

MODEL-BASED CAMERA TRACKING FOR AUGMENTED REALITY

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Aytek Aman

August, 2014

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Uğur Gdkbay(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Blent zgç

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Veysi İşler

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

MODEL-BASED CAMERA TRACKING FOR AUGMENTED REALITY

Aytek Aman
M.S. in Computer Engineering
Supervisor: Prof. Dr. Uğur Güdükbay
August, 2014

Augmented reality (AR) is the enhancement of real scenes with virtual entities. It is used to enhance user experience and interaction in various ways. Educational applications, architectural visualizations, military training scenarios and pure entertainment-based applications are often enhanced by augmented reality to provide more immersive and interactive experience for the users. With hand-held devices getting more powerful and cheap, such applications are becoming very popular.

To provide natural AR experiences, extrinsic camera parameters (position and rotation) must be calculated in an accurate, robust and efficient way so that virtual entities can be overlaid onto the real environments correctly. Estimating extrinsic camera parameters in real-time is a challenging task. In most camera tracking frameworks, visual tracking serve as the main method for estimating the camera pose. In visual tracking systems, keypoint and edge features are often used for pose estimation. For rich-textured environments, keypoint-based methods work quite well and heavily used. Edge-based tracking, on the other hand, is more preferable when the environment is rich in geometry but has little or no visible texture.

Pose estimation for edge based tracking systems generally depends on the control points that are assigned on the model edges. For accurate tracking, visibility of these control points must be determined in a correct manner. Control point visibility determination is computationally expensive process. We propose a method to reduce computational cost of the edge-based tracking by preprocessing the visibility information of the control points. For that purpose, we use persistent control points which are generated in the world space during preprocessing step. Additionally, we use more accurate adaptive projection algorithm

for persistent control points to provide more uniform control point distribution in the screen space.

We test our camera tracker in different environments to show the effectiveness and performance of the proposed algorithm. The preprocessed visibility information enables constant time calculations of control point visibility while preserving the accuracy of the tracker. We demonstrate a sample AR application with user interaction to present our AR framework, which is developed for a commercially available and widely used game engine.

Keywords: Camera localization, augmented reality, vision-based camera tracking, edge-based camera tracking.

ÖZET

ARTIRILMIŞ GERÇEKLIK İÇİN MODEL TABANLI KAMERA TAKIBİ

Aytek Aman

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Yöneticisi: Prof. Dr. Uğur GÜDÜKBAY

Ağustos, 2014

Artırılmış gerçeklik (AG), gerçek ortamların sanal nesnelere zenginleştirilmesidir. AG kullanıcı deneyimi ve etkileşimini geliştirmek için çeşitli şekillerde kullanılmaktadır. Eğitim uygulamaları, mimari görselleştirmeler, askeri eğitim senaryoları ve eğlence amaçlı uygulamalar çoğu zaman daha gerçekçi bir tecrübe yaşatmak amacıyla artırılmış gerçeklik yöntemleriyle desteklenir. Taşınabilir cihazların gelişmesi ve ucuzlamasıyla birlikte bu tarz uygulamalar oldukça yaygın hale gelmiştir.

Daha doğal AG tecrübesi yaşatmak adına dışsal kamera parametrelerinin hatasız, güvenilir ve hızlı bir biçimde hesaplanması gerekir. Böylece, sanal nesnelere gerçek ortam görüntülerinin üzerine isabetli bir şekilde yerleştirilebilir. Dışsal kamera parametrelerinin gerçek zamanlı tespiti zordur. Birçok kamera takip çatısında, kamera parametrelerinin hesaplanması için temel yöntem olarak görsel takip algoritmaları kullanılmaktadır. Görsel takip sistemlerinde anahtar nokta ve kenar özellikleri poz hesabı için sıkça kullanılır. Zengin kaplamalı ortamlarda anahtar nokta tabanlı metotlar iyi sonuçlar vermektedir ve yaygınca kullanılmaktadır. Buna karşılık, kenar tabanlı takip yöntemleri geometri bakımından zengin fakat kaplama açısından zengin olmayan ortamlarda daha çok tercih edilir.

Kenar tabanlı takip yöntemiyle yapılan poz hesabı genel olarak model kenarlarında belirlenen kontrol noktalarını kullanır. Hatasız takip için bu noktaların görünürlüğü doğru bir şekilde hesaplanmalıdır. Kontrol noktalarının görünürlüğünün belirlenme süreci hesaplama açısından masraflıdır. Kontrol noktalarının görünürlüğünü önceden hesaplayarak kenar tabanlı takip yönteminin hesaplama masrafını düşürmek için bir yöntem öneriyoruz. Bu amaçla, ön

işleme aşamasında dünya koordinat düzleminde belirlenmiş kalıcı kontrol noktaları kullanılmaktadır. Ek olarak, kalıcı kontrol noktalarının ekranda daha düzenli dağılımını sağlayabilmek adına daha isabetli olan uyarlanabilir bir izdüşüm yöntemi kullanıyoruz.

Önerdiğimiz algoritmanın performans ve kalitesini ölçmek için kamera takipçimizi çeşitli ortamlarda test ettik. Önceden işlenmiş görünürlük değerleri, takip kalitesinden ödün vermeden sabit zamanlı görünürlük hesabını mümkün kılmıştır. Geliştirdiğimiz AG çatısı kullanıcı etkileşimi bulunan bir uygulamayla sunulmuştur. Bahsi geçen uygulama çatısı, yaygın olarak kullanılan ticari bir oyun motoru için geliştirilmiştir.

Anahtar sözcükler: Kamera konumlandırma, artırılmış gerçeklik, bilgisayarla görüye dayalı kamera takibi, kenar tabanlı kamera takibi.

Acknowledgement

I would like to thank Prof. Dr. Uğur Gdkbay for his invaluable guidance throughout my research. I am grateful to Prof. Dr. Bulent zgç and Prof. Dr. Veysi İřler for kindly accepting to be in my thesis jury and for their valuable comments and suggestions.

I would also like to thank my family and Dilara Albayrak for their continuous support and encouragement.

Finally, I would like to thank The Scientific and Technological Research Council of Turkey (TBTAK) for providing me financial support for my M.S. studies. The research described in this thesis is supported by The Scientific and Technological Research Council of Turkey (TBTAK) under grant no. EEEAG 112E110.

Contents

1	Introduction	1
1.1	Motivation and Scope	1
1.2	Contributions	2
1.3	Thesis Organization	3
2	Background and Related Work	4
2.1	Augmented Reality	4
2.1.1	Interaction	4
2.1.2	Rendering and Occlusions	5
2.1.3	Scene Preparation	6
2.1.4	Sound and Haptic Feedback	6
2.2	Camera Localization	7
2.2.1	Vision-based Methods	7
2.2.2	Inertial Measurement Units	12
2.2.3	Hybrid Approaches	12

<i>CONTENTS</i>	ix
2.2.4 Markers	13
3 Tracking System	15
3.1 Preprocessing	15
3.2 Initialization	16
3.3 Real-time Tracking	16
3.3.1 Model Projection and Control Point Generation	16
3.3.2 Locating Edges in the Video Feed	17
3.3.3 Formulating the Least Squares Problem	20
4 Improved Tracking	24
4.1 Persistent Control Point Generation and Projection	24
4.2 Exploiting View-dependent Information for Camera Tracking	28
4.2.1 Visibility Information	28
4.3 Additional Details of the Tracking System	30
4.3.1 View Frustum and Occlusion Culling	30
4.3.2 Silhouette - Non silhouette Edges	31
4.3.3 Motion Prediction	31
4.3.4 System Overview	32
5 Evaluation	34
5.1 Cached Visibility Tests	34

<i>CONTENTS</i>	x
5.1.1 Preprocessing	34
5.1.2 Tracking	35
5.2 Sample AR Application	36
6 Experimental Results	37
6.1 Test Environment	37
6.2 Cached Visibility Tests	38
6.2.1 Preprocessing	38
6.2.2 Tracking	40
7 Conclusion	43
Bibliography	45
Appendices	50
A Software Packages Used in the Implementation	50
A.1 Unity Game Engine	50
A.2 ALGLIB	51

List of Figures

2.1	Types of markers: active marker (left), passive marker (right) . . .	14
3.1	Sample 3D model: all edges (left), salient edges (right)	16
3.2	Locating edges in the video feed: model edges (blue), sample points (red), sample point - located edge pairs (green)	19
4.1	The control points generated for two marker objects at the preprocessing stage.	25
4.2	Control points generated using the persistent control point generation scheme and their storage in the array. The numbers below the points are the depth of the points in the tree structure and the numbers in the boxes indicate the order of generation.	25
4.3	Control points projected using using fixed marker edge length (top) and control points projected using adaptive approach (bottom) . .	26
4.4	A subset of the control points are projected onto the video feed and used for tracking.	27
4.5	Various types of edges for a cylinder object. Blue: non-salient, non-silhouette edges; green: non-salient, silhouette edges; red: salient edges.	32

4.6	The flow diagram of the improved camera tracking system.	33
5.1	Sample augmented reality application with physics interaction and occlusion handling.	36
6.1	The scenes used to test the camera tracker.	38
6.2	3D models of the scenes that are used as natural markers for camera tracking.	38
6.3	Estimation errors (in meters) of the first scene calculated with respect to the baseline for varying 3D grid cell sizes.	40
6.4	Estimation errors (in meters) of the second scene calculated with respect to the baseline for varying 3D grid cell sizes.	41

List of Tables

6.1	Visibility caching preprocessing times (in seconds) for the first scene with respect to different 3D grid cell sizes and minimum world sampling interval values	39
6.2	Memory requirements of visibility caching (in megabytes) for the first scene with respect to different 3D grid cell sizes and minimum world sampling interval values	39
6.3	Visibility caching preprocessing times (in seconds) for the second scene with respect to different 3D grid cell sizes and minimum world sampling interval values	39
6.4	Memory requirements of visibility caching (in megabytes) for the second scene with respect to different 3D grid cell sizes and minimum world sampling interval values	39
6.5	MSE of trackers using visibility caching for the first scene with different cell sizes	40
6.6	MSE of trackers using visibility caching for the first scene with different cell sizes	41
6.7	Average processing times (in seconds) per frame using different visibility testing methods.	42

6.8	Average processing times (in seconds) per frame using different visibility testing methods.	42
-----	--	----

Chapter 1

Introduction

1.1 Motivation and Scope

Augmented Reality (AR) is the enhancement of real scenes by overlaying virtual entities. It is mostly used for entertainment, education and architectural visualization. High computational power of the modern hardware makes it possible to run AR applications even in mobile devices such as cellphones and tablet computers. With AR, it is possible to create and animate virtual characters in real world environments, prepare educational and instructional applications where visual instructions directly appear on the devices, create complete visualizations of historical buildings.

In order to provide immersive AR experiences, virtual entities should be correctly registered to the real world objects. To this end, intrinsic and extrinsic camera parameters should be determined in a robust and accurate way. Camera pose estimation process is often called as camera tracking. For camera tracking, vision-based methods and Inertial Measurement Units (IMUs) are used extensively. Edge-based and keypoint-based tracking are popular vision-based methods for camera tracking. Additionally, new tracking systems use depth based tracking methods but such systems are less popular since most consumer type mobile devices lack depth cameras.

