Incorporating Video into Google Mobile Street View

By Christina Wright

S.B., C.S. M.I.T., 2008

Submitted to the
Department of Electrical Engineering and Computer Science
In partial fulfillment of the requirements for the degree of
Masters of Science in Computer Science and Engineering
At the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2010

[June 2010]

Author.....

Christina Wright

Certified by.....

Hal Abelson
Professor of Computer Science and Engineering
Thesis Supervisor

Certified by.....

Carole Dulong
Cityblock Pipelines TLM
Thesis supervisor

Accepted by.....

Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Incorporating Video into Google Mobile Street View

by

Christina Wright

Submitted to the

Department of Electrical Engineering and Computer Science

February 2010

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Mobile Street View is a compelling application but suffers from significant latency problems, especially in limited bandwidth circumstances. Currently, the application uses static images to display street level information. Utilizing video in addition to images has the potential to improve the Street View user experience. In this paper, we examine the design and performance of mobile Street View and consider how to use video to reduce user visible latency. Video also allows for alternate navigation methods which could improve the application's ease of use. We created a prototype on Android to examine the plausibility of incorporating video into mobile Street View. Comparing the performance of our video prototype to the traditional step-by-step Street View approach, we found a 4x increase in the speed of viewing an entire street. For the video prototype I also found significant improvement in both user visible latencies and useful screen time. Additionally, I found that the time to fetch video chunks was less than the time to display them.

Thesis Supervisor: Hal Abelson

Title: Professor of Computer Science and Engineering

Thesis Supervisor: Carole Dulong

Title: Cityblock Pipelines TLM

Acknowledgements

I would like to thank Google and the MIT VI-A program for giving me the opportunity to work on this exciting project.

I would like to thank my MIT supervisor Hal Abelson and my Google supervisor Carole Dulong for their guidance and support throughout my thesis. I would also like to thank Jiajun Zhu for all his work on server side adding support for video, and in helping me to design the video prototype. I'd like to thank Pascal Massomino for explaining to me the workings of various video formats. I would like to thank Kevin Law for working with me on the Android Street View application, and later helping me with GMM related things. I'd like to acknowledge Jack Palevich for writing the existing Street View application and for his help in understanding its workings.

Also, I would like to thank everyone who participated in the instrumentation dogfood: Carole Dulong, Luc Vincent, James Knighton, Jeremy Pack, Owen Brydo, Jean-Yves Bouguet, and Augusto Roman. And to everyone I worked with on the Street View team for always being welcoming and willing to help.

Thank you to all my friends for keeping me sane. Finally, I'd like to thank my parents, for their unconditional support.

Contents

List of Tables

List of Figures

1 Introduction

Writing applications for mobile devices is uniquely compelling because of the immersive on-the-go experience provided. Unfortunately, this portability comes at the cost of reduced capabilities. Mapping applications in particular are perfectly suited for mobile use. Google provides mobile versions of Google Maps and Google Street View as native or web-based applications. As the Street View team strives to bring additional functionality to users, we need to keep in mind the limitations of our platforms. In this project I examined whether existing mobile technologies are sufficient to support adding video capabilities to Google's mobile mapping applications, particularly Street View (SV). Additionally, I considered how the mobile Street View (MSV) application might utilize video, and implemented a Video Street View prototype on Android. Finally, I measured the performance of our video prototype and compared it to the existing mobile Street View application.

1.1 Research Goals

The major goal of this work was to determine if and how video could be used to improve mobile Street View. There were five major tasks to achieve this goal.[i]

1.1.1 Analysis of Video Encoding for use in Mobile Applications

The theoretical aspect of this research tried to determine the feasibility of using video in an interactive application hosted on a mobile platform. The key goal was to determine if video files of acceptable quality could be provided by a video server over a mobile, cellular telephone network fast enough to provide an acceptable user experience without excessive delays, gaps, or jitter.

This goal has two subgoals: determining the minimum size a video file can be while maintaining acceptable quality, and considering whether current transmission protocol abilities are sufficient to obtain those optimized videos in a reasonable time. To achieve these goals, I performed analysis on the video encoding parameters including format, resolution, frame rate, bitrate, frame gap, and other factors affecting both playback quality and the size of the video files to be transmitted. I then considered the transmission capabilities of commonly used protocols to determine if the mobile application can play video smoothly. In order to accomplish smooth playback, it must be possible to obtain the next video segment in less time than it takes to play the current segment.

