Visualization of Viewing Fields of Surveillance Cameras by Using Mixed Reality

Dao Ngoc Thanh

(Master's Program in Intelligent Interaction Technologies)

Advised by Yuichi Ohta

Submitted to the Graduate School of
Systems and Information Engineering
in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering
at the
University of Tsukuba

March 2009

Abstract

This research lies in the domain of Outdoor Mixed Reality. It aims to visualize the viewing fields of outdoor surveillance cameras on the screen of mobile devices. In an outdoor environment where cameras on mobile devices can be used, it is required that users can easily understand the visualized viewing fields of surrounding surveillance cameras. Moreover, the visualization system must provide realtime video with no human-sensible delay.

In this research, we study five visualization methods and propose a prototype system based on server-side PTAM. The prototype requires no GPS or gyrocompass devices. We have implemented the prototype and conducted some experiments in our campus.

Contents

1 Introduction **1**
- 1.1 Introduction . . . 1
- 1.2 Use Cases . . . 4
- 1.3 Related Works . . . 4

2 Visualization Methods **7**
- 2.1 Visualization Elements . . . 7
- 2.2 Visualization Methods . . . 8
  - 2.2.1 Volume Method . . . 9
  - 2.2.2 Shadow Method . . . 9
  - 2.2.3 Contour Method . . . 11
  - 2.2.4 Arrow Method . . . 11
  - 2.2.5 Animation Method . . . 11
- 2.3 Scene Model . . . 13
- 2.4 Viewing Modes . . . 14

3 Server-side PTAM Architecture **16**
- 3.1 System Architecture . . . 16
- 3.2 Server-side PTAM . . . 17
- 3.3 PTAM Map Initializing and Scene Model Pre-registering . . . 19
- 3.4 Fusion of GPS and Gyrocompass Devices . . . 21
- 3.5 Software Development Environment . . . 25

4 Evaluation of Visualization Methods **27**
- 4.1 Evaluation Condition . . . 27
- 4.2 Result and Discussion . . . 29

5 Conclusion **30**

Acknowledgements **32**

A Sony VAIO VGN-UX90PS **33**

B Apple iPhone 3G **34**

C Apple MacBook Pro **36**

D Garmin GPSmap 60CSx **38**

E InterSense InertiaCube3 **41**

Bibliography **42**

List of Figures

1.1 A Head Mounted Dislay 2
1.2 Sony VAIO VGN-UX90PS 3
1.3 Prototype architecture 5
1.4 Safe path to school 6

2.1 Viewing volume of camera 8
2.2 Volume method 9
2.3 Shadow method 10
2.4 Contour method 11
2.5 Arrow method 12
2.6 Animation method 13
2.7 Scene model 14
2.8 The three degree-of-freedom parameters of a gyrocompass 15

3.1 RPC vs. Mailbox 17
3.2 A PTAM map 20
3.3 Feeding frame to PTAM to initialize map 22
3.4 Fusion of GPS and gyrocompass devices 23
3.5 GPS error 24

4.1 Evaluation experiment screenshot 28

B.1 Apple iPhone 3G 34

C.1 Apple MacBook Pro 36

D.1 Garmin GPSmap 60CSx 38

E.1 InterSense InertiaCube3 41

Chapter 1

Introduction

1.1 Introduction

Surveillance cameras have become popular and placed almost everywhere, e.g. train stations, airports, banks, shops, streets. The balance between security and privacy is an active subject for political/social debates. It is hard to keep a balance between security and privacy. A person may feel protected when he knows himself and his surroundings are being observed, but at the same time he may feel uncomfortable.

The information that people probably wants to know most about surveillance cameras is probably their viewing fields, e.g. where the cameras are, whether he is inside the area being observed. On the other hand, for indoor environment, because the space is usually small, people may easily locate the position and estimate the viewing fields of the cameras. For outdoor environment, it is rather hard to find surveillance camera systems hence they need special support to visually know the viewing fields in real scene. Fortunately, there is a technology which can be applied for this demand: Mixed Reality (MR). MR is encompassing both Augmented Reality and Augmented Virtuality, merging the real world in which we are living and the virtual world created by computers. MR can produce new environments where real and virtual entities can co-exist and interact in realtime [1].

MR applications are traditionally equipped with Head Mounted Displays (HMD) as shown in figure 1.1. However, HMDs are usually bulky and inconvenient for outdoor applications, where high-level mobility is crucial. In recent years, mobile devices have become popular, devices like cell phones, Personal Digital Assistants (PDA) can be seen everywhere. Their prices have come down and they could reach the hands of ordinary people even in developing countries like Viet Nam. High-end mobile devices, such as Apple iPhone and Google T-Mobile G1, usually have high specifications. As for their important advantages for outdoor MR applications, they are almost always equipped with built-in cameras, Global Positioning System (GPS), and accelerometer devices. These auxiliary devices usually meet the requirements on building up outdoor MR applications because they can provide video and information to help estimating position and orientation of the mobile devices [2] [3].

GPS and accelerometer devices are not sufficient for achieving highly precise MR which we need to visualize viewing field of surveillance cameras. Hence we need to introduce an

Figure 1.1: A Head Mounted Dislay

image-based camera registration method which could reach to the accuracy we need at frame rate. However, image processing applications usually require large memory and powerful computing capability. Even current high-end cell phones and PDAs may not run MR applications properly without special program optimizations. For computation-intensive outdoor MR applications, more powerful Ultra-Mobile Portable Computers (UMPC) like the one in figure 1.2 may be used. UMPCs equipped with built-in camera, GPS, and gyrocompass devices have been found practical and are the topic of many researches [2] [3] [4].

Figure 1.2: Sony VAIO VGN-UX90PS with a camera at the front and at the back

This research aims to visualize the viewing fields of outdoor surveillance cameras on the screen of mobile devices. In this research, we propose:

- Five methods to visualize viewing fields of surveillance cameras for outdoor MR.
- A new architecture of online camera calibration: server-side Parallel Tracking and Mapping (PTAM) [5] system. Our prototype system uses a mobile device which can be either a Sony VAIO VGN-UX90PS or an Apple iPhone and can provide realtime video. Because the mobile devices do not have sufficient computation facility to execute the heavy PTAM processing, we connect it wirelessly to a PTAM server which is an Apple MacBook Pro. This results in a system providing realtime MR video with no requirement for any GPS or gyrocompass devices.

Structure of this thesis:

- Rest of chapter 1 introduces the requirement on visualizing viewing fields of surveillance cameras, and some related technologies and researches.
- Chapter 2 poses basic elements of visualization methods, then studies five visualization methods to show viewing fields of surveillance cameras.

- Chapter 3 proposes online camera registration method based on server-side PTAM which implements and is integrated to the visualization methods in chapter 2.
- Chapter 4 conducts an experiment to evaluate the five methods using the prototype, then discusses the result.
- Chapter 5 summarizes this research and proposes future directions.

1.2 Use Cases

When a user wants to see the visualized viewing fields of surrounding surveillance cameras on his own mobile device screen, the typical usage scenario is (figure 1.3):

1. The user points the mobile camera in a certain direction.
2. Through wireless network, the mobile device continuously sends video frames captured by the camera of the scene to a PTAM server.
3. The PTAM server estimates the position and orientation of the mobile camera by analyzing the received video frames, and send them to the mobile device.
4. Based on the position and orientation, the mobile device will correctly render visual aids that are instances of viewing fields of the surveillance cameras, and shows the MR video images on its screen.

When the user changes the position or orientation of the mobile camera, the MR video displayed on the screen will simultaneously change accordingly to the camera movement.

The above use case gives an example of the application of visualizing viewing fields of surveillance cameras and a rough idea of how the prototype works. There may be other applications. For example, we can build a system which visualizes "safe paths" or "safe areas" for children (figure 1.4). Children can use their cell phones to see if their current location is being well observed when they walk to school or back home.

1.3 Related Works

For outdoor MR, bulky devices that reduce the mobility of users are not preferable. For example: heavy computers, devices wired to a fixed location, devices with big auxiliaries, big HMDs, HMDs that prevent users from seeing the outdoor environment while walking etc. Devices for outdoor MR should be small, lightweight, and can connect wirelessly to other devices if they need to.

MR systems usually need to know the position and orientation of the devices in order to merge images of the virtual world and the images of the real world at high accuracy. For indoor environment, marker-based solutions are known to work very well [6]. However, outdoor environment is usually large and it is impossible to put markers everywhere. There have been many researches that use UMPCs equipped with GPS and/or gyrocompass

Figure 1.3: Prototype architecture. The mobile device can be either a VAIO (appendix A) or an iPhone (appendix B).

Figure 1.4: Safe path to school

devices to realize markless camera registration [2] [3] [4]. Hardware devices have been becoming smaller and more powerful according to Moore's law. Today's high-end cell phones, like Apple iPhone and Google T-Mobile G1, usually have built-in GPS and accelerometer devices. Because of their small size, high specifications, and competitive prices, such devices have found their use in outdoor MR applications and researches, like Sekai camera [7] and Enkin [8].