In the presence of rich textured entities in the environment, keypoint-based tracking methods can be employed for camera tracking. If the objects are geometrically complex but contains little or no texture, edge-based tracking methods can be used for pose estimation. Edge-based tracking systems rely on the control points that are defined on the model edges. For correct pose estimation, visibility of these control points should be determined correctly. Visibility determination process for control points requires extra computation time. With limited resources of popular AR devices (tablets, mobile phones), this becomes a problem. In this thesis, we propose a technique to perform constant time visibility determination for the control points in the simulation environment to reduce computational cost of the tracking algorithm.

1.2 Contributions

The main contributions of this thesis are given as follows.

We use more accurate adaptive projection scheme for persistent control points that are generated during preprocessing. This allows more evenly distributed control points in the screen space along the projected marker edges, thus, giving equal importance to the measurements across projected edges. Such projection scheme is especially useful in cases where trackable environment surrounds the camera and contain long edges.

We extend the previously proposed persistent control point scheme to pre-compute visibility values of the persistent control points. We store visibility values of the control points in spatial indexing structure which is a uniform 3D grid. In this way, visibility tests can be performed in constant time per control point.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 summarizes the works related to augmented reality and camera tracking. Chapter 3 describes the edge-based tracking process in detail. Chapter 4 elaborates the proposed methods to improve the edge-based trackers. Chapter 5 explains the evaluation methods that are used to assess the performance of the proposed methods, along with the system parameters. Chapter 6 presents the test results for the proposed approaches using different configurations of the tracking system. Finally, Chapter 7 concludes with the discussion of the results and provides some possible directions for future research.

Chapter 2

Background and Related Work

2.1 Augmented Reality

Augmented Reality (AR) is very active research field. High computational power of modern hardware makes it possible to provide AR content on standard desktop computers and recent mobile devices like tablets and cell phones. In the following sections, fundamental research areas related to the augmented reality is summarized.

2.1.1 Interaction

Interaction in AR environments is a challenging research area. Interaction techniques in many AR applications depend on the capabilities of touch screens on mobile devices. Using touch inputs on the screen in which the real scene and synthetic objects are displayed, users are expected to interact with the world. Depth cameras that are used to detect human motion provide more immersive interaction mechanism for AR environments. Recently, thermal touch input devices are introduced. In thermal touch devices, users directly interact with the real environment. Special cameras attached to the mobile devices detect user input from the amount of heat transferred to the surface after the touch.

2.1.2 Rendering and Occlusions

Realistic rendering of virtual entities onto real-world imagery is still a challenge. Real scenes exhibit complex surface characteristics due to illumination and material properties. Local illumination models, such as Phong illumination model, make the virtual objects look unnatural. Furthermore, in order to use even a simple illumination model, light source information must be extracted from the scene, such as the light position, color, and intensity. Similarly, realistic looking shadows, ambient effects and diffuse light interactions provide enhanced immersion for AR applications yet the existence of such effects depend on the correct lighting parameters.

Extracting the illumination information received great amount of interest from researchers. This problem is often called *Inverse Illumination Problem*, introduced by Schoeneman et al. [1]. Wang et al. [2] estimate multiple directional light sources from the objects with Lambertian reflectance. Their work relies on the existence of the three dimensional (3D) model of the objects in the scene. For light source estimation, surface shading and shadows are used. Xing et al. [3] use a user-assisted approach to estimate lighting parameters. Their methods work in real-time and adapts to changes in the illumination but it assumes that the camera movement is limited to rotations. Lopez-Moreno et al. [4] estimate the light parameters of the scene using a single image.

As a simple alternative, lighting setup of the AR scene can be constructed manually. In this case, real world lighting can be approximated by tweaking the parameters of the lighting setup, considering the light sources in the environment. Additionally, for outdoor AR scenarios, position of the Sun can be detected with the help of a Global Positioning System (GPS) receiver and compass.

Typical AR scenarios often contain motion blur due to fast camera movement. In such cases, rendering (assuming camera tracking is successful) is more challenging. With standard rendering methods, virtual entities look detached from the real environment due to blur in the background. Youngmin and Lepetit [5] apply the perspective correct motion blur effect to the virtual entities to provide

a more realistic AR experience where the camera is moving rapidly.

In order to increase the realism in AR environments, physical phenomena, such as reflections and refractions, can be utilized. Kan et al. [6] use ray tracing-based techniques to render effects, such as caustics, refraction, reflection, and depth of field, in an AR framework.

Occlusions are another important issue that must be handled for AR applications. Without proper occlusion information with respect to the camera, correct overlaying of the virtual entities is impossible. 3D models of the environment is often used to handle occlusions in AR frameworks. In its simplest form, these 3D models are used as a depth mask and their depth value is recorded to a z-buffer. After that, the virtual entities are drawn using a depth test.

2.1.3 Scene Preparation

Augmented reality scenes require varying amount of preparation beforehand.

For some frameworks, the 3D model of the environment has to be extracted beforehand in order to use the system. The 3D model can be used for tracking, handling occlusions, and it acts as a physical environment, for example, to detect and resolve collisions with the virtual objects.

Many methods (e.g., keypoint based), require texture information of the scene. Textures on the objects, along with the simple geometric information about the scene, have to be provided. These textures are processed offline and the keypoints are extracted. The extracted keypoints are used by the vision-based tracker to estimate the camera pose.

2.1.4 Sound and Haptic Feedback

AR is not limited to the visual augmentation. Ideally, full-fledged AR scenario must provide extra immersion to the user through sound and haptic feedback.

Haptic feedback is especially important and necessary for some AR applications, including medical training and military education.

2.2 Camera Localization

Camera localization is the process of determining the extrinsic parameters of the camera. The extrinsic camera parameters are the position and orientation of the camera. The camera parameters can be represented using a transformation matrix where R is the 3×3 rotation matrix and T is the 1×3 translation matrix.

$$M = \begin{bmatrix} & R & T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The camera parameters can be computed using vision-based methods and Inertial Measurement Units (IMUs). In this section, we will describe the existing camera localization methods.

2.2.1 Vision-based Methods

Vision-based methods make use of the camera image that is available to locate the camera in the scene. RGB cameras are widely used for this task. RGB-D cameras are becoming increasingly popular due to their low costs. RGB-D cameras provide additional depth information along with the color information. Depth information can be used in different ways to locate the camera in the scene.

In order to locate the camera in the scene, markers can be placed in the environment. These markers contain discriminative properties that make the tracking process fast and accurate. This type of tracking is called *marker-based tracking*. Tracking can be performed without markers where the environment itself has discriminative features like edges, key points, and rich geometry. Such tracking methods are called *markerless* (or *natural marker-based*) methods.

2.2.1.1 Edge-based Tracking

Edge-based methods rely on edges that can be detected in the video feed. Most approaches need an actual edge information of the objects to be tracked. RAPID [7] is one of the earliest model-based tracking system. In each frame, using the prior camera pose, control points on model edges are projected on the camera image. Then, the nearest edge is searched in the image in the direction of the corresponding edge normal and the distance of the nearest edge is recorded per control point. The transformation matrix that will minimize these set of distances can be calculated using a least squares solver. Rapid uses very small number of control points thus it can run on very modest hardware. Additionally, edge polarity is used to minimize the false detection of the edges and unreliable control points can be ignored to make the tracker robust in the presence of occlusions. Gennery [8] also developed a method in the same spirit as RAPID. Gennery elaborates on weight assignment to the tracking points that are sampled from the model edges. It is stated that the tracking points that are easily and uniquely detectable are more reliable, and hence, their associated weight values must be higher.

After RAPID, different edge-based tracking systems are developed. Comport et al. [9] use a very similar edge-based method for tracking. In order to detect edges in the image, they use the moving edge detector developed by Bouthemy [10]. They emphasize that tracking with moving edges are more reliable because sample-point-based trackers might falsely follow different features or noise in the video feed. The tracker developed by Petit et al. [11] uses edge features to improve robustness. The color statistics around the edge are gathered dynamically and used for verification in the successive frames to prevent mismatches between the model edges and the actual edges.

Wuest et al. [12] propose an adaptive learning strategy for edge features to eliminate edges detected incorrectly. Learning is carried out at sample point level and each sample point stores its appearance on the one-directional search path. To make their system work properly, they generate sample points adaptively in a tree-like manner. Sample points are directly assigned in the 3D model itself

and they are projected onto the image until a relatively equal distribution on the projected edge is accomplished. The system selects the best edge location based on the prior appearance information when multiple edges are detected on the search path of the sample point. The accurate detection of edge locations increases the accuracy of the tracker.

2.2.1.2 Keypoint-based Tracking

Keypoint-based tracking involves detecting points that have distinctive properties. Such keypoints are invariant to rotation, scaling and illumination changes to a certain degree. Therefore, keypoints can be detected over different frames and the camera motion can be estimated using the correspondences between the frames.

In the literature, there are different keypoint detection algorithms. Scale-Invariant Feature Transform (SIFT) is one of the standard keypoint detection methods [13]. Difference of Gaussians (DoG) are used to locate the keypoints in the image. Then, a set of orientation histograms are calculated around each keypoint. Using these histograms, a highly descriptive feature vector of 128 elements are formed. Speeded Up Robust Features (SURF) is another keypoint detector (and descriptor) [14]. It is faster than SIFT yet still very robust. Image operations are performed on precalculated integral images; this increases the performance dramatically. Features from Accelerated Segment Test (FAST) is a feature detector based on a very simple mechanism [15]. It uses pixels that form a ring around the keypoint with a small radius, thus, very few pixels have to be queried. FAST performs faster (only for keypoint detection) than both SIFT and SURF and it is very suitable for real time applications. Features from Accelerated Segment Test-Enhanced Repeatability (FAST-ER) [16] is an enhancement over FAST, which increases the repeatability rate of the FAST detector. Ferns is a keypoint classification technique that is used to classify points that exhibits a planar structure [17, 18]. Using Ferns, keypoints are classified using binary intensity comparison tests with the points around the keypoint specified by two

random offsets. It provides a scalable solution for keypoint matching. The accuracy of the method can be increased by performing more binary tests with the expense of computational cost and vice versa. Binary Robust Independent Elementary Features (BRIEF) [19] is a feature descriptor that is inspired by Ferns. The classification depends on the efficient binary tests as in Ferns, yet in BRIEF, the feature descriptor is encoded using a bit vector.

Wagner et al. [20] state that the SIFT and Ferns features can be combined for efficient and robust tracking. They use modified versions of SIFT and Ferns, which is more suitable for devices with low computational power. Template-based tracking is another approach that they fuse into their tracker. It is stated that the orthogonal strengths and weaknesses between the template-based tracking with SIFT and Ferns makes the combination of three even more robust for tracking purposes.

Frame-to-frame correspondences that are calculated using keypoints can be used to estimate the camera motion. RANSAC [21] is a fundamental technique to remove outlier measurements that is the result of falsely matched keypoint pairs among the frames. Keypoints can be used to calculate the optical flow among frames. The optical flow is then used to estimate the camera motion.

Commercial AR frameworks such as Vuforia [22] and Metaio [23] extract features on texture-rich objects with a well-defined structure, e.g., cubes, cylinders, planes. Because the geometry of the objects are known beforehand, the keypoints that are extracted can be located in the 3D scene. These systems then estimate the camera pose by searching and matching these feature points in the video feed. It should be noted that the keypoints may not be detectable in some environments. Feature extraction on textureless environments is not possible; the tracking methods that depend on the existence of keypoints fail in such scenes.

