algorithms make them inappropriate for online modeling in a wearable system. Shum et al.'s [99] and Chai et al.'s [17] work on concentric mosaics

and their applications is an image based approach that is more appealing in terms of computational and storage complexity, but it requires unusual camera sweeps that limits its flexibility to handle the sort of behavior an exploring scout will exhibit. Other research has been done on world modeling that involves significant amounts of user input, including Shum et al.'s panorama markup [96], and Román et al.'s multiperspective images [88], but for Anywhere Augmentation, fully automatic construction is desirable. The Photo Tourism project [101] generates 3D point clouds fully automatically, but runs offline. Davison's single camera SLAM [25] also generates point clouds, in realtime. Unfortunately, point clouds are too sparse a representation for the world model – a dense representation is needed for effects such as lighting and environment mapping. Other groups have built dense world models automatically, such as Uyttendaele et al. [110] and Teller et al. [107], but their approaches require significant expensive hardware rigs that acquire very large amounts of data, which is then processed offline for high quality results. Finally, Tour Into the Video [53] is a very different approach, creating a stylized world model that combines image-based and model-based approaches that would work well for my proposed technique. However, it does require user input and offline processing, and is somewhat limited in terms of how large a scene it can represent.

I envision a novel means of acquiring and modeling an outdoor environment in realtime, fully automatically. The first implementation can construct cylinders (with hemispherical end caps) of image data along a path - the rough equivalent of a spherical environment map stretched to encompass a linear path instead of a point location. The texture map representing the environment around the user's path can be a set of four rectangles running the length of the path (a stretched cube map), with squares on either end. This texture will be mapped onto a cylinder of triangles, extended as the user continues to walk forward. Specifically, the cylinder will be made of concentric rings of triangles, and the width of the last ring will increase until a threshold is reached, at which point it will be replaced by two rings. After splitting, only the second ring will be stretched, until the same threshold is reached and it is split itself. This way the majority of the geometry is static – only the last ring will be updated at each step. As additional rings of geometry are added, the texture map can be extended as well. See Figure 4.1 for an example of the geometry.

Applying the video texture to the geometry will be similar to the approach for panorama construction. The same method of feature tracking will be used. An initial implementation can focus on background image data, so the good features can be determined based on the observation that their apparent depth is further away than some maximum threshold, at which point they are assumed

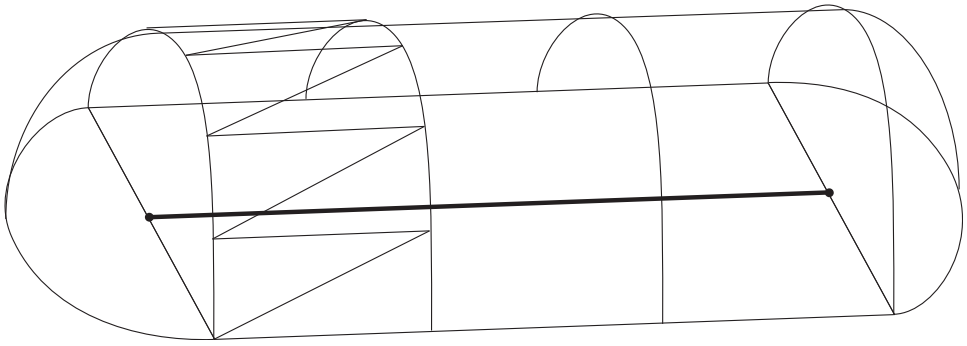

Figure 4.1: Example constructed geometry for a linear segment of a user's path. The dark line is the user's path. The geometry surrounding it is completely triangulated, but triangles are omitted for clarity.

to be "at infinity". Computing depth of features, or more specifically 3D position estimates, is a matter of triangulation of the feature positions over multiple frames of camera translation. Other tracking techniques such as the GroundCam will provide translation information, so a 3D position estimate can be computed by finding the point of best fit of a series of rays cast from consecutive camera positions. This best fit point will be a Gaussian estimate of the actual location, with standard deviation computed from the error in triangulation. Further cast rays can be used to refine this position and error estimate until the point is either found within some error tolerance or determined to be bad (too large error) and discarded. Points that are far enough away as to be outside a distance threshold separating foreground and background objects, will be considered to be at infinity. With this information, good patches of background image texture can be found which can then be projected onto the surrounding geometry and written into the