However, normal GPS devices have error of about 5–10 m, and gyrocompass devices suffer from drift error. There have been many researches that are based on the images taken by the camera on the mobile device to deal with this problem. We can follow the approach that uses (1) gravity accelerometer to help eliminating the drift error, (2) model-based tracking on edges and (3) texture tracker to help produce accurate localization result [4]. There is also an approach that completely does not require GPS or gyrocompass devices, but uses the images taken by the camera and a landmark database of natural feature points built before-hand [9].

When MR programs need more memory or CPU power than what the mobile devices can provide, we have a backup solution in which the mobile devices connect wirelessly to remote non-mobile computers to ask for help. It is possible to send realtime video images wirelessly to remote computers for further processing because 54 Mbps wireless network has become popular today. This may be a good approach because Moore's law has slowed down in recent years, thus we cannot expect the computation power of mobile devices to go much higher anytime soon while at the same time keeping their small size and mobility.

Chapter 2

Visualization Methods

The purpose of our research is to visualize the subspace of the real world in which any objects could be inserted by surveillance cameras. We take up ordinary perspective cameras for surveillance cameras in this research. We propose to insert "visual aids" into real images captured by the mobile camera. We take up five kinds of visual aids in this paper. First we discuss the elements of visualization of viewing fields in section 2.1, then the detailed definitions of the visual aids are given in section 2.2.

2.1 Visualization Elements

Viewing volume of a projective camera is defined by an infinite half cone, one apex of which corresponds to the focal point of the camera. Because real computers cannot have unlimited resources, in practical computer 3D technology we usually limit the bottom of the cone by a far clip plane or by boundary surfaces which may cut the viewing volume, then limit the top of the cone even more by a near clip plane. This results in a frustum (figure 2.1). In this research we abstract the viewing volume for visualization by the term "viewing field".

There may be many methods for visualizing the viewing fields, not limited to the ones discussed in the next section. But for a method to find practical use in outdoor MR, it should be able to be implemented (see section 1.2 to have an illustration) so that it meets the following basic qualitative requirements, expressed in the "it should" tongue of most Domain Specific Languages of the Behavior Driven Development methodology in software engineering:

- It should be easy to understand when users see the visualized viewing fields of surveillance cameras.
- It should work in realtime. When users change the position or orientation of the mobile device, the MR video displayed on the screen of the mobile device should simultaneously change accordingly to the movement of the camera with no or little delay.
- It should have good accuracy. In order to correctly overlay visual aids onto the original

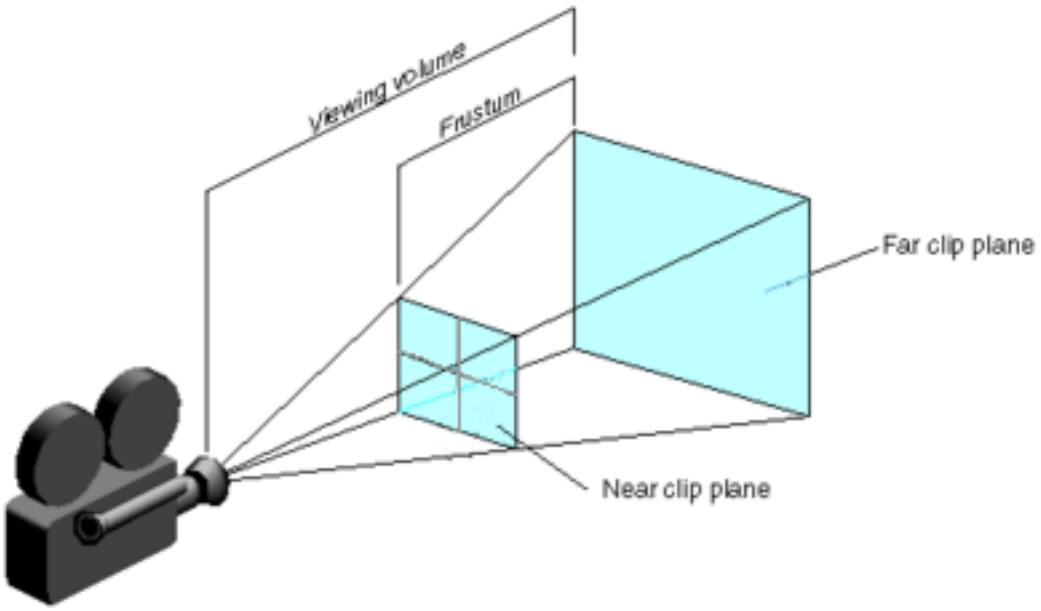

Figure 2.1: Viewing volume of camera

video frames taken at the user side, the position and orientation of the mobile camera must be estimated within a small registration error.

- It should be robust to disturbance in outdoor environment. Some of the disturbance that may arise in real situations are passers in front of the mobile camera and GPS signal noise/weakness. For example, in outdoor environment, there may be bicycle riders passing by and they may temporarily occlude the mobile camera. Another example is that if the system uses GPS device, the GPS signal strength may change a lot when users walk from an open space to a space shadowed by trees or buildings.

We must take the above requirements into account when implementing the prototype in chapter 3.

2.2 Visualization Methods

In all the five visualization methods, in order to emphasize the boundary of the viewing fields of the surveillance camera, we draw four straight lines to visualize the edges of the cone in figure 2.1. The way we draw the five sides and the inside space of the cone gives us various visualization methods. A good side effect when drawing the four straight lines is that the position of the camera can be inferred because it is the only intersection point of the lines.

2.2.1 Volume Method

The most direct method to visualize the viewing field is simply visualizing the viewing volume (figure 2.2) in alpha blending (see-through) fashion. However, the volume usually occupies parts of the mobile device's screen, especially if the user is inside the viewing volume, hence it may be hard to be understood.

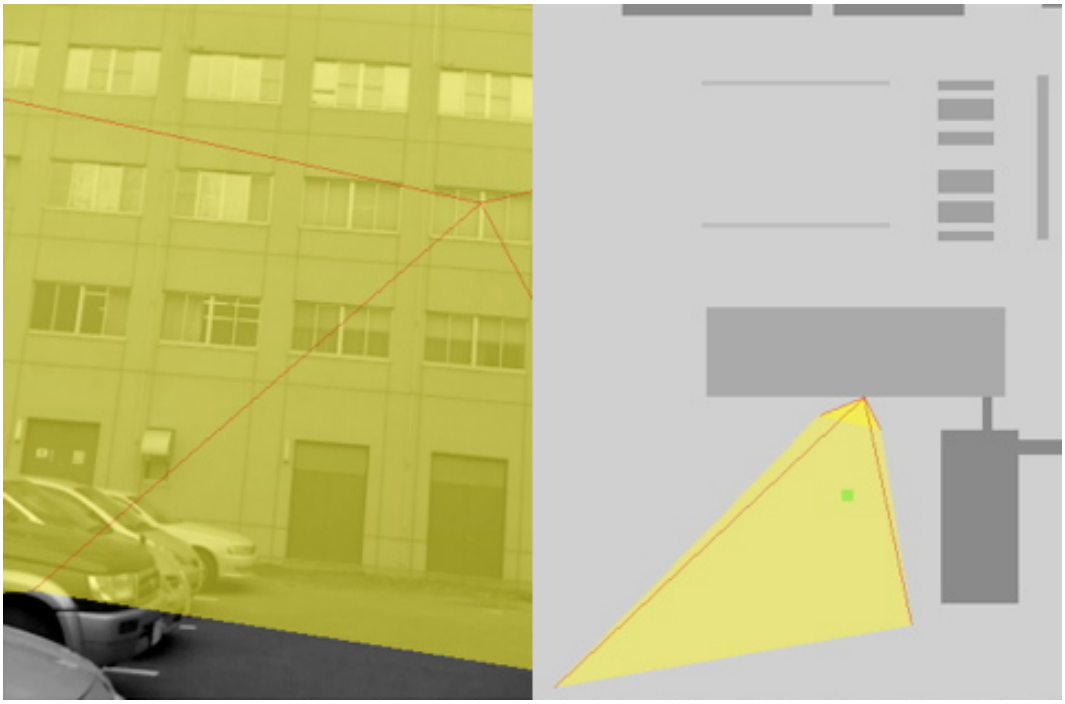

Figure 2.2: The viewing field of a surveillance camera on the building is being visualized using the volume method. The user's position is indicated by the green dot.

2.2.2 Shadow Method

The fact that the frame captured by a surveillance camera can be thought to be a perspective projection of the volume into the near plane of the frustum gives us another method, shadow method. The shadow casts the virtual shadow created by the light positioned at the focal-point of the camera and the near plane of the frustum (figure 2.3). This method may give more understandable visualization.