2.2.1.3 Depth-based Tracking

Recently, depth sensors become widely available and cheap as consumer products. As a result, depth sensors become an effective tool for camera localization.

A typical depth sensor provides the depth information for a scene. The depth information can be used for both camera localization and for environment mapping. With high computational power of the modern hardware, it is possible to perform environment mapping and camera localization simultaneously. Such methods are called Simultaneous Localization and Mapping (SLAM). In a typical SLAM scenario, the depth camera is moved and rotated around the scene slowly. With the incoming depth data, the parts of the scene are constructed. As the camera moves around, the new depth data arrives. Because the camera movement is slow, the new depth data partly overlaps with the old data. Similar depth data is processed so that the correspondences are found between successive frames. These correspondences are used to find the relative camera transformation across frames. The depth data that does not resemble the depth data of the previous frame is considered as new data and added to the environment data. This method facilitates the reconstruction of the real scenes.

During SLAM, the depth information is often processed as a point cloud. The correspondences between the two point clouds can be calculated using the Iterative Closest Point (ICP) algorithm or other similar algorithms. Robust tools are available for SLAM. Kinect Fusion [24] is such a tool that is used for scene reconstruction. Six DOF camera parameters are tracked in real-time using Graphics Processor Unit (GPU) supported ICP algorithm while the 3D scene is constructed. Kinect Fusion can export the underlying scene data as a triangular mesh, which can be used for various purposes.

SLAM approach can also be employed using a standard RGB camera. Davison [25] uses Structure-from-Motion (SfM) approach for this purpose. Starting with a small number of keypoints, the camera is tracked and the new features are added to the environment. As opposed to the previous approaches, Davison's method works in real-time, thus very suitable for interactive AR applications. Newcombe et al. [26] extend this work by generating a simple mesh representation of the environment and constructing a high resolution mesh of the same environment. High resolution mesh generation makes it possible to handle occlusions and physical interactions accurately. Salas-Moreno et al. [27] extend the SLAM paradigm to the object level. Pre-existing 3D object database is used to

locate the objects and track the camera in real-time.

2.2.2 Inertial Measurement Units

Modern tracking systems often use extra sensory devices for more accurate and robust tracking. Accelerometer measures acceleration, which can be used to locate the camera in an AR scene. Gyroscope measures the rotation rate, which is useful to determine the orientation of the camera for AR applications. GPS sensor is another device that helps locating the camera. The positioning accuracy of the GPS is relatively low, thus it is only used in applications that does not require the exact location information. Many mobile devices are equipped with similar sensory devices; the tracking systems depend on them for accurate and robust camera localization along with the vision-based methods. The majority of these devices do not provide absolute position and orientation values. The position and orientation information is often calculated through integration of the relative values supplied by the sensory devices. This causes drift in measurements as the tracking continues. Such characteristics of the sensory devices make them inaccurate. To alleviate this issue, the measurements of the sensory devices are combined with the measurements of the vision-based methods.

2.2.3 Hybrid Approaches

Hybrid approaches combine different tracking techniques to provide robust and efficient tracking. Rosten et al. [28] combine edge and keypoint-based tracking methods so that the resulting tracker becomes more reliable. They use FAST keypoint detector for high performance detection of the features. State et al. [29] combine the magnetic tracking and the vision-based tracking. Their tracking system uses magnetic tracking devices for rough pose estimation. The accuracy of the estimation is further improved by using a vision-based tracker. Their vision-based tracker use simple circular shaped and colored markers.

Hybrid approaches use different methods to combine the measurements of different trackers and devices. One of the most popular algorithms for this purpose is the Kalman filter [30]. Essentially, the Kalman filter combines different input measurements and produces a more reliable output measurement. Every input measurement is given a weight, thus have a varying effect on the final measurement. The input measurement weights are dynamically adjusted according to the variance in the input measurements. The Kalman filter is a linear filter; its Non-linear variant is called the Extended Kalman Filter (EKF). The EKF is used in many tracking methods to combine different sensory inputs for robust and accurate camera localization. Reitmayr et al. [31] use edge-based tracker along with the gyroscope, gravity, and magnetic field measurements. The measurements are combined using an EKF for robust and accurate tracking. Similar to the previous approaches, Klein et al. [32] combine the inertial measurement units and visual tracking. However, their method differs on the usage of the inertial sensor. They use inertial sensor output to predict motion blur. They adjust the edge search method according to the motion blur direction so that the edge localization is robust and accurate under blurry video feed.

2.2.4 Markers

Markers are gizmos or textures that can be detected and identified easily by the sensors. Most AR systems needs markers to be placed in the environment so that the camera pose can be estimated. *Active markers* are simple and compact devices that emits light. Special cameras designed for that purpose can detect and identify these markers. *Passive markers* are generally feature-rich objects or simple colorful objects that can be detected and tracked easily. Passive markers are tracked by the vision-based systems. In some cases, passive markers may need to be placed in the environment. Alternatively, the scene itself can be used as a marker (natural marker). Such systems are more preferable because the resulting AR experience is more natural.

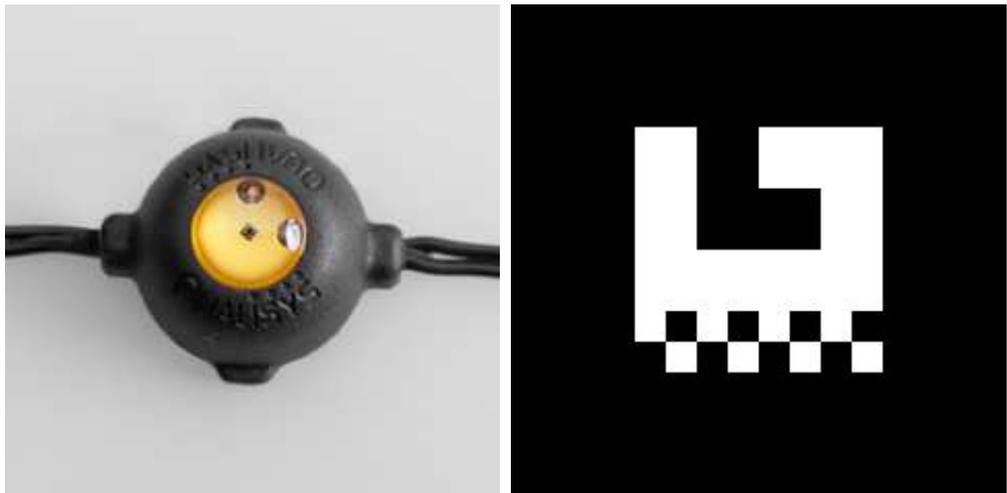


Figure 2.1: Types of markers: active marker (left), passive marker (right)

Chapter 3

Tracking System

Our tracking system is based on the methods described in [7, 8], which uses visible edges as features to be tracked. In a very broad sense, the screen space algorithm is used to minimize the distance between the projected edges of the model and the edges that are located in the image retrieved from the video feed. The minimization is performed using a least squares solver. The following sections describe the tracking system in detail.

3.1 Preprocessing

We assume that the coarse 3D model of the environment is available as a collection of triangular faces. The 3D model is analyzed and edges are extracted from the model. Edge detectability in the video feed is often correlated with angle difference of the normals of the neighboring faces. Thus, the edges that are shared among faces can be ignored if these faces are on the same plane or the angle between normals of these faces are below a certain threshold. This angle threshold is denoted as A_{min} . Additionally, edges that are too short or too long can be ignored using the threshold values L_{min} and L_{max} . Extracted edges are then used by the tracking algorithm for camera pose estimation. Figure 3.1 shows a sample 3D model where all edges of the model are shown in green and the salient

edges that are extracted in the preprocessing step are shown in red.

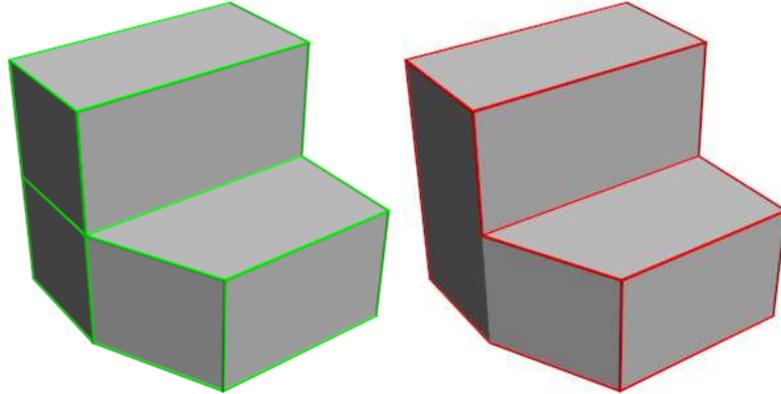


Figure 3.1: Sample 3D model: all edges (left), salient edges (right)

3.2 Initialization

Model edges are not directly associated with their counterparts in the video feed. Additionally, the edge search is performed in a small range in the direction of the projected edge normal. Thus, camera must be positioned and oriented before tracking so that the projected edges of the model should coarsely match their counterparts in the image.

3.3 Real-time Tracking

The camera pose is recalculated every time a video feed is updated. Upon arrival of a new image through the video camera, the steps followed are as follows.

3.3.1 Model Projection and Control Point Generation

When a new image is retrieved from the video feed, the 3D model is projected onto the image using the last estimated position and orientation of the camera. The matrix M for this transformation is calculated using Equation 3.1.

$$M = WPV, \quad (3.1)$$

where V is the view matrix (the inverse transformation matrix of the camera). This matrix is calculated using the previous estimate of the camera parameters (position and rotation). P is the projection matrix that transforms points in the view-space to the normalized device coordinate space. W is the viewport matrix that converts normalized coordinates into the coordinates in the screen space.

It should be noted that some edges of the model will be outside of the viewing frustum of the camera. Such edges need to be culled and clipped before projection. For this purpose, edges that are completely outside of the viewing frustum are culled. Edges that are partially outside the viewing frustum are clipped. Clipping and culling can be performed by using simple intersection tests.

After the edges in the viewing frustum are determined, they are projected onto the camera image with the matrix M . The projected edges are then sampled at regular intervals as control points and the visibility of these points are determined by using raycasting, which is performed on the available 3D model of the scene. Once all visible control points are determined, the algorithm proceeds with the edge localization.

3.3.2 Locating Edges in the Video Feed

For each control point, an intensity discontinuity is searched in the image. Let C_i be the i^{th} control point that is projected. The search direction for control point C_i is denoted as \vec{N}_i . The search direction for a control point is the normal vector of the edge (in screen space) that contains control point C_i . Starting from the location of control point C_i , pixel intensity discontinuities are searched along the path in the direction of the vector \vec{N}_i . The intensity of a pixel is the weighted sum of its RGB components. The intensity for pixel p is calculated using Equation 3.2.

$$I_p = 0.29R_p + 0.59G_p + 0.12B_p. \quad (3.2)$$

For performance considerations, we use an approximation of Equation 3.2, as given in Equation 3.3.

$$I_p = 0.25R_p + 0.5G_p + 0.25B_p. \quad (3.3)$$

Equation 3.3 can be realized using bit shift and add operations; this leads to a performance gain in the implementation. Before searching for an edge, the RGB pixel buffer (three element per pixel) can be converted into pixel intensity buffer (one element per pixel) or pixel intensities can be calculated on demand. Both approaches have small performance implications with respect to the parameters such as the number of control points and the resolution of the video feed. Since the normals are not necessarily parallel to the x or y axes in the image coordinates, the queried locations might not fall exactly on the center of a pixel. Different methods can be used to handle this issue:

The naive approach is to directly use the pixels that contain the sampled points along the search path. In this approach, the pixel intensity values might be duplicated because successive points that are sampled along the search path may fall on the same pixel.