1.1.2 Understand the Mobile Street View Design

Jack Palevich ported the Street View application that to the Android platform. However, the project was completed very quickly and the software design was not documented well at the time. Therefore, I had to review the code baseline in order to capture its overall design and also to incorporate standard instrumentation. The code review was also necessary as a step towards making improvements to the application.

1.1.3 Validate Video Analysis by Implementing a Prototype

A working prototype demonstrates end-to-end feasibility in a way theoretical analysis cannot. Thus, I complemented the theoretical analysis by implementing a prototype of Mobile Street View utilizing

video segments in addition to the current static imagery. The prototype enhanced the mobile Street View application that runs on the Google Android mobile operating system. I modified the software so that it would play video segments to give the user a view of traveling down a street.

1.1.4 Analyze Mobile Street View Performance

After I reviewed the application design, I instrumented the code in accordance with the Google policies related to performance analysis. (See section 7 for more on instrumentation) Once instrumented, I ran experiments to collect performance metrics and then analyzed the results obtained.

I applied the same techniques for instrumentation to the Video Street View prototype. Then I collected performance metrics and presented them in comparison to the Mobile Street View application's metrics. I aim to demonstrate that incorporating video into Street View improves specific use cases of the application.

1.1.5 Recommend Improvements to the Mobile Street View Application

I wanted to identify areas for improvement of the existing application. Towards this goal, I examined the instrumentation results, in addition to the existing code, to determine where to focus optimization efforts. This also included recommendations for incorporating video features into the application and determining some of the issues impeding implementation.

1.2 Caveats

The complexity of mobile computing can make accurate assessment very difficult if all factors are considered. Thus I made some trade-offs in order to simplify the effort while maintaining validity.

- Android Platform: My experiments and implementations were done exclusively on the Android platform using the HTC G1 smart-phone (HTC Products), though many of the results are applicable to all mobile platforms.
- Video Formats: There are many video encoding formats in use including mp4, avi, mpeg, etc. I chose to limit the analysis to the 3GP protocol because it is the only format supported by the video player on Android. There are also tools available at Google to synthesize videos with the desired settings in 3GP format (3GPP file format (3GP)).
- Communication Protocols: The common protocols supported by cell phone networks today are WiFi and 3G. Many other protocols exist but these seem to be the most prevalent. I did not test the effects of contention or signal attenuation and the resulting effective bandwidth reduction.
- Protocol overhead: When the application requests video from the server, there is a lot of overhead due to the TCP/IP communication. For instance, I did not measure DNS address resolution, routing, caching, etc. This was because I did all the instrumentation on the client-side so it was not possible to measure these factors. We do get a general sense of the server performance from our round trip times, but we do not obtain a fine-grained breakdown of where the time is being spent.
- Platform performance: The instrumentation experiments used several Android phones on WIFI, but did not measure 3G connectivity since not all phones were 3G enabled. Because Google's test servers were not accessible publicly, some assessments had to be done using the device

emulator that is provided with the Android Software Developers Kit. Analysis of data transfer over a WIFI connection or wired Ethernet connection is different from a cellular connection, as illustrated by Figure 1. I expect that on a 3G connection the server communication times would decrease by a constant factor, thus the choices made on WIFI will likely hold over to 3G connections.