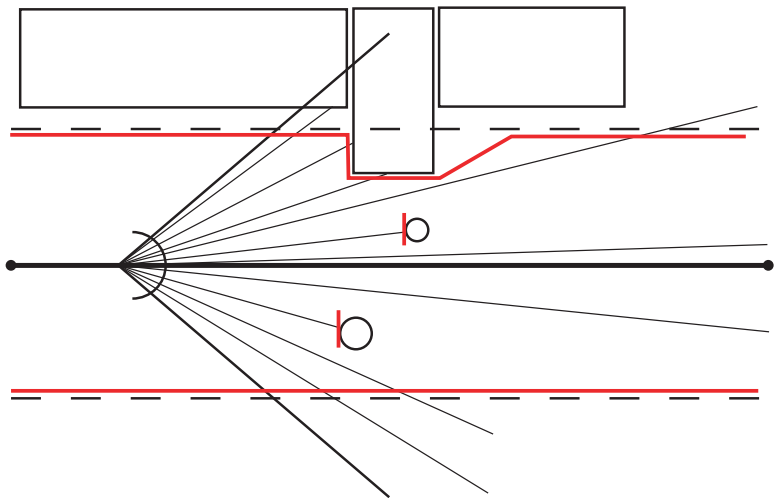

Figure 4.2: An example of geometry creation based on computed depth of features. From the user's position along the path (the dark line), tracked features correspond to cast rays in the user's field of view. These rays end at the computed depth of the objects they track. The dotted lines show the distance threshold for infinity. The red lines show the modified wall geometry and the pop-up polygons generated for foreground objects.

world texture map. Gaps can be filled using the same algorithm as the panorama construction.

Rendering with this simple world model will not convincingly recreate the sensation of moving along the path between locations, as parallax and foreground objects will be absent. This approach could be extended into a more accurate approximation, by denting the tube geometry to represent real measured depths, and adding pop-up geometry for foreground objects (see Figure 4.2). First, modifying the shape of the basic tube geometry is a matter of taking the computed background depths and using them to displace vertices near the feature location.

Features and tube geometry will not match up exactly, so the sparse depths from features will be used to smoothly interpolate the distances of the tube vertices. Second, handling foreground objects is a more complex issue. Features are determined to be part of the foreground when they exhibit a large depth discontinuity with adjacent features. Once a foreground feature has been identified, the foreground image data must be segmented from the background. One approach is to find the min-cost cut of a Delaunay triangulation of the good features (assuming a sufficient number of tracked features), where cost is based on the depth discontinuity of edges. The boundary of the cut is then found by the cut edges' duals in the equivalent Voronoi diagram. This boundary can be refined by using color segmentation within the boundary region based on the colors surrounding the foreground feature, to get a per-pixel mask. Alternately, color segmentation could be used instead of the boundary selection by using the color model to drive a flood fill (similar to how Adobe Photoshop's Magic Wand tool works). Either way, once the foreground object is selected, a polygon can be added to the world model at the correct location, facing the user's current position, with the selected foreground texture applied. This polygon does not necessarily need to be attached to the surrounding environment model. Rendering of these foreground pop-up objects could be as static geometry, or as billboards oriented to face the user. The polygons could also show the same texture on either side, or could be blank or

filled in with the inpainting algorithm on the back side – user experience can guide what is most appropriate here.

4.2.2 Realistic Outdoor Visualization

The result of the world building approach will be a hybrid geometry and light map structure ideal for applications that need this information to render realistic virtual geometry using the physical lighting environment. These applications are often used to visualize archaeological ruins [111] or proposed architectural additions [108], creating medium to large scale representations of landscaping and buildings. The goal then is an application that uses the environment map / world model generated by the scout and the common illumination techniques developed in the desktop AR domain to create better integrated virtual geometry by rendering it with realistic shading from the physical environment.

To accomplish this, filtered material response maps can be created in a similar manner as used in my previous desktop AR common illumination work, except that the environment map will be constructed by projecting the world model to the position of the virtual geometry. This map will then be filtered as before and applied to the geometry with standard OpenGL environment mapping techniques.