There are various ways to render the shadow, the two popular ones are shadow mapping [10] [11] and volume shadow [12].

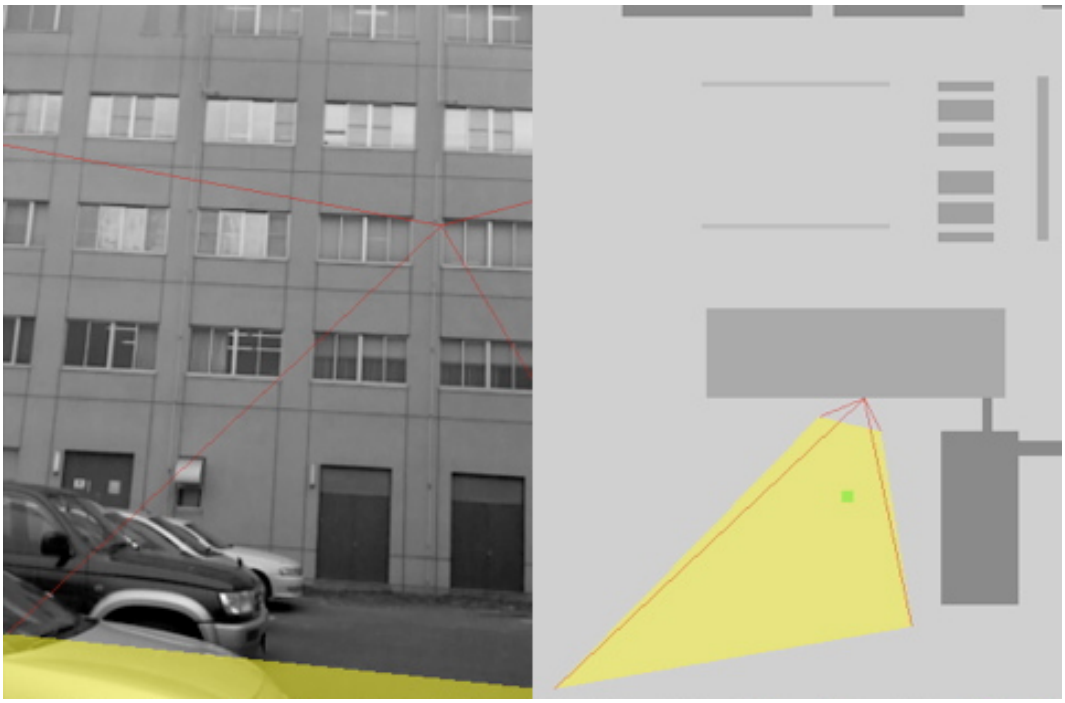

Figure 2.3: The viewing field of a surveillance camera on the building is being visualized using the shadow method. The user's position is indicated by the green dot.

2.2.3 Contour Method

An alternative method is to only visualize the contours of the shadow (figure 2.4) to reduce the occlusion of the scene by the visual aid. Rendering only the contour is much lighter than rendering the whole shadow, thus this method is supposed to be much faster than the shadow method.

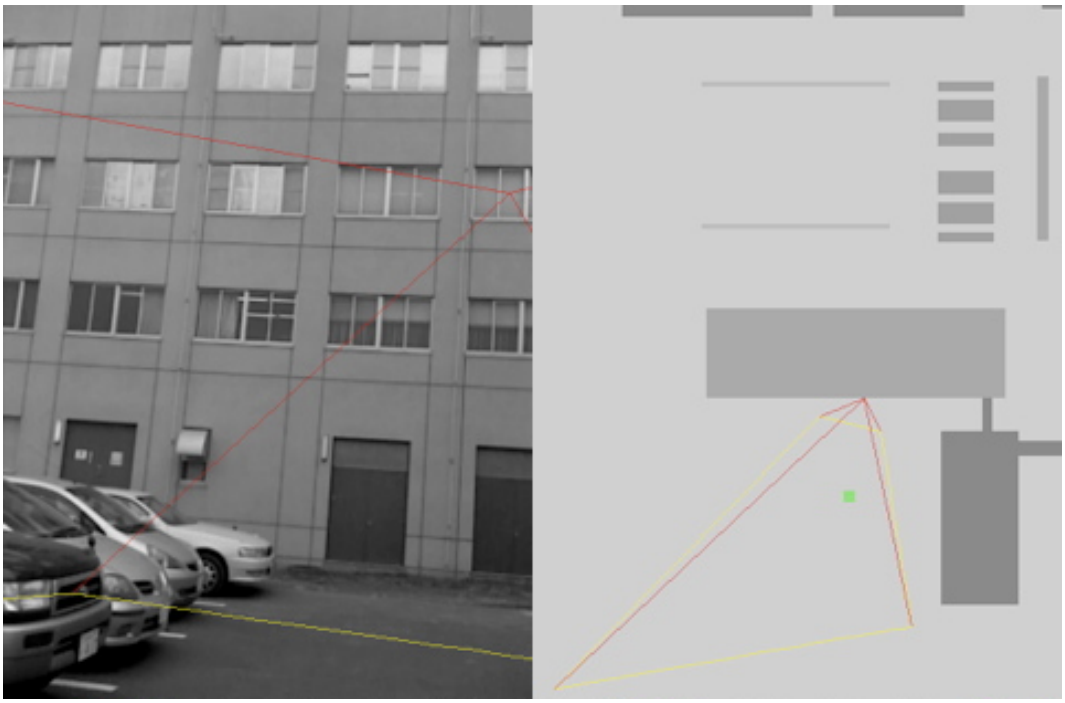

Figure 2.4: The viewing field of a surveillance camera on the building is being visualized using the contour method. The user's position is indicated by the green dot.

2.2.4 Arrow Method

In some cases, a person may want to know the relative distances from points inside the shadow (section 2.2.2) to the camera, in addition to its viewing field itself. To visualize this information, we propose vector method that puts arrows on the shadow area as in figure 2.5. In the figure, all arrows are pointing the camera, and their lengths are reverse proportional to the distance from the root of the arrows to the camera.

2.2.5 Animation Method

This method uses a moving mesh (figure 2.6) coming from the surveillance camera to the clipped end of the viewing field (or in the reverse direction) to visualize the viewing fields. This method has many advantages:

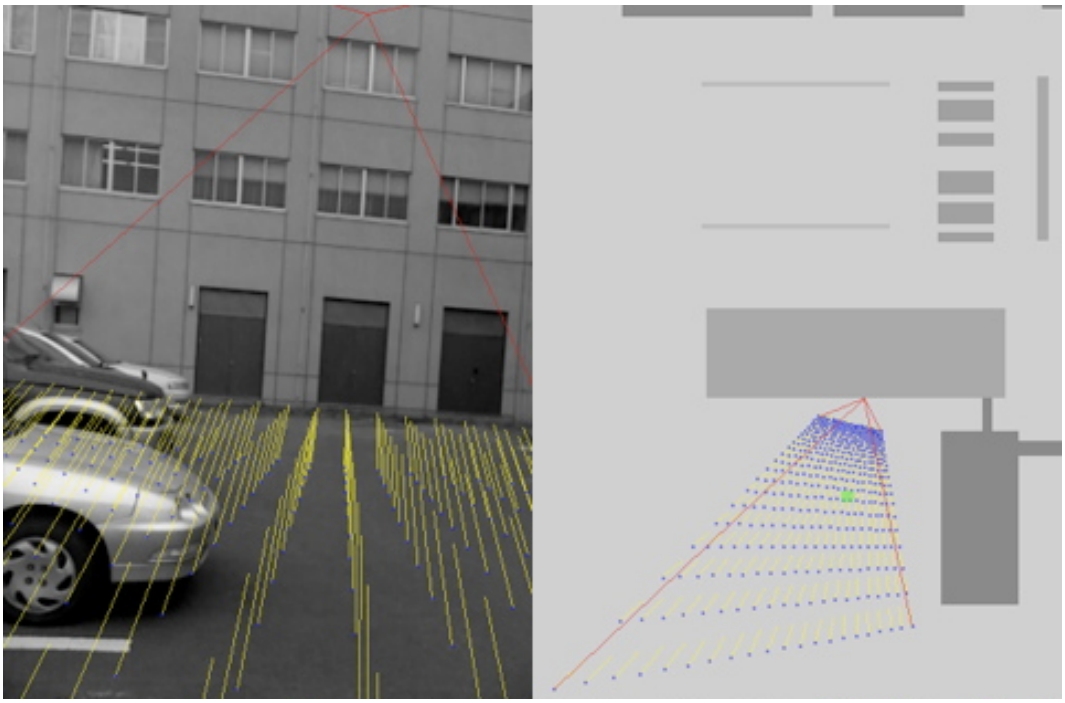

Figure 2.5: The viewing field of a surveillance camera on a building is being visualized using the arrow method. The arrows' roots are indicated by blue dots, the arrows' heads are not drawn for clarity. The user's position is indicated by the green dot.