Another approach is to start from the center of a pixel of the initial sample point and to use the search direction that is rounded to the nearest 45 degrees. This method guarantees that the points that are sampled along the search path will fall exactly on the pixel centers [33].

Alternatively, the bilinear interpolation can be used to calculate the pixel intensity around the point. The bilinear interpolation averages four neighboring pixels, therefore it eliminates image noise to some extent. This method is computationally more expensive than the previous two approaches because four pixels are fetched instead of one and a weighted averaging is performed.

The image noise in the video feed can interfere with the edge detection. To alleviate this issue, smoothing filters are used. After the image is processed with

a smoothing filter, edge search is performed in the smoothed image. Since only the pixels that are on the search paths are queried, a full frame smoothing can be avoided. For that purpose, the filtering is only performed while querying a pixel. However, when a control point is projected to a large area, full frame filter can be more efficient since such filters benefit from cache locality during image processing.

For better localization of the edges in the video feed, only local peaks of the intensity differences are taken into the account. For this purpose, the previous intensity differences have to be recorded during edge search. To make the edge localization process robust to image noise and other features that may be reported as intensity discontinuities, a threshold value is used. The intensity difference values below this threshold are ignored.

Once such an intensity discontinuity is found between two consecutive points on the search path, the halfway point between these two points is recorded as a intensity discontinuity position for the corresponding control point. This point is denoted by T_i . If such a discontinuity cannot be found, then the corresponding control point is excluded from further consideration. Figure 3.2 shows the model edges, control points and the located edges in a video feed.

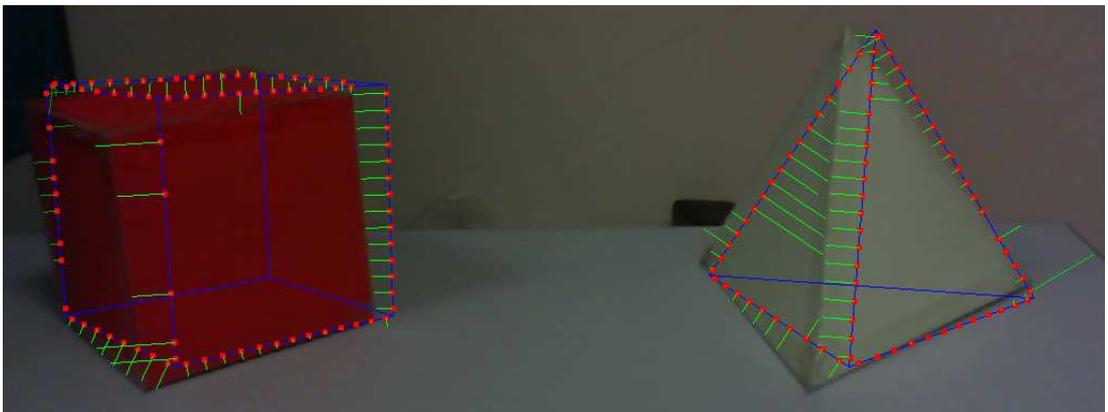


Figure 3.2: Locating edges in the video feed: model edges (blue), sample points (red), sample point - located edge pairs (green)

3.3.3 Formulating the Least Squares Problem

The correct camera pose can be represented by a transformation matrix relative to the previous camera pose, which align all control points with their target discontinuity points detected in the video feed. Thus, the correct camera pose will minimize the expression in Equation 3.4.

$$\sum_{i=1}^n |S_i - T_i|^2, \quad (3.4)$$

where n is the number of control points that have a valid intensity discontinuity position associated with it. The camera pose is represented with a column matrix (X) of six parameters. The matrix X is the combination of the relative translation and rotation values:

$$X = \begin{bmatrix} T_x \\ T_y \\ T_z \\ R_x \\ R_y \\ R_z \end{bmatrix}.$$

Each parameter is associated with a transformation of the control points along the principal axes in a certain amount. For six different unit motions of the camera, these transformations have to be calculated and stored in a different matrix. For example, the unit translation along the y axis can be represented as follows.

$$X = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The new transformation matrix is formed using the relative transformation matrix X . The translation amount of the control point in the direction of the corresponding edge normal is calculated using Equation 3.5.

$$M_i^j = (C_i^j - C_i) \cdot \vec{N}_i, \quad (3.5)$$

where C_i^j is the new location of the i^{th} control point under the transformation generated by the j^{th} unit camera motion (only the j^{th} element of the matrix X is one and the remaining elements are zero). It should be noted that these transformations are not linear with respect to the elements in the camera parameter matrix X . However, it is assumed that they are linear for small motions of the camera. This assumption enables us to use the simple least squares solver to estimate the camera parameters.

Equation 3.4 is used to populate the measurements matrix A , which is given below. $M_{i,j}$ stands for the i^{th} control point motion that is generated by the j^{th} unit camera motion where n is the total number of control points that are sampled on the projected edges.

$$A = \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} & M_{1,4} & M_{1,5} & M_{1,6} \\ M_{2,1} & M_{2,2} & M_{2,3} & M_{2,4} & M_{2,5} & M_{2,6} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ M_{n,1} & M_{n,2} & M_{n,3} & M_{n,4} & M_{n,5} & M_{n,6} \end{bmatrix}.$$

Signed distances between the sample point positions and the corresponding intensity discontinuity positions are used to populate the matrix B . The distance D_i for a control point C_i is calculated as follows.

$$D_i = (T_i - C_i) \cdot \vec{N}_i. \quad (3.6)$$

The matrix B then can be expressed as follows:

$$B = \begin{bmatrix} D_1 \\ D_2 \\ \cdot \\ \cdot \\ \cdot \\ D_n \end{bmatrix},$$

where n is the number of sample points that have a valid intensity discontinuity position associated with it. After matrices A , X , and B are populated, the following linear system needs to be solved.

$$AX = B. \quad (3.7)$$

Since the system is overdetermined ($n > 6$) there will not be an exact solution. Therefore, the objective function given in Equation 3.4 should be minimized. Additionally, weights can be incorporated to this linear system. The weights determine the effect of the measurements on the solution. Measurements with larger weights have more impact on the final solution and measurements that have a zero weight are ignored. To use the weighted least squares, Equation 3.4 should be modified as follows:

$$\sum_{i=1}^n w_i |S_i - T_i|^2, \quad (3.8)$$

where w_i is the weight of the i^{th} measurement. Now, the system of linear equations to be solved becomes

$$A W X = W B, \tag{3.9}$$

where W is the diagonal matrix populated with the weights assigned to the measurements. The matrix W is given as follows.

$$W = \begin{bmatrix} w_1 & 0 & 0 & \cdot & \cdot & \cdot \\ 0 & w_2 & 0 & \cdot & \cdot & \cdot \\ 0 & 0 & w_3 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & w_n \end{bmatrix}.$$

Weights can be assigned using different methods. It is often convenient to assign weights according to the reliability of the measurements. As it is described in Drummond’s work [34], measurement weights can be linearly reduced to zero as they approach to the image boundaries. This prevents rapid changes to the estimation of the camera pose due to appearing/disappearing edges near image borders.

Chapter 4

Improved Tracking

The system described in the previous chapter is the bare bones of the edge-based tracking system. In the following section, persistent control point scheme is explained in detail and a method for projecting persistent control points in screen space is given. After that, constant time visibility determination mechanism is proposed which makes use of persistent control point scheme.

4.1 Persistent Control Point Generation and Projection

Many typical edge-based tracking systems use on-the-fly control point generation. Control points are assigned to the edges in screen space at regular intervals during runtime and measurements are made for these points. Such control points are not persistent across frames. We use an persistent control point generation scheme for reasons that will be described. This mechanism is first proposed by [12]. The persistent control point generation scheme relies on preassigned control points in the world space. In the preprocessing step, after salient marker edges are identified, edges are sampled in recursive manner until distance between two control point is below a certain threshold. This threshold is called as *minimum*

world sampling interval, which is defined in the world space units. Figure 4.1 shows the control points generated for two marker objects. The control points are then stored in an array, starting from the root control point, similar to a heap array storage scheme (see Figure 4.2).

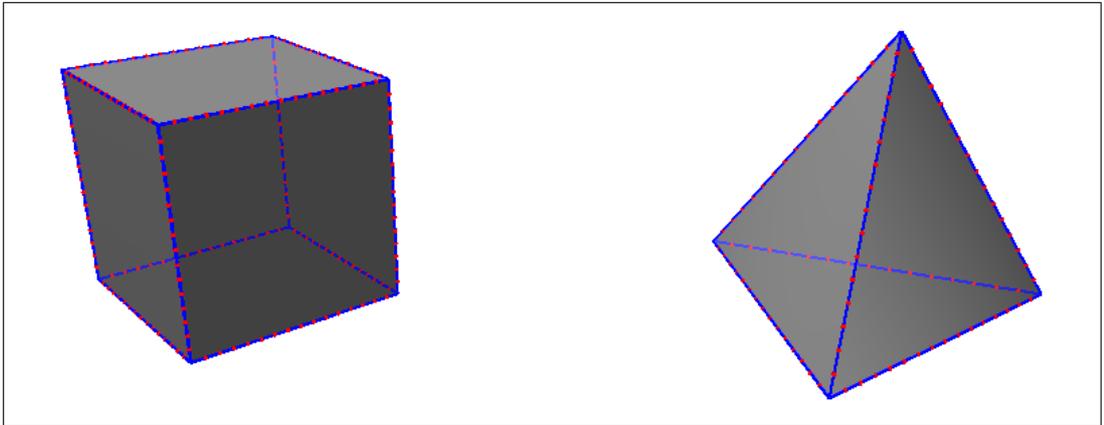


Figure 4.1: The control points generated for two marker objects at the preprocessing stage.

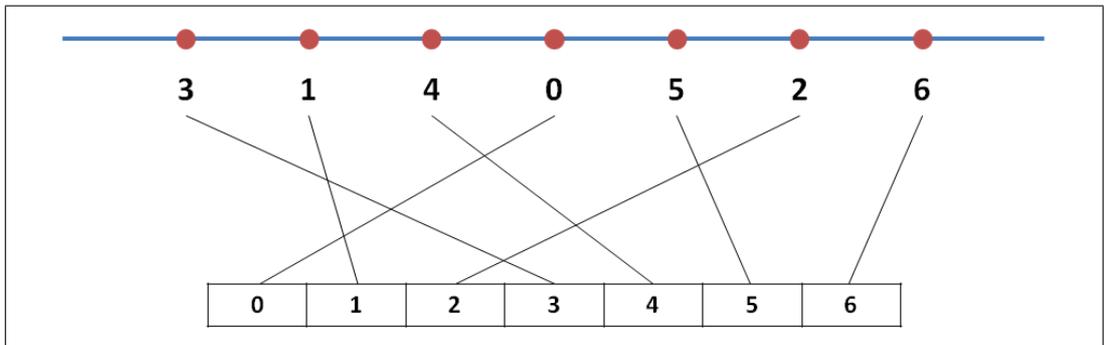


Figure 4.2: Control points generated using the persistent control point generation scheme and their storage in the array. The numbers below the points are the depth of the points in the tree structure and the numbers in the boxes indicate the order of generation.