4.3 Closing Remarks

The result of this work is an advance in the state of Anywhere Augmentation and the capabilities of AR systems in new environments. Future research in this area must continue to strive to achieve the same high quality of traditional AR applications while simultaneously reducing or eliminating the upfront costs normally required. Full online world model acquisition is the next big challenge in the field, which will then open up even more exciting application possibilities for casual AR users.

Bibliography

[1] 360 panorama professional, 2007. http://www.360dof.com/.

[2] 3DVista stitcher, 2007. http://www.3dvista.com/.

[3] ADG panorama tools, 2007. http://www.albatrossdesign.com/.

[4] K. Agusanto, L. Li, Z. Chuangui, and N. Sing. Photorealistic rendering for augmented reality using environment illumination. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2003.

[5] ArcSoft Panorama Maker, 2007. http://www.arcsoft.com/.

[6] M. Ashikhmin and A. Ghosh. Simple blurry reflections with environment maps. *Journal of Graphics Tools*, 7(4), 2002.

[7] Autopano, 2007. http://www.autopano.net/.

[8] R. Azuma. *Predictive Tracking for Augmented Reality*. PhD thesis, University of North Carolina at Chapel Hill, 1995.

[9] Y. Baillot, D. Brown, and S. Julier. Authoring of physical models using mobile computers. In *Proceedings of the International Symposium on Wearable Computers*, 2001.

[10] P. Baudisch, D. Tan, D. Steedly, E. Rudolph, M. Uyttendaele, C. Pal, and R. Szeliski. Panoramic viewfinder: providing a real-time preview to help users avoid flaws in panoramic pictures. In *Proceedings of the Conference of the Computer-Human Interaction Special Interest group of Australia*, 2005.

[11] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision*, 2006.

[12] R. Behringer. Registration for outdoor augmented reality applications using computer vision techniques and hybrid sensors. In *Proceedings of Virtual Reality*, 1999.

[13] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher. Simultaneous structure and texture image inpainting. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2003.

[14] M. Brown and D. Lowe. Recognising panoramas. In *Proceedings of the International Conference on Computer Vision*, 2003.

[15] J. Canny. A computational approach to edge detection. *Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 1986.

[16] D. Capel and A. Zisserman. Automated mosaicing with super-resolution zoom. In *Proceedings of the Conference on Computer Vision and Patter Recognition*, 1998.

[17] J. Chai, S. Kang, and H. Shum. Rendering with non-uniform approximate concentric mosaics. In *Proceedings of the Workshop on 3D Structure from Multiple Images of Large-Scale Environments*, 2001.

[18] S. Chen. QuickTime VR: an image-based approach to virtual environment navigation. In *Proceedings of SIGGRAPH*, 1995.

[19] E. Coelho, B. MacIntyre, and S. Julier. OSGAR: A scene graph with uncertain transformations. In *International Symposium on Mixed and Augmented Reality*, 2004.

[20] J. Coleshill and A. Ferworn. Spherical panoramic video – the space ball. In *Proceedings of the International Conference on Computational Science and its Applications*, 2003.

[21] A. Comport, E. Marchand, and F. Chaumette. A real-time tracker for markerless augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2003.

[22] S. Coorg and S. Teller. Spherical mosaics with quaternions and dense correlation. *International Journal of Computer Vision*, 37(3), 2000.

[23] A. Criminisi, P. Perez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *Transaction on Image Processing*, 13(9), 2004.

[24] D Joiner, 2007. http://www.d-vw.com/.

[25] A. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the International Conference on Computer Vision*, 2003.

[26] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of SIGGRAPH*, 1998.

[27] S. DiVerdi and T. Höllerer. Combining dynamic physical and virtual illumination in augmented reality. Technical report, University of California, Santa Barabara, 2004. UCSB//CSD-04-28.

[28] S. DiVerdi and T. Höllerer. Image-space correction of AR registration errors using graphics hardware. In *Proceedings of Virtual Reality*, 2006.

[29] S. DiVerdi and T. Höllerer. GroundCam: A tracking modality for mobile mixed reality. In *Proceedings of Virtual Reality*, 2007.

[30] S. DiVerdi, T. Höllerer, and R. Schreyer. Level of detail interfaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2004.

[31] S. DiVerdi, D. Nurmi, and T. Höllerer. ARWin - a desktop augmented reality window manager. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2003.