- The moving mesh starts from the surveillance camera position (or moves towards the camera). As a result the user can easily know the camera position.
- Even when the mobile device is fixed at a certain position and pose, the visualization effect is still helpful to understand the viewing field because of its animation.
- The moving mesh does not occlude the scene. As a result the user can easily see both the scene and the visualized viewing field at the same time.

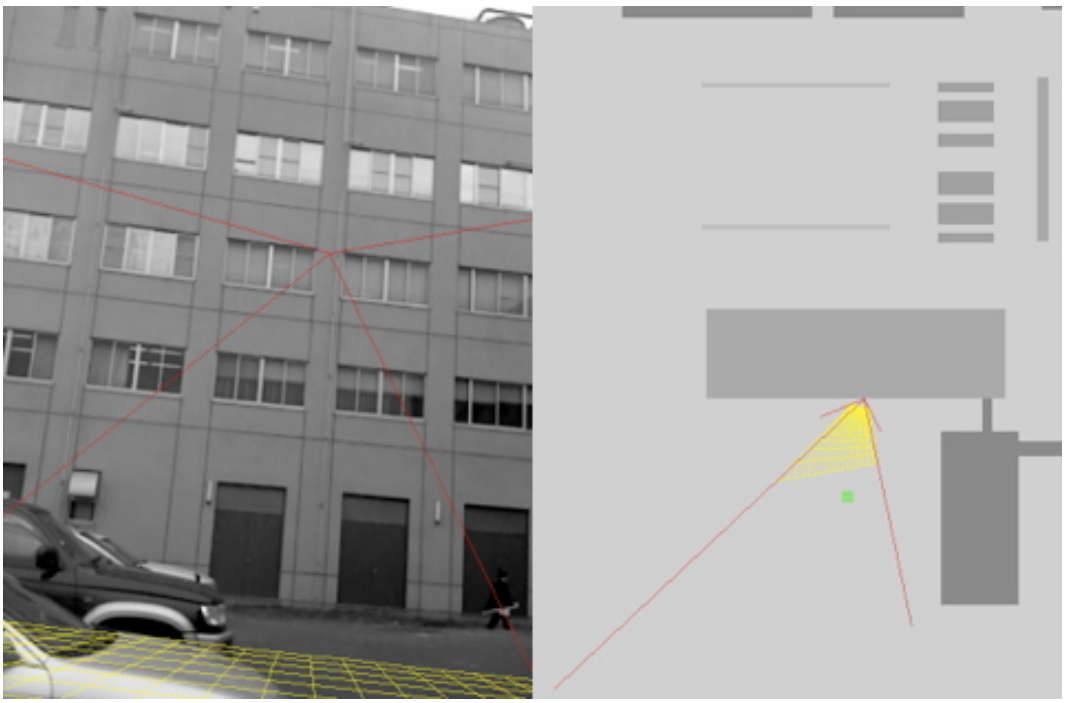

Figure 2.6: The viewing field of a surveillance camera on the building is being visualized using the animation method. The mesh is moving from the camera down to the ground. The user's position is indicated by the green dot.

To make the visual aids look more "real", the moving mesh stops when it touch the surface of the building walls or the ground. In order to do this, we need the model of the scene as described in the next section.

2.3 Scene Model

In order to define the viewing field for the visualization methods in section 2.2, we need the model of the scene. The model contains the 3D structure of the buildings (figure 2.7) and the positions and orientations of the surveillance cameras. The model is rather simple and only contains flat planes. We do not need to have complexed ones, such as those

that contain texture, because we only need a coarse-grained approximation of the viewing field. Moreover, because surveillance cameras are usually stationary, their positions and orientations are usually static.

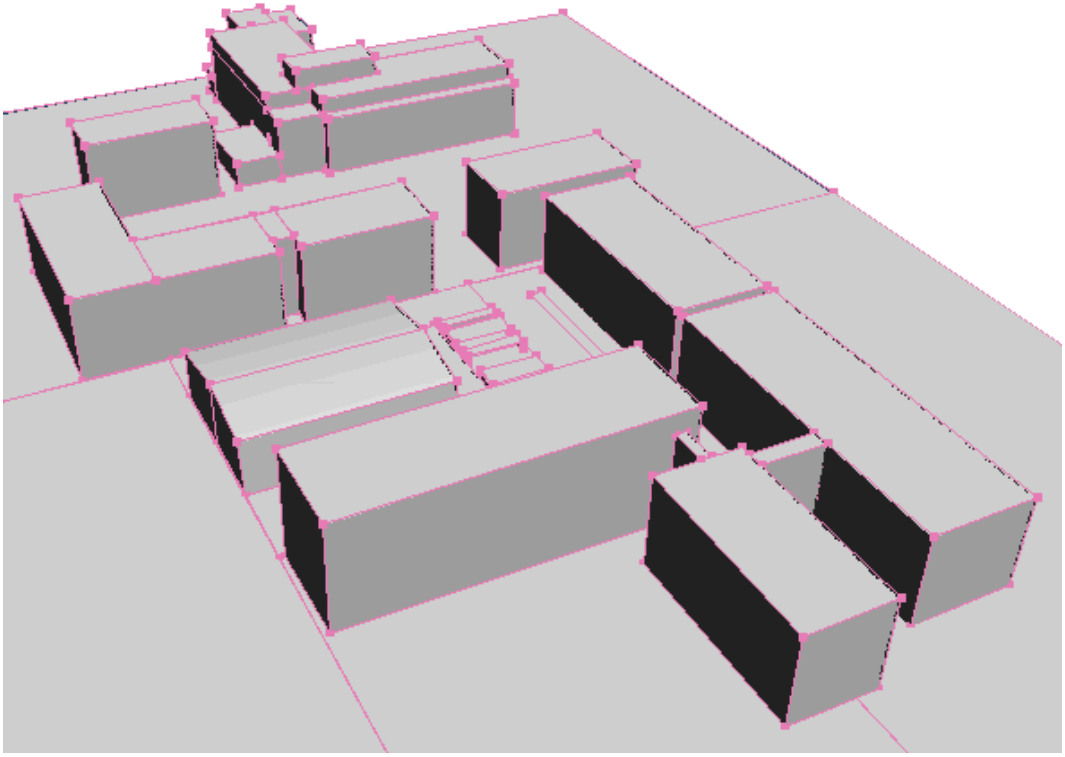

Figure 2.7: 3D structure of the buildings in the scene model

Using the scene model, we can also render visual aids in various modes as described in the next section.

2.4 Viewing Modes

In general it is easier to understand a thing when it is put in a context. In our case, in real scene when users see our visual aids that visualize the viewing fields of the cameras, they also see the big picture of the area around them. Consequently, if we let the users see the map of the area where they are standing, the map will help them understand the visualized viewing fields of the surrounding surveillance cameras better.

Because we have the 3D scene model as described in the previous section, we propose three viewing modes:

- Real mode: In this mode frames taken by the mobile camera are overlaid with any of the visual aids.

Table 2.1: Using gyrocompass for navigation

Degree-of-freedom parameter	Usage
Yaw	360°rotation
Pitch	Forward-backward movement
Roll	In/out zooming

- Map mode: Map of the area around the current position is displayed to let users overview the scene.
- First-person shooter (FPS) mode: Users can navigate the 3D world as in FPS games. Using this mode, users can virtually walk to other positions of the area to see other viewing fields of surveillance cameras there. In this mode only the scene model is rendered, video frames taken by the mobile camera are not used.

In section 2.2, figures on the left and on the right show the screen in real mode and map mode respectively.

In order to navigate the scene in map and FPS mode, we can use the keypads on the mobile devices. However, keypads on mobile devices are usually hard to use, especially when only one hand is used to both hold the device and press the keys. When a mobile device does not have a touch screen, but it is equipped with an accelerometer (most cell phones today have one built-in) or gyrocompass, these auxiliary devices can be used for navigating around. This usage has been proved by existing applications of some game gadgets and iPhone that have those sensors. For example, in map mode the three degree-of-freedom parameters of the gyrocompass can be used as in table 2.1.

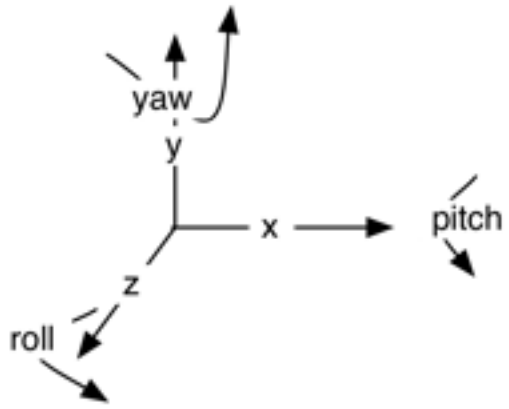

Figure 2.8: The three degree-of-freedom parameters of a gyrocompass

For devices that have multi-touch screen like the iPhone, the touch screen can be used to move, rotate, zoom in and out the map in the map viewing mode very easily.