During tracking, the precomputed control points are projected onto the video feed using an adaptive strategy so that control points are evenly distributed along the projected marker edge. Wuest et al.[12] determine the number of control points that are to be projected considering the projected length of the marker edge. Different from their approach, we use an adaptive projection scheme so that the control points are evenly distributed when distortions due to perspective projection are prominent. These approaches are demonstrated in figure 4.3.

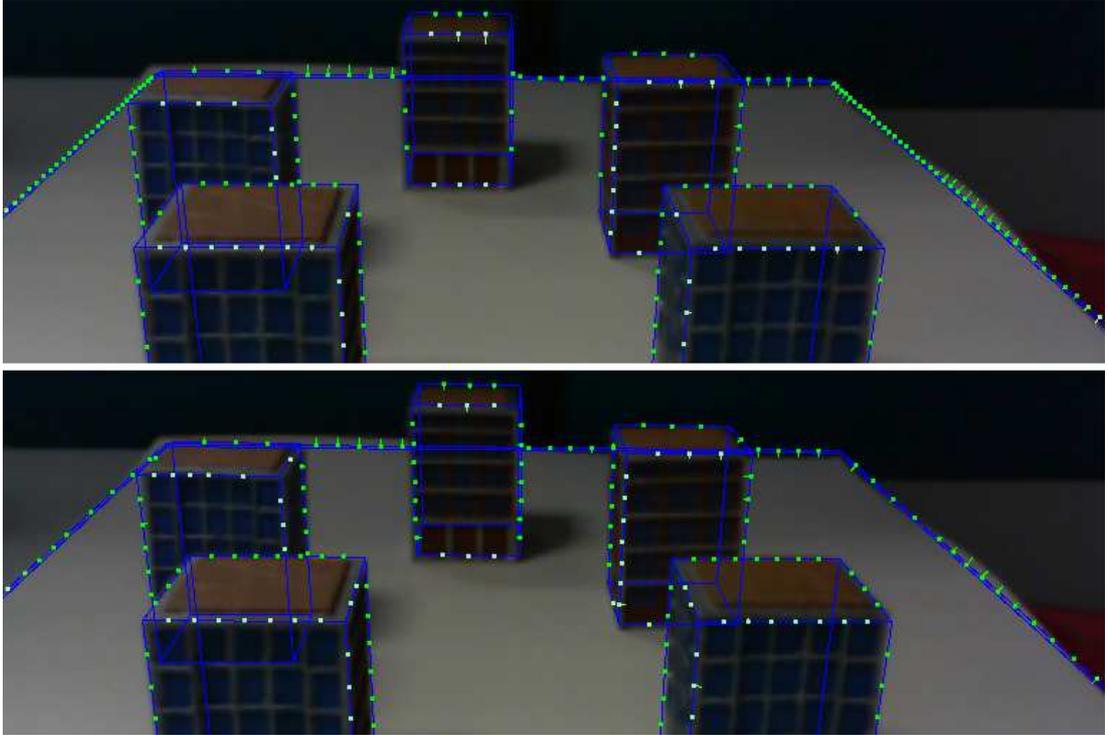


Figure 4.3: Control points projected using using fixed marker edge length (top) and control points projected using adaptive approach (bottom)

The effects of such distortions depend on the tracking environment. If the camera is focused on a small marker object which is centered on the screen, the adaptive projection strategy is not needed. However, when the environment is used as a marker surrounds the camera (e.g., the room as a natural marker), this approach generates more evenly distributed control points. We project a subset of the control points onto the video feed to be used for tracking (see Figure 4.4).

The adaptive control point projection algorithm uses a threshold value in pixels called *minimum screen sampling interval*. The algorithm starts by calculating the projected length of the initial marker edge. If the projected edge length is above this value, the first control point is projected and stored for further calculations and the marker edge is divided into two equal pieces. The algorithm then continues in a recursive manner for these edges until the termination criteria is met (all of the projected subdivision edge lengths are below the threshold). If no precomputed control point exists for the line segment, the algorithm terminates

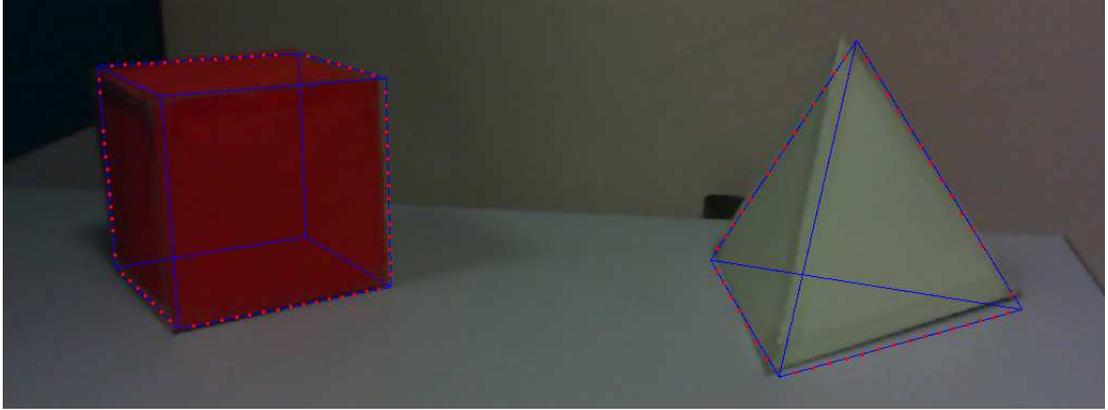


Figure 4.4: A subset of the control points are projected onto the video feed and used for tracking.

for that node. Additionally, if start and end points of the line segment lies in the same region in space which is defined by the frustum planes, the algorithm terminates since none of the control points in that segment will be visible on screen.

The persistent control point generation scheme has some advantages over an on-the-fly control point projection scheme:

Precomputed control points are permanent and extra information can be stored along these control points. Extra information can be used to increase the performance, accuracy, and robustness of the tracker.

Precomputed control points have fixed positions in the world space. This property allows view dependent information to be stored easily. For example, visibility tests can be cached using this approach. This can be achieved by storing visibility state of the points in some space subdivision structures like 3D grid or spatial trees.

4.2 Exploiting View-dependent Information for Camera Tracking

Control points have view-dependent features that can be used to improve camera tracking. We cache the view dependent properties of the control points by using a simple axis aligned 3D grid. The boundaries of the 3D grid are specified by the user (application developer) by considering the expected movement limits of the camera. Similarly, the cell size of the grids can be specified by the user but it is possible to adjust grid cell size automatically with respect to the memory requirements. Below, visibility determination process for a control point is given which is used to reduce computational cost of the camera tracker.

4.2.1 Visibility Information

Visibility tests are performed at every frame for the candidate control points for projection. For a small number of control points, this operation has little overhead. However, when number of control points is high, it has a detrimental effect on the system performance. To make visibility tests faster, the visibility state of each control point can be cached for different camera configurations.

After the construction of the 3D grid, a ray is cast from the center of each cell to all control points originating from that cell. If the ray hits a surface before arriving at the position of the control point, that control point is marked as invisible. Otherwise, the control point is marked as visible. Because the visibility tests are performed for all control points, a bit mask of n bits is allocated for each cell where n is the total number of control points generated for the marker during the preprocessing stage. Each control point is assigned a unique id between 0 and $n - 1$. The size of the grid cell determines the accuracy of the visibility caching. Small cells provide more accurate visibility information while increasing the memory usage and computation time. To reduce the memory consumption, the bit masks that store the visibility information can be merged. The cells that are close to each other have often identical bit masks. By detecting these bit

masks, duplicate ones can be removed, thus reducing the memory consumption. Such merging procedure can be performed in a similar way as the bottom-up octree construction. During the 3D grid construction, some cells overlap with the scene geometry that is used as a marker. These cells are ignored during the 3D grid construction to avoid additional memory usage because the visibility information in these cells are invalid.

During tracking, visibility information can be queried by first finding the cell that the camera belongs to, and then querying the bit associated with the control point. Finding the cell that contains the camera location is a constant time operation because a uniform 3D grid is used. Cell indices i, j, k can be calculated using Equation 4.1.

$$\begin{aligned}
 i &= \lfloor (C_x - O_x)/s \rfloor \\
 j &= \lfloor (C_y - O_y)/s \rfloor \\
 k &= \lfloor (C_z - O_z)/s \rfloor
 \end{aligned} \tag{4.1}$$

where C is the position of the camera, O is the origin of the 3D grid, and s is the cell size. If the camera is out of the boundaries of the 3D grid, cached visibility tests cannot be performed. In this case, we use online raycasting as a fallback method to determine the visibility of a control point.

In an AR simulation environment, some extra occluders can be defined. These occluders are not treated as natural markers (due to their characteristics, such as lack of trackable edges). Such objects are also included in the visibility calculations and contribute to the visibility determination for the control points.

Once 3D grid is constructed to store visibility states of the control points, it can be used for tracking purposes multiple times until scene configuration (3D grid parameters and control point generation parameters) is changed. Otherwise, visibility states for the control points must be calculated again.

4.3 Additional Details of the Tracking System

We discuss some methods concerning the real-time tracking process, which are used to improve camera tracking.

4.3.1 View Frustum and Occlusion Culling

Before control points are projected onto the video feed, invisible parts of the edges should be determined. A visibility of an edge is affected by two factors.

Frustum culling: Edges that are outside of the view frustum are ignored. If an edge is partly visible in the view frustum, it is clipped to the view frustum. Then, only the control points which are in the visible part of the edge are projected onto the video feed.

Occluder geometry: Edges can be occluded by marker geometry and extra occluder geometry. For such cases, visible edge parts are not calculated. Instead, visibility tests are carried out for control points directly.

Additionally, the edges that connect two back-facing triangles are ignored because they are invisible to the camera. These edges satisfy the following inequalities (Equation 4.2).

$$\begin{aligned} \overrightarrow{(P_E - P_c)} \cdot \vec{N}_a &> 0, \\ \overrightarrow{(P_E - P_c)} \cdot \vec{N}_b &> 0, \end{aligned} \tag{4.2}$$

where P_c is the position of the camera, P_E is the center of edge E in the world coordinates, and \vec{N}_a and \vec{N}_b are the normals of the faces that meet on edge E .

4.3.2 Silhouette - Non silhouette Edges

Silhouette edges separate the object from its background. The edges that form the silhouette of a polygonal object is the subset of the edges of the polygonal object. An edge is a part of the silhouette, if it satisfies the following inequality (Equation 4.3).

$$((\overrightarrow{P_E - P_c}) \cdot \vec{N}_a) ((\overrightarrow{P_E - P_c}) \cdot \vec{N}_b) < 0, \quad (4.3)$$

where P_c is the position of the camera, P_E is the center of edge E in the world coordinates, and \vec{N}_a and \vec{N}_b are the normals of the faces that share edge E . Note that Equation 4.3 may not be valid if the object is not convex.