[32] S. DiVerdi, D. Nurmi, and T. Höllerer. A framework for generic inter-application interaction for 3D AR environments. In *Proceedings of the International Augmented Reality Toolkit Workshop*, 2003.

[33] L. Fang, P. Antsaklis, L. Montestruque, M. McMickell, M. Lemmon, Y. Sun, H. Fang, I. Koutroulis, M. Haenggi, M. Xie, and X. Xie. Design of a wireless assisted pedestrian dead reckoning system - the NavMote experience. *Transactions on Instrumentation and Measurement*, 54(6), 2005.

[34] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.

[35] M. Fleck. Perspective projection: the wrong imaging model. Technical report, University of Iowa, 1995. TR 95-01 Computer Science.

[36] A. Fournier, A. Gunawan, and C. Romanzin. Common illumination between real and computer generated scenes. In *Proceedings of Graphics and Interface*, 1993.

[37] E. Foxlin. Inertial head-tracker sensor fusion by a complementary separate-bias Kalman filter. In *Proceedings of the Virtual Reality Annual International Symposium*, 1996.

[38] E. Foxlin and L. Naimark. VIS-tracker: a wearable vision-inertial self-tracker. In *Proceedings of Virtual Reality*, 2003.

[39] J. Fung and S. Mann. Using multiple graphics cards as a general purpose parallel computer : Applications to computer vision. volume 1, 2004.

[40] S. Gibson and A. Murta. Interactive rendering with real-world illumination. In *Proceedings of Eurographics Workshop on Rendering*, 2000.

[41] Google maps, 2006. http://maps.google.com/.

[42] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The Lumigraph. In *Proceedings of SIGGRAPH*, 1996.

[43] X. Gu. GPU-based conformal flow on surfaces. Technical report, State University of New York at Stony Brook, 2006. Computer Science.

[44] M. Hansen, P. Anandan, K. Dana, G. van der Wal, and P. Burt. Real-time scene stabilization and mosaic construction. In *Proceedings of the Workshop on Applications of Computer Vision*, 1994.

[45] T. Höllerer and C. Coffin. INVITE: 3D-augmented interactive video teleconferencing. In *Proceedings of the Pervasive 2006 International Workshop on the Tangible Space Initiative*, 2006.

[46] T. Höllerer and S. Feiner. Mobile augmented reality. In H. Karimi and A. Hammad, editors, *Telegeoinformatics: Location-Based Computing and Services*. Taylor and Francis Books Ltd., 2004.

[47] T. Höllerer, J. Wither, and S. DiVerdi. *Location Based Services and TeleCartography*, chapter Anywhere Augmentation: Towards Mobile Augmented Reality in Unprepared Environments. Lecture Notes in Geoinformation and Cartography. Springer, 2007.

[48] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4, 1987.

[49] S. Hsu, H. Sawhney, and R. Kumar. Automated mosaics via topology inference. *Computer Graphics and Applications*, 22(2), 2002.

[50] M. Imura, Y. Yasumuro, Y. Manabe, and K. Chihara. Fountain designer: Control virtual water as you like. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2006. Demo.

[51] Kaidan 360 OneVR, 2007. http://www.kaidan.com/.

[52] M. Kalkusch, T. Lidy, M. Knapp, G. Reitmayr, H. Kaufmann, and D. Schmalstieg. Structured visual markers for indoor pathfinding. In *Proceedings of the International Augmented Reality Toolkit Workshop*, 2002.

[53] H. Kang and S. Shin. Tour into the video: image-based navigation scheme for video sequences of dynamic scenes. In *Proceedings of the Symposium on Virtual Reality Software and Technology*, 2002.

[54] S. Kim, E. Chang, C. Ahn, and W. Woo. Image-based panoramic 3D virtual environment using rotating two multi-view cameras. In *Proceedings of the International Conference on Image Processing*, 2003.

[55] G. Klein and T. Drummond. Sensor fusion and occlusion refinement for tablet-based AR. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2004.

[56] D. Koller, G. Klinker, E. Rose, D. Breen, R. Whitaker, and M. Tuceryan. Real-time Vision-Based camera tracking for augmented reality applications. In *Proceedings of the Symposium on Virtual Reality Software and Technology*, 1997.