Chapter 3

Server-side PTAM Architecture

In this chapter, we implement a prototype system based on the reference implementation of PTAM [5]. The prototype integrates the five visualization methods in section 2.2. It provides realtime MR video with no requirement for any GPS or gyrocompass devices.

3.1 System Architecture

The prototype can use either a Sony VAIO (appendix A) or an Apple iPhone (appendix B) as the mobile device. We base our program on the PTAM reference implementation source code [13] to get the position and orientation of the camera attached to the mobile device. However, this implementation can run on neither mobile devices because it requires a dualcore CPU to be able to run smoothly. In order to add computation power to the prototype system, we connect a MacBook Pro (appendix C) as a PTAM server to the mobile device via wireless network. The PTAM part of the program is set aside on the PTAM server (figure 1.3).

For the sake of quick setup time, the mobile device and the PTAM server are connected directly using ad hoc mode, which means an additional wireless hub is not needed. This setup has another advantage that the data transmission is very quick: the "ping" time is only about 1 ms. If a hub is used, the "ping" time may sometimes go up to 100 ms.

On theory the network interfaces on both VAIO and MacBook Pro are able to work at 54 Mbps, but for some reason of Microsoft Windows SP2 they can only work at 11 Mbps (table 3.1). This is an unexpected problem which lowers the speed, and consequently the accuracy and stability of the prototype system. They can be connected via LAN cable for faster 100 Mbps speed, but since the VAIO will be wired to another bulky device, wiring

Table 3.1: System configuration

Configuration	VAIO–MacBook Pro	iPhone–MacBook Pro
Operating system	Windows XP SP2	iPhone OS 2.2, Mac OS X 10.5.6 (Leopard)
Wireless network speed	11 Mbps	54 Mbps

for network is not desirable because the mobility of the VAIO is totally lost.

3.2 Server-side PTAM

We modify the original PTAM source code so that the "video source" is remotely located on the mobile device, not on the server where the PTAM process actually runs. We learn a little from Erlang language [14] of how to write concurrent, high-performance systems. Right after the Transmission Control Protocol (TCP) connection between the mobile device and the PTAM server is established:

- The mobile device continuously feeds video frames captured from its camera to the PTAM server.
- The PTAM server continuously feeds position and orientation of the mobile camera to the mobile device.

As the above explanation suggests, we use "push" style data passing instead of "pull", one side actively sends data to the other side without having to wait for the request for the data from the other side. In other words we use the mailbox design pattern instead of the Remote Procedure Call (RPC) one, which is slower and requires synchronizing the call and the result (figure 3.1).

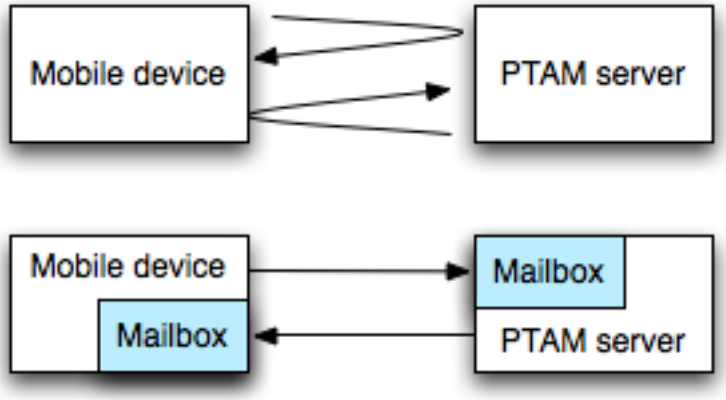

Figure 3.1: RPC (above) vs. Mailbox (below)

There are two essential threads in the PTAM process: tracking and mapping threads. The tracking thread tracks feature points in the frames received from the mobile camera, and the mapping thread builds a 3D map of these points. Because the tracking thread only need grayscale frames, instead of sending all the three RGB color channels of each frame, the mobile device only needs to send the brightness Y = 0.3*R + 0.59*G + 0.11*B. There are several optimizations to further improve the frame rate sent to the PTAM server:

Table 3.2: Frame rates of various conditions between the VAIO and the MacBook Pro (see table 3.1)

Size (only G color channel)	Compressed	With PTAM processing	Frame-per-second
320 x 240	NO	NO	8.01
320 x 240	YES	NO	10.67
640 x 480	NO	NO	1.97
640 x 480	YES	NO	3.03
320 x 240	NO	YES	6.57
320 x 240	YES	YES	6.23
640 x 480	NO	YES	1.84
640 x 480	YES	YES	1.69

- Send G color channel instead of Y, because G affects Y the most as in the above equation. This does not affect PTAM. This optimization enormously improves the frame rate because we can omit the pixel-wise calculation of the brightness estimation.
- Losslessly compress G color channel using the standard library "zlib". The compression rate is about 50%. But we should use compression with care because it may take more time to compress and uncompress than to send uncompressed images.

However, as table 3.2 suggests, we should use 320 x 240 frame size with uncompressed G color channel for the VAIO–Macbook Pro configuration. Bigger frame size allows the PTAM server to find more feature points, but the following drawbacks pop up:

- Data for each frame gets bigger. This increases data transmission time for each frame.
- The PTAM server must consume computation power on the bigger frame. This decreases the result frame rate.

When the image size is enlarged from 320 x 240 to 640 x 480, the frame rate drops about 4 times, and the prototype no longer works smoothly as required in section 2.1.

Color channels of the camera frames are not saved separately, but in the format BGR for each pixel. Hence to extract the G color channel from each frame we cannot do it in one pass by "memcpy"ing the whole channel. We must do a pixel-wise loop through all the pixels of each frame. This processing does consume computation power of the mobile device. Consequently, it may be faster to send all the BGR values to the PTAM server in one pass, rather than extracting the G color channel. Because the computation power of the mobile device is far weaker than that of the server, in order to improve the result frame rate we should make the following decisions: (1) which frame size should we take and (2) when should we extract the G color channel and/or compress the data?

The answer is that we should not make any optimization as long as the wireless network is not saturated by the data for each frame. The tracking thread of the PTAM server can work well with about 7 frame-per-second rate, hence ideally in case of 11 Mbps network,

the data size threshold for each frame is 11 x 1024 x 1024 / 8 / 7 = 205970 Bytes. We should not apply any optimization if the data for each frame is less than this threshold:

- 320 x 240 x 1 color channel = 76800 (< 205970)
- 324 x 240 x 3 color channels = 230400 (> 205970)
- 640 x 480 x 1 color channel = 307200 (> 205970)

Color channel extraction is lighter than data compression, hence we can conclude that for the VAIO (11 Mbps network) the frame rate is the best if we use 320 x 240 and G color channel extraction. For the iPhone (54 Mbps network), we can do similar calculation. However, due to its hardware limitation the frame is fixed at 304 x 400 x BGRA (4 channels). The calculation suggests that we should not apply any optimization.

On the mobile device, there are pre-registered positions and orientations of surveillance cameras as described in section 2.3. Having the position and orientation of itself tracked and fed by the PTAM server, the mobile device can overlay the visual aids on the screen precisely.

3.3 PTAM Map Initializing and Scene Model Pre-registering

PTAM continuously builds and maintains a 3D map by adding stable feature points taken from the frames and removing unstable ones (figure 3.2).

The problem is the 3D scene model (section 2.3) must be in match with this map. We assume that the scene is unchanged. Thus we have the following workflow:

1. Initialization and registration phase: Let the PTAM server build the map. After that manually translate and rotate the scene model to match the map (or vice versa), then save the translation and rotation parameters.
2. Usage phase: On the next program startup, as long as we can make sure that the map is not different from the one used to save the parameters, we can use the parameters to translate and rotate the scene model to match the map.

The reference implementation of PTAM initializes the map on every startup by taking two keyframes which form a stereo pair. As a result, there are two considerable approaches to initiate the map-model matching:

- Control the two keyframes. This approach is easy because we simply take two images before-hand, then feed them to the PTAM server on every startup. We prepare two images of the scene as we can have reliable feature points in the map. The PTAM process allows the user at startup to stand at a slightly different position from the ones where we took the two images (figure 3.3).