In most cases, silhouette edges can be detected reliably because they separate the object from the background. Therefore, control points that belong to silhouette edges can be assigned with higher weight values. For silhouette edges, weight value of the corresponding control points are set as 1 while for non-silhouette edges, weight value is set to 0.5.

It should be noted that some edges that do not pass the saliency threshold in the preprocessing stage can still be a silhouette edge in a different camera configuration. Thus, during the projection stage, all edges must be considered as a candidate for the control point generation. If an edge is both non-salient and non-silhouette, it is ignored. But, if it is a silhouette edge, regardless of its saliency value, the control points of the edge are projected onto the video feed and used in the camera pose estimation. Figure 4.5 illustrates various types of edges for a cylinder object.

4.3.3 Motion Prediction

During camera pose estimations, the camera position and orientation can be used to calculate the linear and angular velocity of the camera. The linear and angular velocities are then used to predict the camera parameters in the next frame. This

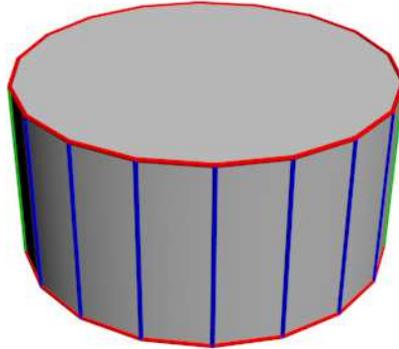


Figure 4.5: Various types of edges for a cylinder object. Blue: non-salient, non-silhouette edges; green: non-salient, silhouette edges; red: salient edges.

is beneficial when the camera moves and rotates at a high speed, which may cause the tracker to fail because the edge search in the video feed is limited to the local neighborhood.

4.3.4 System Overview

The flow diagram of the camera parameter estimation process with persistent control point generation and exploiting view-dependent information is depicted in Figure 4.6.

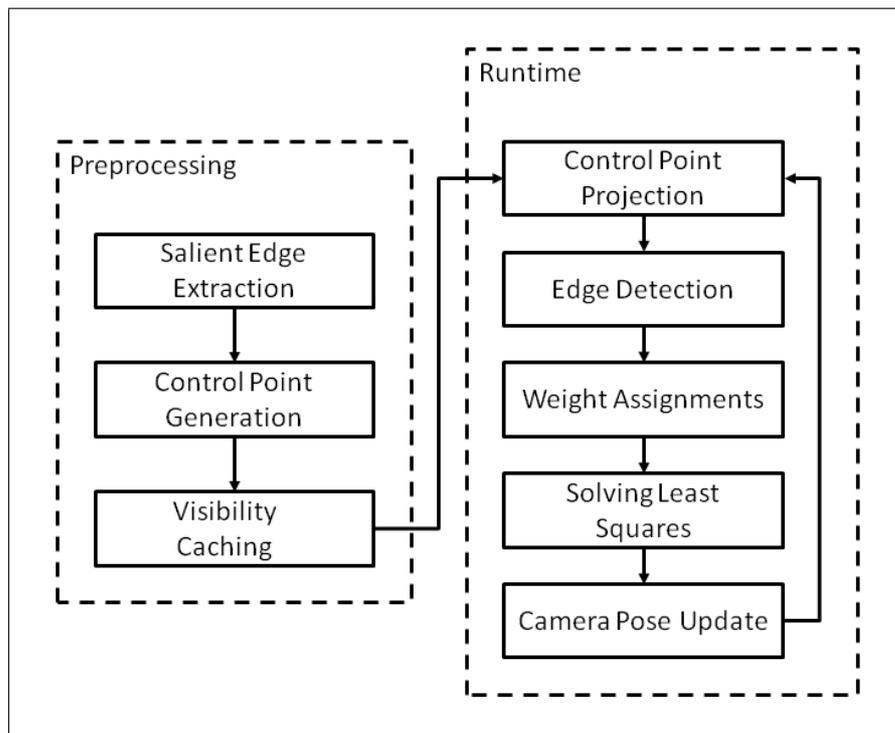


Figure 4.6: The flow diagram of the improved camera tracking system.

Chapter 5

Evaluation

We evaluate the proposed camera tracking approach in terms of cached visibility determination during the preprocessing and runtime tracking stages.

5.1 Cached Visibility Tests

Cached visibility testing method is evaluated for its preprocessing cost, tracking accuracy and performance. Details of these evaluation methods are given in the following sections.

5.1.1 Preprocessing

In order to evaluate the preprocessing cost of the cached visibility tests, different marker setups are prepared with varying *minimum world sampling intervals*. The 3D grid for visibility caching is constructed using different cell sizes and preprocessing times are measured for different setups. Additionally, memory requirements of these setups that are built with different parameters are calculated.

5.1.2 Tracking

To measure camera tracking accuracy, two videos are recorded where the camera is moved and rotated in the AR environment and tracking is performed on pre-recorded video sets. Then, estimated camera parameters are recorded for each frame without using cached visibility tests and considered as a baseline measurement. After that, camera tracking is performed with cached visibility tests on a 3D grid constructed with different cell sizes and estimated camera parameters are recorded for each frame. Estimation errors are recorded per frame with respect to the baseline measurement. Additionally, mean square errors (MSE) are calculated for these measurements with respect to the baseline measurement. Finally, the average processing costs per frames are presented for both online and offline visibility determination methods. The parameters that affect the cost and accuracy of the cached visibility tests are summarized below.

Marker and occluder geometry: Marker geometry determines the number of salient edges thus have an effect on the number of generated control points. Preprocessing time and memory usage increase as number of generated control points increases since visibility tests are performed for each control point.

Minimum world sampling interval: The number of generated control points are determined by the minimum world sampling interval, thus has an effect on the preprocessing time and memory usage.

3D grid and cell size: Visibility tests are performed for each cell and number of cells are determined by the 3D grid and the cell size. Smaller cells yield more accurate camera tracking but increases the preprocessing time and memory usage.

5.2 Sample AR Application

To demonstrate camera tracker's performance and usability in AR environments, we developed an augmented reality application within our framework where the user can interact with the environment by creating virtual entities which physically interact with the environment. For this purpose, marker geometry is defined as a collision geometry. Similarly, to handle occlusions for rendering, marker geometry is also defined as an occluder.

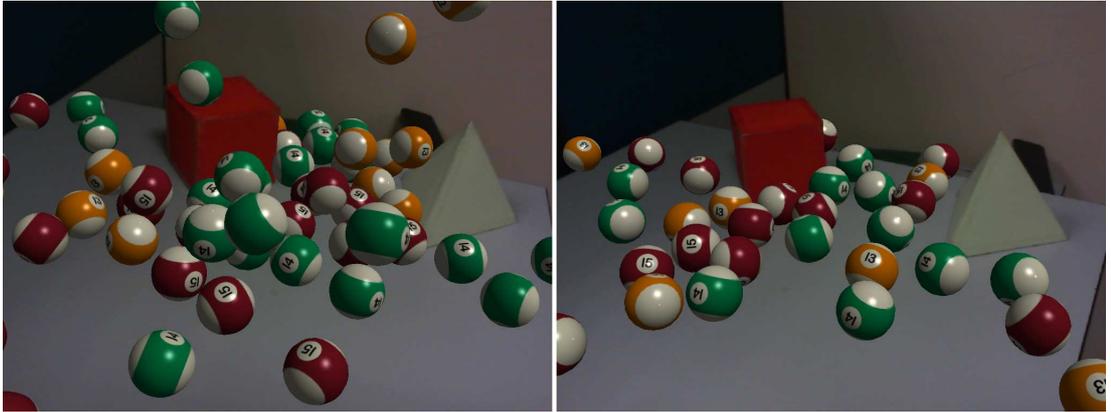


Figure 5.1: Sample augmented reality application with physics interaction and occlusion handling.

Chapter 6

Experimental Results

6.1 Test Environment

The videos used in the experiments are recorded using a standard web-cam with display resolution of 640×480 at 30 fps. Tests are performed on a machine with Intel i7-3840QM (8Mb Cache, 2.8Ghz Clock) processor, 16Gb Ram, Nvidia Quadro K3000 Graphics Processing Unit (GPU). The Augmented Reality framework is completely built on the Unity Game Engine.

For evaluation purposes, two scenes with natural markers are prepared. The first scene contains two simple objects (a triangular prism and a cube) on a table. The marker objects in the first scene contain 36 triangles. In the salient edge extraction stage, 30 edges are extracted. The second scene contains five miniature building models on a base. The marker objects in the second scene contain 92 triangles. In the salient edge extraction stage, 72 edges are extracted. These scenes are shown in Figure 6.1. The 3D models of these environments that are used as a natural markers are shown in Figure 6.2. The size of the 3D grid constructed for visibility caching in both scenes is $1.0 \times 0.5 \times 1.0$.

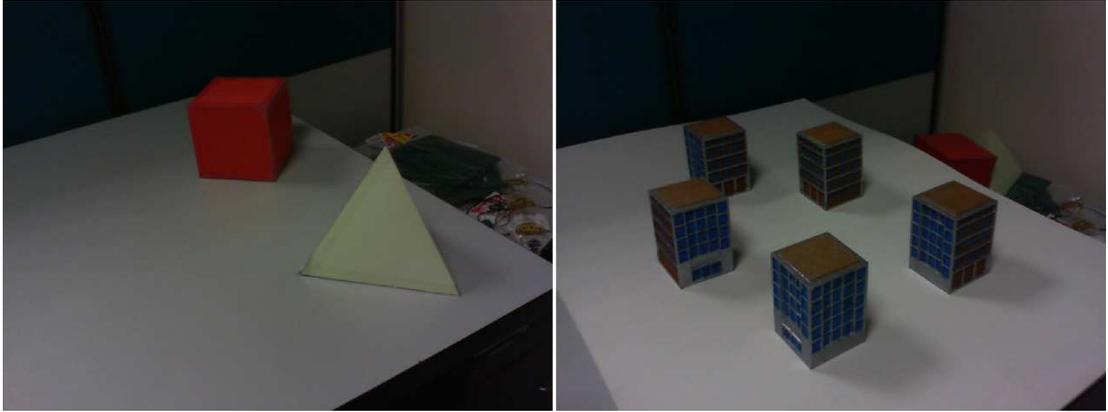


Figure 6.1: The scenes used to test the camera tracker.

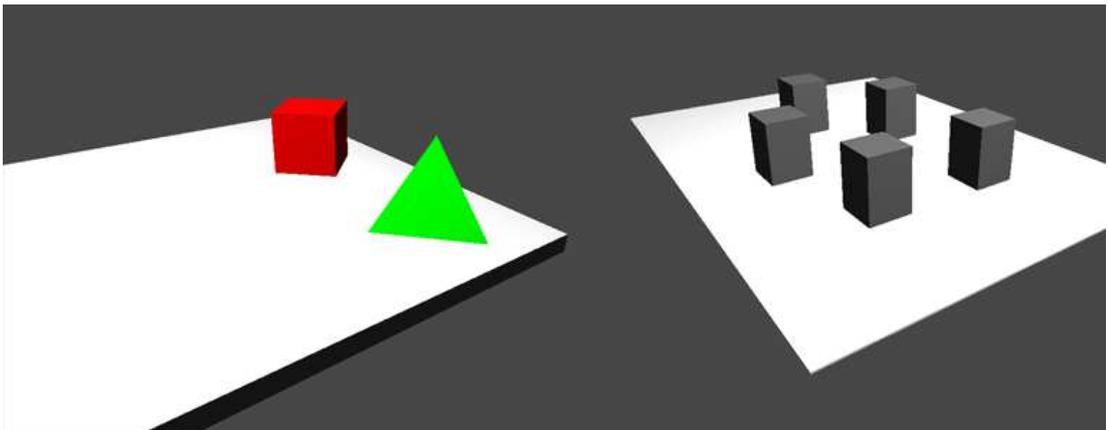


Figure 6.2: 3D models of the scenes that are used as natural markers for camera tracking.