[57] A. Kropp, N. Master, and S. Teller. Acquiring and rendering high-resolution spherical mosaics. In *Proceedings of the Workshop on Omnidirectional Vision*, 2000.

[58] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings of SIGGRAPH*, 1996.

[59] LM Stitch, 2007. http://www.lostmarble.com/.

[60] C. Loscos, G. Drettakis, and L. Robert. Interactive virtual relighting of real scenes. *Transactions on Visualization and Computer Graphics*, 6(4), 2000.

[61] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 20, 2003.

[62] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.

[63] B. MacIntyre and E. Coelho. Adapting to dynamic registration errors using level of error (LOE) filtering. In *International Symposium on Augmented Reality*, 2000.

[64] S. Mann and R. Picard. Video orbits of the projective group: a new perspective on image mosaicing. Technical Report 338, MIT Technical Report, 1995.

[65] L. McMillan and G. Bishop. Plenoptic modeling: an image-based rendering system. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995.

[66] J. Montiel and A. Davison. A visual compass based on SLAM. In *Proceedings of the International Conference on Robotics and Automation*, 2006.

[67] Myst, 1995. Cyan Worlds, http://www.cyanworlds.com/.

[68] S. Nayar. Omnidirectional video camera. In *Proceedings of the Image Understanding Workshop*, 1997.

[69] M. Oliveira, B. Bowen, R. McKenna, and Y. Chang. Fast digital image inpainting. In *Proceedings of the Conference on Visualization, Imaging, and Image Processing*, 2001.

[70] Open source computer vision library reference manual, December 2000. Intel Corporation.

[71] PanaVue ImageAssembler, 2007. http://www.panavue.com/.

[72] Panorama factory, 2007. http://www.panoramafactory.com/.

[73] PanoStitcher, 2007. http://www.pixtra.com/.

[74] PanoWeaver, 2007. http://www.easypano.com/.

[75] S. Peleg and J. Herman. Panoramic mosaics by manifold projection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1997.

[76] S. Peleg, B. Rousso, A. Rav-Acha, and A. Zomet. Mosaicing on adaptive manifolds. *Transactions on Pattern Analysis and Machine Intelligence*, 22(10), 2000.

[77] Photoshop elements, 2007. http://www.adobe.com/.

[78] PhotoVista Panorama, 2007. http://www.iseephotovista.com/.

[79] W. Piekarski and B. Thomas. Augmented reality working planes: A foundation for action and construction at a distance. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2004.

[80] I. Poupyrev, D. Tan, M. Billinghurst, H. Kato, H. Regenbrecht, and N. Tetsutani. Developing a generic augmented-reality interface. *Computer*, 35(3), 2002.

[81] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of the International Conference on Mobile Computing and Networking*, 2000.

[82] Project Looking Glass, 2006. Sun Microsystems, http://www.sun.com/software/looking_glass/.

[83] Quicktime vr, 2007. Apple, http://www.apple.com/quicktime/technologies/qtvr/.

[84] RealViz stitcher, 2007. http://stitcher.realviz.com/.

[85] K. Rehman. Visualisation, interpretation and use of location-aware interfaces. Technical report, University of Cambridge, Computer Laboratory, 2005. UCAM-CL-TR-634.

[86] G. Reitmayr and T. Drummond. Going out: Robust model-based tracking for outdoor augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2006.

[87] G. Robertson, M. Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, D. Thiel, and V. Gorokhovsky. The Task Gallery: A 3D window manager. In *Proceedings of the Conference on Human Factors in Software*, 2000.

[88] A. Roman, G. Garg, and M. Levoy. Interactive design of multi-perspective images for visualizing urban landscapes. In *Proceedings of the Conference on Visualization*, 2004.

[89] B. Rousso, S. Peleg, I. Finci, and A. Rav-Acha. Universal mosaicing using pipe projection. In *Proceedings of the International Conference on Computer Vision*, 1998.

[90] K. Satoh, M. Anabuki, H. Yamamoto, and H. Tamura. A hybrid registration method for outdoor augmented reality. In *Proceedings of the International Symposium on Augmented Reality*, 2001.

[91] H. Sawhney, S. Hsu, and R. Kumar. Robust video mosaicing through topology inference and local to global alignment. *Lecture Notes in Computer Science*, 1407, 1998.