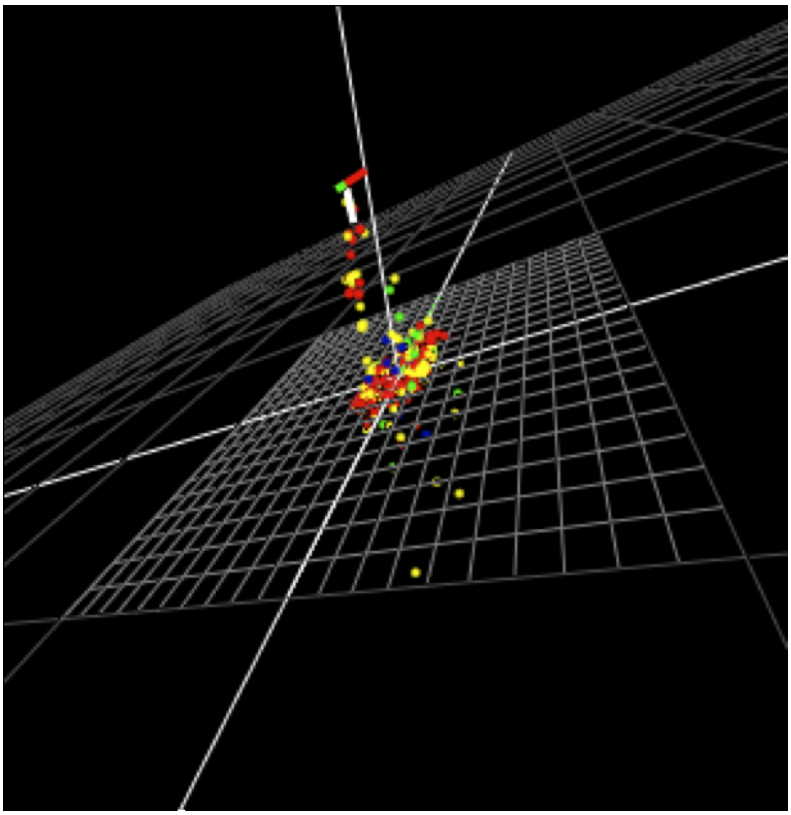

Figure 3.2: A PTAM map. Color of the feature points indicates their stability levels: red > yellow > green > blue > light yellow.

- In the initialization and registration phase, serialize the map to string and save the string to disk. After that on booting up the PTAM process just skip the map initialization, load the map from disk, deserialize and feed it directly to the PTAM process. We can prebuild a large fine-grained map beforehand, thus the system may become very robust. With this approch, a user can stand at many totally different positions and orientations at start up, as long as the positions are inside the prebuilt map.

The second approach is in general more powerful than the first one, but if the scene changes too much then it may be less powerful because parts of the prebuilt map will become invalid and cause noise or heavy load to the PTAM server. The first approach always works given a small static scene where the two images were taken, because the map can be updated dynamically by the PTAM process.

With the first approach, because of the reference implementation of PTAM requirement to make the map initialization stable, actually we must take about 5-10 images and feed them to the PTAM process, with the first and the last one act as the two keyframe (figure 3.3). The tracking and mapping parts of the PTAM process run in two separate threads, as a result slightly different maps may be initialized each time the PTAM process starts. To make sure that the PTAM process always initialize the same map, we must wait for some time after feeding each frame, so that the two threads have enough time to process and communicate with each other. This amount of time is about one second for the MacBook Pro we are using.

3.4 Fusion of GPS and Gyrocompass Devices

The PTAM based camera registration as described in the previous section is quite robust and it has the advantage that it requires only a single camera as for hardware. However, the PTAM based approach does not always work well. For example it may not work when the images taken in real scene is too different from the images used to build the initial PTAM map because of lighting condition changes. We need other sensor support to enhance the applicability of our prototype system. That are GPS and gyrocompass devices as in figure 3.4.

The GSP and gyrocompass devices provide position and orientation information of the mobile device. Together with the prebuilt 3D scene model, we can render the visual aids to visualize viewing fields of surveillance cameras. the orientation information is quite good since the update rate of the gyrocompass is high at 180 Hz. However, the GPS device error is about 5–10 m (figure 3.5), which affects badly the accuracy of the visualization. To improve the GPS device error, model-based tracking method as described in [4] may be used.

Although the more auxiliary devices we add the bulkier and more complicated in programming the mobile device becomes, GPS and gyrocompass devices can be used together with PTAM in the multi-sensor fusion [15] style, which may greatly improves the robustness of the system:

- Initialization problem: The GPS and gyrocompass devices may provide initial position

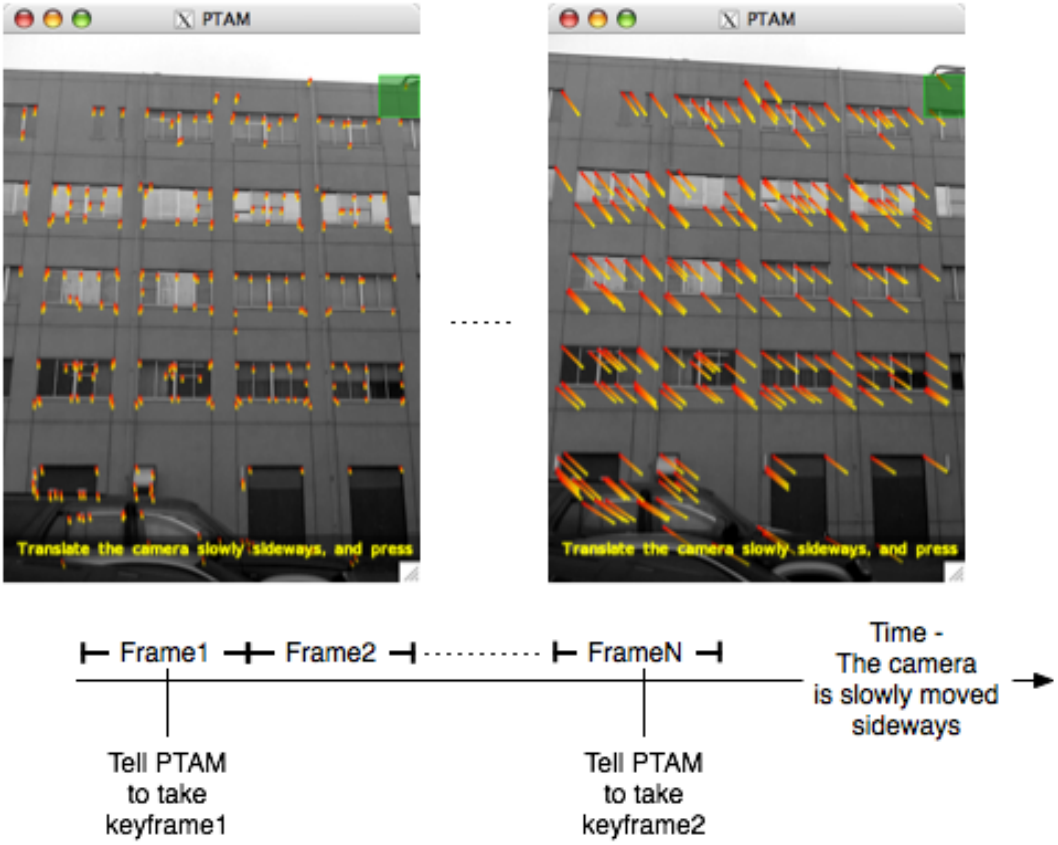

Figure 3.3: Feeding frame to PTAM to initialize map

Figure 3.4: The Garmin GPSmap 60CSx (appendix D) and InterSense InertiaCube3 (appendix E) are connected to the Sony VAIO VGN-UX90PS via USB cable and hub

Figure 3.5: The points do not lie on the red line because of GPS error

and orientation of the mobile device. Although the error of the GPS device is not small, in a large map it may help the PTAM process to reduce the search space.

- Quick movement: PTAM is image-based, thus it does not work well when the user quickly moves the mobile device. At this time, the gyrocompass may provide PTAM the orientation information because its update rate is high. Although the gyrocompass suffers from drifting error and needs to be continuously adjusted by the other sensors in the fusion, quick mobile device movement usually lasts only in a short time, thus gyrocompass may provide precious temporary information during this time.

3.5 Software Development Environment

Visualization methods are expected to be found experimentally and visually. Thus, trial and error methodology can be applied here. To shorten the trial and error cycle, we need a good development environment which allows quick compiling, running and modifying the programs.

At first we used C/C++ language and direct OpenGL API [16]. Because the language and the API are both in too low level, the development speed was slow. Later, C/C++ language was fully replaced by Ruby language. Ruby is an object oriented strongly-typed dynamic language that attracts attention of developers world-wide in recent years. From the perspective of software engineering, the above features provide the following benefits:

- Object oriented feature: Easy to structure the program and later maintain the source code over time.
- Strongly-typed feature: Variables have types, thus we can avoid bugs which usually occur in weakly-typed languages like PHP.
- Dynamic feature: There is no compile time. With static languages like C/C++, a lot time is wasted on compiling source code. With Ruby we can modify source code and run right away.

The development speed was better but still slow, partly because of the slow. It was concluded that the speed of the development is largely affected by the API rather than the language. Consequently, a higher level 3D rendering engine has been adopted: Irrlicht. Later, we read many reviews on the Internet that say that OGRE [17] is easier to use than Irrlicht, thus we migrated to OGRE. OGRE, Object-Oriented Graphics Rendering Engine, is a scene-oriented, flexible 3D rendering engine written in C++ designed to make it easier and more intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics. The class library abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes.