6.2 Cached Visibility Tests

6.2.1 Preprocessing

Table 6.1 presents the preprocessing times of visibility caching for the first scene. Table 6.2 shows memory requirements for the first scene. Table 6.3 presents the preprocessing times of visibility caching for the second scene. Table 6.4 shows memory requirements for the second scene.

		minimum world sampling interval				
		0.0050	0.0750	0.0100	0.0125	0.0150
cell size	0.025	45.197	23.743	22.343	12.025	11.351
	0.050	5.583	3.061	2.758	1.489	1.431
	0.750	2.005	1.052	0.959	0.537	0.492
	0.100	0.706	0.369	0.348	0.187	0.182

Table 6.1: Visibility caching preprocessing times (in seconds) for the first scene with respect to different 3D grid cell sizes and minimum world sampling interval values

		minimum world sampling interval				
		0.0050	0.0750	0.0100	0.0125	0.0150
cell size	0.025	55.2	28.6	27.1	14.4	13.8
	0.050	6.9	3.5	3.3	1.8	1.7
	0.750	2.3	1.2	1.1	0.6	0.5
	0.100	0.8	0.4	0.4	0.2	0.2

Table 6.2: Memory requirements of visibility caching (in megabytes) for the first scene with respect to different 3D grid cell sizes and minimum world sampling interval values

		minimum world sampling interval				
		0.0050	0.0750	0.0100	0.0125	0.0150
cell size	0.025	47.985	28.055	22.785	14.886	12.485
	0.050	6.088	3.508	2.922	1.886	1.593
	0.750	2.026	1.172	0.981	0.646	0.547
	0.100	0.739	0.427	0.359	0.235	0.199

Table 6.3: Visibility caching preprocessing times (in seconds) for the second scene with respect to different 3D grid cell sizes and minimum world sampling interval values

		minimum world sampling interval				
		0.0050	0.0750	0.0100	0.0125	0.0150
cell size	0.025	40.8	23.3	19.4	13.0	10.6
	0.050	5.1	2.9	2.4	1.6	1.3
	0.750	1.7	0.9	0.8	0.5	0.4
	0.100	0.6	0.3	0.3	0.2	0.1

Table 6.4: Memory requirements of visibility caching (in megabytes) for the second scene with respect to different 3D grid cell sizes and minimum world sampling interval values

6.2.2 Tracking

Figure 6.3 depicts the camera position estimation errors across frames for the first scene. The errors are calculated with respect to the baseline measurement.

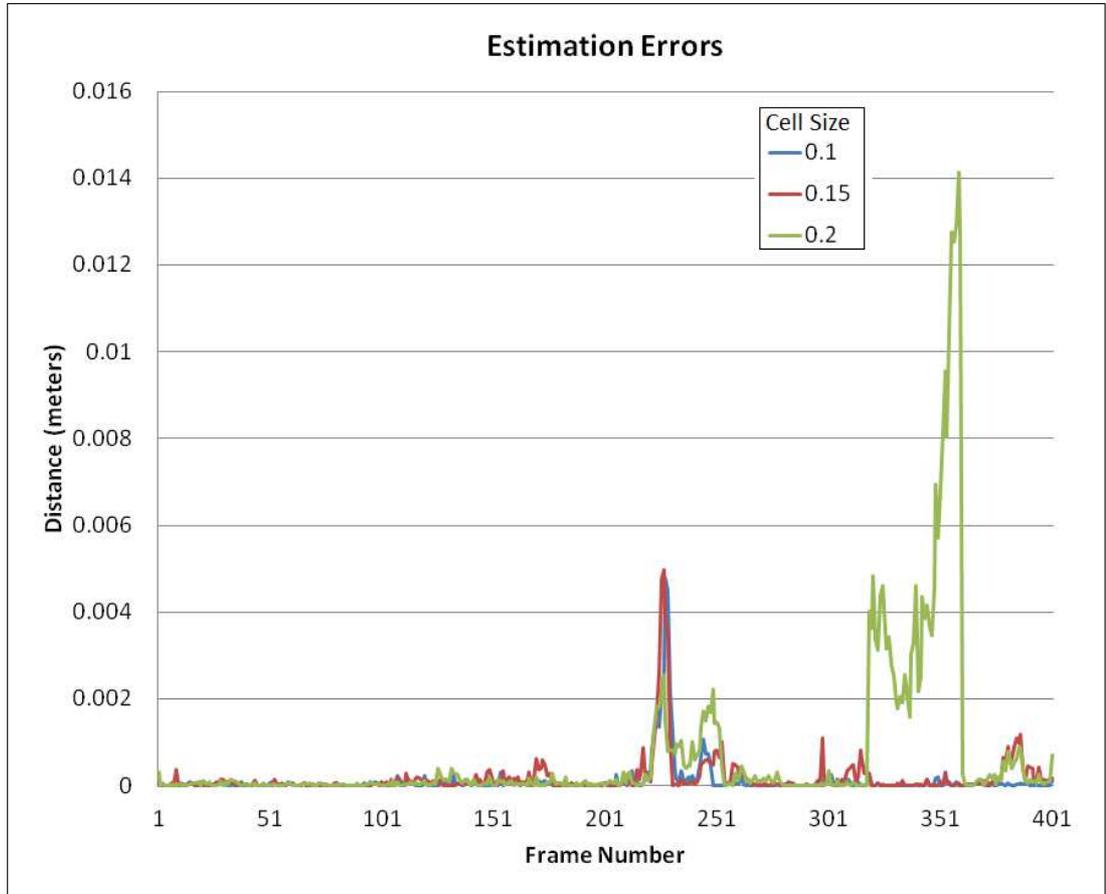


Figure 6.3: Estimation errors (in meters) of the first scene calculated with respect to the baseline for varying 3D grid cell sizes.

Table 6.5 gives the mean square error (MSE) of the estimated camera positions for the first scene with respect to the camera position estimates that are calculated using online visibility tests.

cell size	0.1	0.15	0.2
mean square error	1.08×10^{-4}	1.76824×10^{-4}	7.08205×10^{-4}

Table 6.5: MSE of trackers using visibility caching for the first scene with different cell sizes

Figure 6.4 depicts the camera position estimation errors across frames for the second scene. The errors are calculated with respect to the baseline measurement.

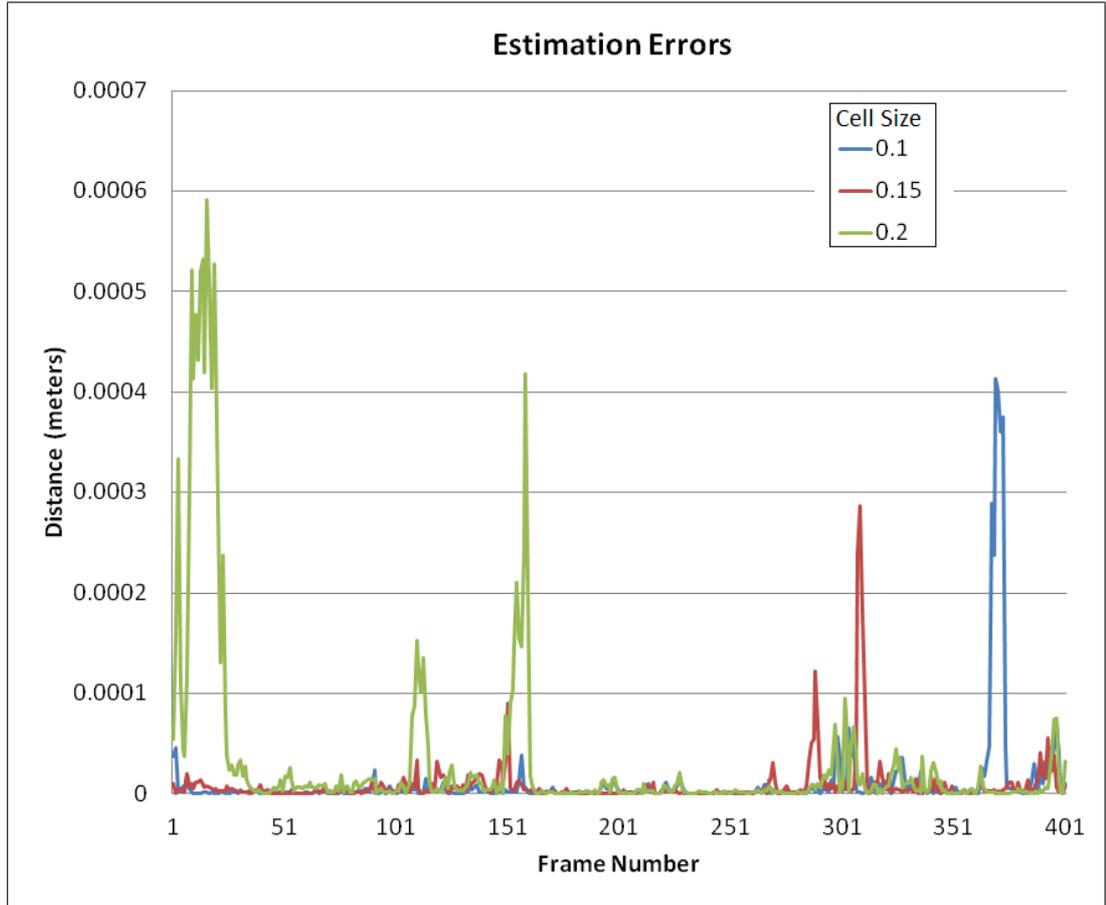


Figure 6.4: Estimation errors (in meters) of the second scene calculated with respect to the baseline for varying 3D grid cell sizes.

Table 6.6 gives the mean square error (MSE) of the estimated camera positions for the second scene with respect to the camera position estimates that are calculated using online visibility tests.

cell size	0.1	0.15	0.2
mean square error	1.04506×10^{-5}	8.05839×10^{-6}	3.27426×10^{-5}

Table 6.6: MSE of trackers using visibility caching for the first scene with different cell sizes

Table 6.7 gives a comparison of visibility testing methods in terms of the computation times per frame for the first scene. The camera parameters are

estimated in two iterations. The *minimum world sampling interval* is 0.005.

	minimum screen sampling interval				
	5	7.5	10	12.5	15
Online visibility	0.00347	0.00286	0.00229	0.00205	0.00175
Cached visibility	0.00304	0.00249	0.00204	0.00181	0.00155

Table 6.7: Average processing times (in seconds) per frame using different visibility testing methods.