[92] H. Sawhney, R. Kumar, G. Gendel, J. Bergen, D. Dixon, and V. Paragano. VideoBrushTM: Experiences with consumer video mosaicing. In *Proceedings of the Workshop on Applications of Computer Vision*, 1998.

[93] D. Schmalstieg, A. Fuhrmann, and G. Hesina. Bridging multiple user interface dimensions with augmented reality. In *International Symposium on Augmented Reality*, 2000.

[94] C. Shen, H. Shum, and J. O'Brien. Image based rendering and illumination using spherical mosaics. In *Proceedings of SIGGRAPH Technical Sketches*, 2001.

[95] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1994.

[96] H. Shum, M. Han, and R. Szeliski. Interactive construction of 3D models from panoramic mosaics. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 1998.

[97] H. Shum and L. He. Rendering with concentric mosaics. In *Proceedings of SIGGRAPH*, 1999.

[98] H. Shum, K. Ng, and S. Chan. Virtual reality using the concentric mosaic: construction,rendering and data compression. In *Proceedings of Image Processing*, 2000.

[99] H. Shum, K. Ng, and S. Chan. A virtual reality system using the concentric mosaic: construction, rendering, and data compression. *Transactions on Multimedia*, 7(1), 2005.

[100] H. Shum and R. Szeliski. Panoramic image mosaics. Technical Report MSR-TR-97-23, Microsoft Research, 1997.

[101] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *Transactions on Graphics*, 25(3), 2006.

[102] D. Steedly, C. Pal, and R. Szeliski. Efficiently registering video into panoramic mosaics. In *Proceedings of the International Conference on Computer Vision*, 2005.

[103] Stitch FishEye Pro, 2007. http://www.stitch-fisheye.com/.

[104] N. Sugano, H. Kato, and K. Tachibana. The effects of shadow representation of virtual objects in augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2003.

[105] R. Szeliski. Video mosaics for virtual environments. *Computer Graphics and Applications*, 16(2), 1996.

[106] R. Szeliski and H. Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of SIGGRAPH*, 1997.

[107] S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, and N. Master. Calibrated, registered images of an extended urban area. *International Journal of Computer Vision*, 53(1), 2003.

[108] B. Thomas, W. Piekarski, and B. Gunther. Using augmented reality to visualise architecture designs in an outdoor environment, 1999.

[109] Ulead Cool 360, 2007. http://www.ulead.com/.

[110] M. Uyttendaele, A. Criminisi, S. Kang, S. Winder, R. Szeliski, and R. Hartley. Image-based interactive exploration of real-world environments. *Computer Graphics and Applications*, 24(3), 2004.

[111] V. Vlahakis, N. Ioannidis, J. Karigiannis, M. Tsotros, M. Gounaris, D. Stricker, T. Gleue, P. Daehne, and L. Almeida. Archeoguide: An augmented reality guide for archaeological sites. *Computer Graphics and Applications*, 22(5), 2002.

[112] VRstitcher fisheye, 2007. http://www.360dof.com/.

[113] G. Welch and G. Bishop. An introduction to the Kalman filter. *Proceedings of SIGGRAPH Course Notes*, 2001. Course 8.

[114] J. Wither, S. DiVerdi, and T. Höllerer. Using aerial photographs for improved mobile AR annotation. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2006.

[115] J. Wither and T. Höllerer. Pictorial depth cues for outdoor augmented reality. In *Proceedings of the International Symposium on Wearable Computers*, 2005.

[116] Yahoo maps, 2006. http://maps.yahoo.com/.

[117] S. You and U. Neumann. Fusion of vision and gyro tracking for robust augmented reality registration. In *Proceedings of Virtual Reality*, 2001.

[118] Z. Zhang. A flexible new technique for camera calibration. *Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 2000.

[119] Z. Zhu, G. Xu, and X. Lin. Constructing 3D natural scene from video sequences with vibrated motions. In *Proceedings of the Virtual Reality Annual International Symposium*, 1998.

[120] Z. Zhu, G. Xu, E. Riseman, and A. Hanson. Fast generation of dynamic and multi-resolution 360-degree panorama from video sequences. In *Proceedings of the International Conference on Multimedia Computing and Systems*, 1999.