However, OGRE is generally used for creating 3D games thus contain too many features that we do not need. The framework forced us to write too much bloated code for the purpose of this research. As a result, we finally migrated to a combination of Ruby and

C language with pure OpenGL API. At this time Ruby has grown to version 1.9. In this version the interpreter is replaced by a virtual machine, which boosts our Ruby program's speed to about 10x. Moreover importantly it is ridiculously easy to write C extension for Ruby [18]. For program parts that need speed like networking, video frame retrieving, and image processing, we use the Ruby standard library which is written in C, or write them as C extension for Ruby. For program parts that need trial-error or does not need speed like the user interface part, we write them in pure Ruby.

Programming for the iPhone requires Objective-C language, which is similar to C/C++ language. There are some difficulties programming for the iPhone, for example: the API to retrieve video frames from the iPhone camera is not documented, iPhone uses OpenGL ES 1.1 not the standard OpenGL. However, based on the program for the VAIO, we could create the program for the iPhone with some effort.

In short, our experience suggests that keeping a balance between dynamic language and static language is a good choice.

Chapter 4

Evaluation of Visualization Methods

In this chapter, we use the prototype implemented in chapter 3 to conduct an experiment to evaluate the five visualization methods proposed in section 2.2.

4.1 Evaluation Condition

The experiment only focuses on the understandability evaluation. Other things like real-timeness, accuracy, and robustness as described in section 2.1 are not considered.

We have conducted the experiment outside the building of the College of Engineering Systems, at the area as in figure 2.2 and 2.7:

- "Two keyframes" method in section 3.3 is used to initialize the PTAM map.
- The iPhone is used. The video frame size is 304 x 400, and is enlarged to 320 x 421 to fit the width of the screen. It operates at about 7 frame-per-second.
- Students of about 20–30 years old are invited to use the system with the five visualization methods. They point the mobile camera at the wall and see the visualized viewing field of one virtual camera. They can change the position and/or orientation of the mobile. They can also walk around the area.
- After that, each user is asked to rank the understandability of each visualization method in five levels of ranking points (1: worst, 5: best). Each user is forced to give unique ranking points across all five methods.

Figure 4.1 shows a screenshot of the arrow visualization method. The users can use button 1–5 to switch among visualization methods, and button M to toggle between real and map viewing mode. The A button is for administration task, not used by the users.

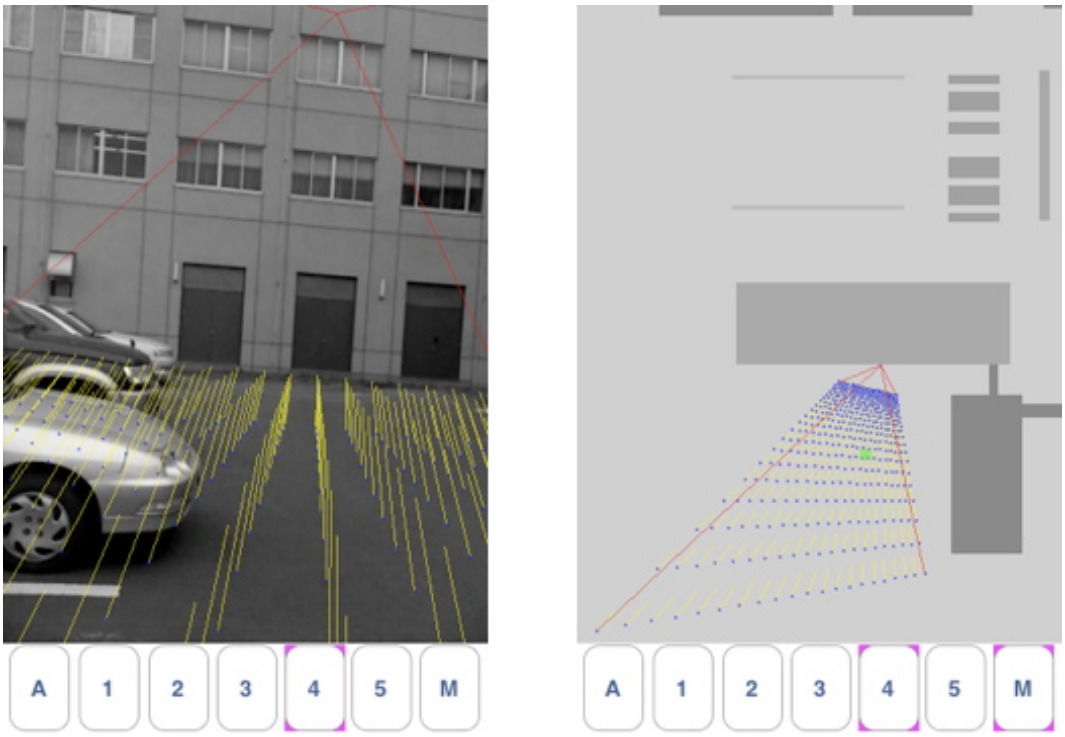

Figure 4.1: Evaluation experiment screenshot of the arrow visualization method in real viewing mode (left) and map viewing mode (right)

Table 4.1: Result of the evaluation experiment of visualization methods

Method	Understandability
Volume	1.0 (1 1 1 1 1)
Shadow	3.2 (3 4 2 4 3)
Contour	2.2 (2 2 3 2 2)
Arrow	4.2 (5 3 4 5 4)
Animation	4.4 (4 5 5 3 5)

4.2 Result and Discussion

The result is shown in table 4.1. The numbers inside parentheses are points given by the users. The outside numbers are the average of the inside numbers.

The most preferred visualization methods are the arrow and animation method. A good visualization method is the one in which the visual aids do not occlude the scene. The arrows and the moving mesh are constantly displayed on the screen but do not occlude the scene over time, hence they give the best understandability. Moreover, this method also points out where the surveillance camera. On the contrary, the visual aids by the volume method is the worst because its CG object usually occludes large parts of the screen.

We have only conducted the experiment in a small outdoor area with the viewing field of one surveillance camera. More extensive experiments in various conditions, such as larger areas, more surveillance cameras, overlapping viewing fields etc. should be conducted before more solid conclusion could be made. In those conditions, methods with non-animated visual aids may be of less understandability because the visual aids may have similar color with the color of the texture of the real scene, or they overlap with one another.

The understandability is largely affected by the way we draw the visual aids. In previous experiments, we did not draw the four straight lines as described at the begin of section 2.2, and the users could hardly understand the shadow and contour methods. Colors of the visual aids may also affect the understandability. Currently the visual aids are drawn semi-transparent:

- Blending function: result color = 0.5 x source color + 0.5 x destination color
- The four boundary lines of the viewing field are drawn in red
- The visual aids at the sides or inside the viewing field are drawn in yellow, and green
- The buildings are drawn in gray

These colors are bright hot and have good contrast, hence users can easily differentiate the different elements. In previous experiments, we used dark colors and users had bad feelings about the visual aids.

A good gene is usually a mixture of pure gene or other mixed gene. Likewise a good visualization can be a combination of various methods. In the future we should try to combine the methods to find ones better than the current methods.

Chapter 5

Conclusion

This research aims to visualize the viewing fields of outdoor surveillance cameras on the screen of mobile devices. In this research, we have proposed:

- The Volume, shadow, contour, arrow and animation methods to visualize viewing fields of surveillance cameras for outdoor MR.
- A new architecture of online camera registration: server-side PTAM [5] system. Our prototype system can provide realtime MR video with no requirement for any GPS or gyrocompass devices, although we can also take these sensors to make the system more reliable.

We have also conducted the evaluation experiment on the visualization methods, using the prototype system. The experiment shows that the prototype system is usable even with a device with limited capability like the iPhone, and the server-side PTAM architecture is practical for outdoor MR.

After pushing heavy work to the server side, the client side has less work left: sending video images and displaying visual aids. We could use the iPhone which is a cell phone as the mobile devices in the prototype system, rather than UMPCs like the VAIO. Many existing games on high-end cell phones have proved that they are able to display 3D CG perfectly using OpenGL ES. Because cell phone is a big market, if we can push outdoor MR to cell phones, the market will push back and become a great force for the advance of outdoor MR. Moreover, today's cell phones are usually equipped with GPS and accelerometer devices. We can use them together with the camera to create a multisensor fusion [15] to improve the accuracy of the system.

Moore's Law is the observation that the amount you can do on a single chip doubles every two years. But it may have gradually come to an end: devices can hardly get smaller and faster. We cannot only sit and hope that mobile devices will constantly become much faster without changing their physical sizes. We should find other options. A new trend in computer processor technology is multicore. Rather than producing faster and faster CPUs, companies such as Intel and AMD are producing multicore ones: single chips containing two, four, or more cores. Within several years there may be hundreds of cores in a CPU.