Table 6.8 gives a comparison of visibility testing methods in terms of the computation times per frame for the second scene. The camera parameters are estimated in two iterations. The *minimum world sampling interval* is 0.005.

	minimum screen sampling interval				
	5	7.5	10	12.5	15
Online visibility	0.00626	0.00455	0.00364	0.00284	0.00240
Cached visibility	0.00530	0.00399	0.00312	0.00252	0.00209

Table 6.8: Average processing times (in seconds) per frame using different visibility testing methods.

As expected, the memory and preprocessing costs increase when the *minimum world sampling interval* and the cell size decrease due to the increasing number of control points and 3D grid cells. Experiments also show that the 3D grid with smaller cells provide better approximation for the visibility states of the control points.

Chapter 7

Conclusion

We propose methods to increase the performance and effectiveness of the model-based camera trackers. To this end, we use the persistent control points approach. We cache visibility states of the control points from different viewing positions using a uniform 3D grid structure and provide constant time access to the visibility state at run-time. Additionally, we use an accurate adaptive projection mechanism so that the persistent control points are more evenly distributed in the screen space.

In our experiments, we show that visibility caching can be used to provide constant time access to the visibility state of each control point. The tracking tests confirm that cached visibility tests perform similar to the online visibility calculations in terms of accuracy at the expense of a reasonable memory overhead, which is due to the 3D grid that is constructed. Conveniently, such 3D grid does not need to be calculated over and over again unless scene parameters that is related to the control point generation or grid construction are changed.

It is observed that the adaptive control point projection mechanism generates more evenly distributed control points in the screen space. This effect is especially noticeable when the marker edges are long and lie parallel to the camera viewing direction. Finally, we demonstrated a sample AR application where the user can interact with AR environment using the mouse and generate virtual entities.

These entities can collide with the marker geometry (and extra geometry that can be defined by the user) and occlusions are handled automatically.

Several extensions and improvements can be considered to improve the camera tracker. Firstly, different view-dependent information can be introduced to the tracking system. Example of such information would be control point reliability. Each control points have different reliability value depending on the camera position and its position with respect to the environment. Such information can be preprocessed for different camera configurations and stored in a 3D grid structure for constant time access. These reliability points then can be used to determine weights for the pose estimation process.

As a result of the 3D grid that is used, visibility states of the control points change only on cell borders. These rapid changes may cause jitter. In order to provide smoother transitions across the cell borders, visibility of a control point can be regarded as a continuous value which can be calculated by trilinear interpolation between eight neighboring cells.

Finally, adaptive control point projection mechanism can be improved. Currently, control points can appear/disappear rapidly when visible length of the marker edge changes. Such change can happen even with small camera movement and can result in jitter. To alleviate this issue, time based interpolation can be employed. With such approach, measurement weight of the control point should be increased gradually as it becomes active and decreased gradually as it becomes inactive.

Bibliography

- [1] C. Schoeneman, J. Dorsey, B. Smits, J. Arvo, and D. Greenberg, “Painting with Light,” in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*, (New York, NY, USA), pp. 143–146, ACM, 1993.
- [2] Y. Wang and D. Samaras, “Estimation of Multiple Directional Light Sources for Synthesis of Augmented Reality Images,” *Graphical Models*, vol. 65, pp. 185–205, 2003.
- [3] G. Xing, Y. Liu, X. Qin, and Q. Peng, “A Practical Approach for Real-time Illumination Estimation of Outdoor Videos,” *Computers & Graphics*, vol. 36, no. 7, pp. 857–865, 2012.
- [4] J. Lopez-Moreno, E. Garces, S. Hadap, E. Reinhard, and D. Gutierrez, “Multiple Light Source Estimation in a Single Image,” *Computer Graphics Forum*, vol. 32, no. 8, pp. 170–182, 2013.
- [5] Y. Park, V. Lepetit, and W. Woo, “Handling motion-blur in 3d tracking and rendering for augmented reality,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, pp. 1449–1459, Sept 2012.
- [6] P. Kan and H. Kaufmann, “High-quality Reflections, Refractions, and Caustics in Augmented Reality and Their Contribution to Visual Coherence,” in *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR '12)*, (Washington, DC, USA), pp. 99–108, IEEE Computer Society, 2012.

- [7] C. Harris and C. Stennet, “RAPiD – A Video-Rate Object Tracker,” in *Proceedings of the British Machine Vision Conference (BMVC '90)*, pp. 73–77, Sept. 1990.
- [8] D. B. Gennery, “Visual Tracking of Known Three-Dimensional Objects,” *International Journal of Computer Vision*, vol. 7, no. 3, pp. 243–270, 1992.
- [9] A. Comport, E. Marchand, M. Pressigout, and F. Chaumette, “Real-time Markerless Tracking for Augmented Reality: the Virtual Visual Servoing Framework,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 615–628, July 2006.
- [10] P. Bouthemy, “A maximum likelihood framework for determining moving edges,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 499–511, May 1989.
- [11] A. Petit, E. Marchand, and K. Kanani, “A Robust Model-based Tracker Combining Geometrical and Color Edge Information,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '13)*, pp. 3719–3724, Nov 2013.
- [12] H. Wuest, F. Vial, and D. Stricker, “Adaptive Line Tracking with Multiple Hypotheses for Augmented Reality,” in *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR '05)*, pp. 62–69, Oct 2005.
- [13] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.
- [14] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-Up Robust Features (SURF),” *Computer Vision and Image Understanding*, vol. 110, pp. 346–359, June 2008.
- [15] E. Rosten and T. Drummond, “Machine Learning for High-Speed Corner Detection,” in *Proceedings of the European Conference on Computer Vision (ECCV '06)*, vol. 1, pp. 430–443, May 2006.

- [16] E. Rosten, R. Porter, and T. Drummond, “FASTER and Better: A Machine Learning Approach to Corner Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, 2010.
- [17] M. Özuysal, P. Fua, and V. Lepetit, “Fast Keypoint Recognition in Ten Lines of Code,” in *Proceedings of the IEEE Conference on Computing Vision and Pattern Recognition (CVPR '07)*, pp. 1–8, 2007.
- [18] V. Lepetit and P. Fua, “Keypoint Recognition Using Randomized Trees,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 1465–1479, Sept. 2006.
- [19] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “BRIEF: Binary Robust Independent Elementary Features,” in *Proceedings of the 11th European Conference on Computer Vision (ECCV '10): Part IV*, (Berlin, Heidelberg), pp. 778–792, Springer-Verlag, 2010.
- [20] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, “Real-Time Detection and Tracking for Augmented Reality on Mobile Phones,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 3, pp. 355–368, 2010.
- [21] M. A. Fischler and R. C. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography,” *Communications of the ACM*, vol. 24, pp. 381–395, June 1981.
- [22] Qualcomm, Inc., “Vuforia Developer Portal.” <https://developer.vuforia.com/>. Accessed: 2014-06-03.
- [23] Metaio, GmbH, “Metaio Augmented Reality Products & Solutions.” <http://www.metaio.com/products/sdk/>. Accessed: 2014-06-03.
- [24] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, “KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera,” in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*, (New York, NY, USA), pp. 559–568, ACM, 2011.

- [25] A. Davison, “Real-time Simultaneous Localisation and Mapping with a Single Camera,” in *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV '03)*, vol. 2, pp. 1403–1410, Oct 2003.
- [26] *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, IEEE, 2010.
- [27] R. Salas-Moreno, R. Newcombe, H. Strasdat, P. Kelly, and A. Davison, “SLAM++: Simultaneous Localisation and Mapping at the Level of Objects,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '13)*, pp. 1352–1359, June 2013.
- [28] E. Rosten and T. Drummond, “Fusing Points and Lines for High Performance Tracking,” in *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV '05)*, vol. 2, pp. 1508–1515 Vol. 2, Oct 2005.
- [29] A. State, G. Hirota, D. T. Chen, W. F. Garrett, and M. A. Livingston, “Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking,” in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*, (New York, NY, USA), pp. 429–438, ACM, 1996.
- [30] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Transactions of the ASME—Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
- [31] G. Reitmayr and T. Drummond, “Going Out: Robust Model-based Tracking for Outdoor Augmented Reality,” in *Proceedings of the IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR '06)*, pp. 109–118, Oct 2006.
- [32] G. Klein and T. Drummond, “Tightly Integrated Sensor Fusion for Robust Visual Tracking,” in *Proceedings of the British Machine Vision Conference (BMVC 02)*, pp. 787–796, 2002.

- [33] G. Klein, *Visual Tracking for Augmented Reality*. PhD thesis, Department of Engineering, University of Cambridge, 2006.
- [34] T. Drummond and R. Cipolla, “Real-time Visual Tracking of Complex Structures,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 932–946, 2002.
- [35] Unity Technologies, “Unity - Game Engine.” <http://unity3d.com/>. Accessed: 2014-06-08.
- [36] ALGLIB Project, “ALGLIB - Cross-platform Numerical Analysis and Data Processing Library.” <http://www.alglib.net/>. Accessed: 2014-06-08.

Appendix A

Software Packages Used in the Implementation

A.1 Unity Game Engine

Unity [35] is a commercially available game engine. It simplifies game and application development process by streamlining many concepts (Asset import, Physics, Input management, and so on) and providing built-in solutions. It supports popular platforms including Windows, Linux, iOS as well as gaming consoles. Because of its vast mobile device support and ease of use, Augmented Reality framework that is presented in this thesis is developed in Unity.

Unity provides component-based architecture for game development. Entities in the game world is simple container objects, called *GameObjects*. *GameObjects* gain functionality through reusable *Components*. *Components* have well-defined responsibilities in subsystems like physics, rendering and audio. *Transform*, *Collider*, and *Rigidbody* are examples of such components. Users can develop their own components, which are called *Monobeaviours*. *Monebehaviors* can be created using C# or JavaScript.

3D models that are prepared in 3D modeling packages can easily be exported to Unity. The models used for tracking are prepared in popular 3D modeling software and exported to the Unity. Unity provides the *Mesh* structure to access and modify the structure of the imported models. The edge extraction process is performed using the *Mesh* structure.

For visual tracking, the video feed is processed at each frame. Unity provides *WebCamTexture* for this purpose. *WebCamTexture* is a dynamic texture that is updated when a new image is available through the RGB camera. *WebCamTexture* provides interface to access individual pixels and thus allows processing of the video feed.

Unity has a built-in physics engine. It is capable of processing complex collisions, providing realistic simulations of clothes, hinges, ragdolls, and so on. Unity Physics Engine also exposes a set of methods to perform efficient raycasting operations. Visibility tests are performed using raycasting methods provided by Unity Physics Engine. Physics interaction is realized using *rigidbodies* and *colliders* that are available in the Unity Game Engine.

Unity provides a convenient *Prefab* system. A *Prefab* is an asset that stores a *GameObject* along with its components and attributes. Such prefabs can be instantiated to the scene at any time and can be modified simultaneously by editing the prefabs instance. The AR framework developed provides a simple drop-in prefab for the tracking camera for the ease of use.

The software produced in this thesis research is provided as a simple drop-in Unity package that is easy to use.

A.2 ALGLIB

ALGLIB [36] is a cross-platform numerical analysis and data processing library. It provides many useful packages for data analysis, optimization, nonlinear systems, linear-squares fitting and other related areas. In this work, ALGLIB is used to solve the linear system of equations for camera parameter estimation.