Each PTAM session requires two cores. The server in our prototype system is a dualcore and can only process one PTAM session at a time. A server with more cores can simultaneously process more PTAM sessions.

Multicore hardware requires parallel processing software. The frequency of a CPU does not go higher (about 3–4 GHz as of current time), but its computation potential goes higher because there are many cores integrated into it. If a program is not parallel, it will only run on a single core at a time and users will think that the program is slow. However, parallel programming is hard and must be further studied [14].

With these technological advances, we believe that in the future more powerful viewing field visualization systems can be built and put in wide use, and they can help realize more safe society.

Acknowledgements

There are many people and organizations I would like to thank, without my study in computer vision and image processing and this research would not have been possible.

I would like to thank professor Yuichi Ohta, associate professor Yoshinari Kameda, and associate professor Itaru Kitahara of the Computer Vision and Image Media Laboratory for their guidance on the field of computer vision and image processing, their support, comments, and valuable advices in the three years on this research. I would like to thank Shinya Yamazaki, Masayuki Hayashi, and other members of the laboratory for their help and discussion.

I would like to thank the Japanese Ministry of Education which has given me the chance to come to Japan and provided me with nancial support in the early years. I would like to thank Watanuki International Scholarship Foundation, and Epson International Scholarship Foundation for their generous financial support in the last two years.

I would like to thank my family and friends in Viet Nam and friends in Tokyo, Takamatsu, and Tsukuba for their time, support and encouragement.

Appendix A

Sony VAIO VGN-UX90PS

See figure 1.2.

- Size: 150.2 x 95 x 32.2 mm
- Weight: 520 g
- CPU: Core solo U1400 1.20 GHz
- RAM: 512 MB DDR2
- OS: Microsoft Windows XP Professional (SP2)
- LCD Display: 4.5 inch wide TFT color LCD
- Display mode: 16190000 colors
- Video memory: 128 MB
- Interface:
  - Hi-speed USB (USB 2.0)
  - Network (LAN) connector
  - Wireless IEEE801.11a/b/g, WPA2. Wi-Fi. Bluethooth2.0
- Camera: Web camera MOTION EYE x 2
  - 1/8 inch VGA Progressive CMOS, 3.3 M pixel, f = 2.6 mm F4
  - 1/4 inch SXGA Progressive CMOS, 13.4 M pixel, f = 3.8 mm F4

Appendix B

Apple iPhone 3G

Figure B.1: The Apple iPhone 3G used in preliminary experiment

- CPU: 620 MHz ARM
- GPU: PowerVR MBX Lite 3D
- Memory: 128 MB DRAM
- Display: 320 x 480

- Wi-Fi: 802.11b/g
- Assisted GPS, Accelerometer
- Battery: Up to 6 hours on Wi-Fi
- Capacity: 8 GB (flash memory)

Appendix C

Apple MacBook Pro

Figure C.1: The Apple MacBook Pro used in the prototype system

- CPU: Intel Core 2 Duo (Number Of Cores: 2) 2.4 GHz, L2 Cache: 3 MB
- Memory: 2 GB 667 MHz DDR2 SDRAM
- Bus Speed: 800 MHz

- OS: Mac OS X 10.5.6, Microsoft Windows XP Professional SP2

Appendix D

Garmin GPSmap 60CSx

Figure D.1: Garmin GPSmap 60CSx

- Electronic compass feature:

  - Accuracy: 2 degrees with proper calibration (typical); 5 degrees extreme northern and southern latitudes
  - Altimeter feature: (GPSMAP 60CSx only)
  - Resolution: 1 foot
  - Range: -2,000 to 30,000 feet
  - Elevation computer: Current elevation, resettable minimum and maximum elevation, ascent/descent rate, total ascent/descent, average and maximum ascent/descent rate
  - Pressure: Local pressure (mbar/inches HG)

- Physical:
  - Size: 2.4W x 6.1H x 1.3D inches (61 x 155 x 33 mm)
  - Weight: 7.5 oz. (213 g) est.
  - Display: 1.5 x 2.2 inches (38.1 x 56 mm) 256-color transflective TFT (160 x 240 pixels) (160 x 240 pixels)

- Navigation features:
  - Waypoints/icons: 1000 with name and graphic symbol, 10 nearest (automatic), 10 proximity
  - Routes: 50 reversible routes with up to 250 points each, plus MOB and TracBack modes
  - Tracks: 10K point automatic track log; 20 saved tracks 500 points each let you retrace your path in both directions
  - Trip computer: Current speed, average speed, resettable max. speed, trip timer and trip distance
  - Map datums: More than 100 plus user datum
  - Position format: Lat/Lon, UTM/UPS, Maidenhead, MGRS,
  - Loran TDs and other grids, including user UTM grid only

- GPS performance
  - Receiver: 12 channel SiRFstar III (TM) high-sensitivity
  - GPS receiver (WAAS-enabled) continuously tracks and uses up to 12 satellites to compute and update your position
  - Acquisition times: Warm: $<$ 1 sec; Cold: $<$ 38 sec; AutoLocateTM: $<$ 45 sec
  - Update rate: 1/second, continuous
  - GPS accuracy: Position: $<$ 10 meters, typical; Velocity: .05 meter/sec steady state

- DGPS (WAAS) accuracy: Position: < 5 meters, typical; Velocity: .05 meter/sec steady state
- Protocol messages: NMEA 0183 output protocol
- Antenna: Built-in quad helix receiving antenna, with external antenna connection (MCX)

Appendix E

InterSense InertiaCube3

Figure E.1: InterSense InertiaCube3

- Degrees of Freedom: 3 (Yaw, Pitch and Roll)
- Angular Range Full: 360° - All Axes
- Maximum Angular Rate: 1200° per second
- Minimum Angular Rate: 0° per second
- RMS Accuracy: 1 in yaw, 0.25° in pitch and roll at 25 °C
- RMS Angular Resolution: 0.03°
- Serial Interface Update Rate: 180 Hz

- Minimum Latency: 2 ms for RS-232 (PC host OS dependent)
- Prediction: up to 50 ms
- Serial Rate: 115.2 kbaud
- Interface: RS-232 Serial (shown above)
- Size: (without mounting plate) 1.031 in x 1.544 in x 0.581 in (26.2 mm x 39.2 mm x 14.8 mm)
- Weight: 0.6 ounces (17.0 g)
- Cable Length: 15 ft. (4.572 m) - Max. 75 ft (22.86 m)
- Power: 6 VDC, 40 ma
- Operating Temperature Range: 0 to 70 °C

Bibliography

[1] Yuichi Ohta and Hideyuki Tamura. *Mixed Reality - Merging Real and Virtual Worlds*. Ohmsha, 1999.

[2] Yoshinari Kameda Takahiro Tsuda, Itaru Kitahara and Yuichi Ohta. Smooth video hopping for surveillance cameras. *The 33rd International Conference on the Computer Graphics and Interactive Techniques (SIGGRAPH2006)*, 2006.

[3] Taisuke Takemasa Yoshinari Kameda and Yuichi Ohta. Outdoor see-through vision utilizing surveillance cameras. *IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR04:151–160, 2004.

[4] Gerhard Reitmayr and Tom W. Drummond. Going out: Robust tracking for outdoor augmented reality. In *Proc. ISMAR 2006*, pages 109–118, Santa Barbara, CA, USA, October 22–25 2006. IEEE and ACM, IEEE CS.

[5] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234, 2007.

[6] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferenencing system. In *Proc. IWAR 99*, pages 85–94. IEEE, IEEE CS, October 21–22 1999.

[7] Sekai camera, http://sekaicamera.com/.

[8] Enkin, http://www.enkin.net/.

[9] Tomokazu Sato Motoko Oe and Naokazu Yokoya. Estimating camera position and posture by using feature landmark database. *Lecture Notes in Computer Science*, pages 171–181, 2005.

[10] L. Williams. Casting curved shadows on curved surfaces. *SIGGRAPH*, pages 270–274, 1978.

[11] William R. Mark Gregory S. Johnson and Christopher A. Burns. The irregular z-buffer and its application to shadow mapping. Technical report, University of Texas Department of Computer Sciences Technical Report, March 2004.

[12] Frank Crow. Shadow algorithms for computer graphics. *SIGGRAPH*, pages 242–248, 1977.

[13] Ptam reference implementation source code.

[14] Joe Armstrong. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, July 2007.

[15] David L. Hall and James Llinas. *Handbook of Multisensor Data Fusion*. CRC Press, June 2001.

[16] Mason Woo Jackie Neider OpenGl Architecture Review Board, Dave Shreiner and Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2*. Addison-Wesley, 5 edition, 2005.

[17] Gregory Junker. *Pro OGRE 3D Programming*. Apress, 2006.

[18] Ruby language repository. How to make extension libraries for ruby.