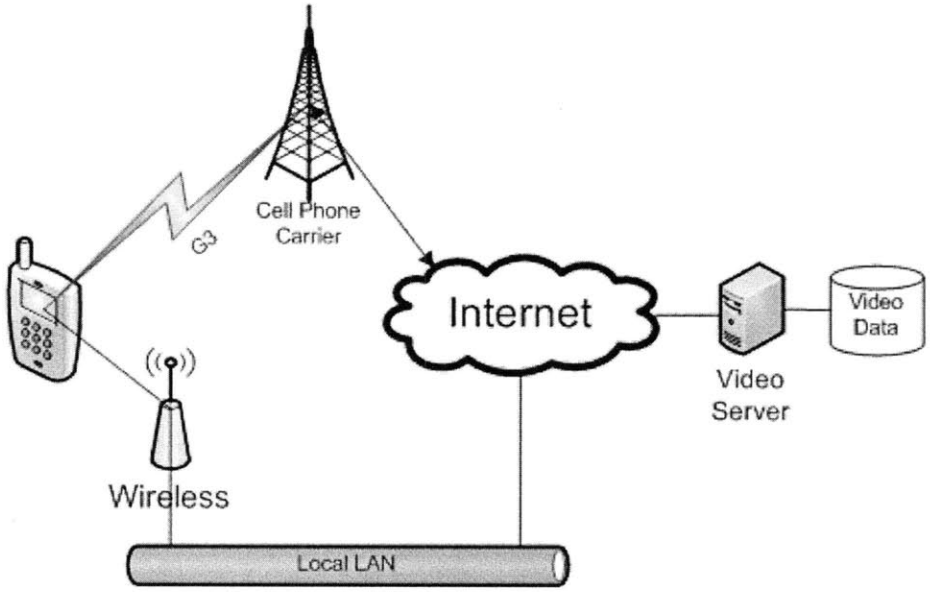

Figure 1 : Data paths for cellular connection (top) and WIFI connection (bottom). Note that the cellular connection path uses cell phone towers as opposed to the wireless connection, which instead routes through a LAN.

1.3 Document Organization

In this paper I propose and analyze a procedure for incorporating videos into the Street View application. Section 3 describes the Street View application from a user perspective then proposes one way in which video could be incorporated. Section 4 examines various video configurations to find one suitable for mobile Street View, then analyzes the feasibility of using video given current technology constraints. Section 5 explains the workings of the Street View application as it currently stands. Section 6 details how I altered the application to include video. Section 7 describes how I obtained timing measurements on both the existing and prototype application, and analyzes the results.

2 Background

This section gives some background information on the basics of Street View and Android necessary to understand the work described in the rest of this paper.

2.1 Google Maps and Street View

Google Maps provides interactive online map data to users. In addition to abstract road-mapping information, the Google Maps application can display satellite imagery giving the user a sense of the actual view of the region. In contrast to the overhead views provided in Maps, Street View provides an

eyelevel perspective. It displays imagery of what people would see if they were to actually drive down a particular road.

Google obtained the Street View imagery using its own fleet of modified cars. The cars drove routes while tracking location via GPS and take pictures of the roads along the way. Afterwards the data was post-processed to associate every location with its corresponding images. In addition to locating images, processing stitched the images from the cameras into a single master panorama. It was necessary to warp the imagery while stitching since the Street View cameras view locations in a vaguely spherical manner, but the end panorama was a flat image. When the user later views a location in Street View the application undoes this warping before displaying the imagery. The master panoramas were also scanned for identifying information or offensive content and that content was made unrecognizable through blurring. The result was a set of panorama images aligned with streets and spaced one to two meters apart.

These master images were assigned a unique panorama ID and a corresponding configuration file. The configuration file describes all of the metadata relevant to the panorama. For instance, one value stored in the panorama configuration is the yaw of a panorama. The **yaw** of a panorama is the difference between the direction of the image and due north. Panoramas have non-zero yaws because each panorama was stitched such that the direction the car faced was in the center of the panorama. See Section 5.3 for more details about the configuration file and how it was used.

Not every panoramic image is available to users via the Street View application. Some panoramas may be from an older generation of images, and thus retired. Others are simply too close together since application serves images with a separation of 10m. The locations that are available by client facing applications are called **live panoramas**.

In order to provide the best experience, Street View imagery is served in stages, or **zoom levels**. Each stage has a set of 514x514 **tiles** constructed from the master panoramic image. Zoom level 0 consists of a single tile containing the entire panorama for its location. Each subsequent zoom level quarters the tiles from the previous level, thus serving four times the number of tiles. This is continued until the total resolution of all the tiles at a level is comparable to the resolution of the master panorama. The specific tiles within a zoom level are identified by an x-y coordinate. The applications load low zoom levels to provide imagery quickly, and then display higher quality imagery (from high zoom levels) as it becomes available. (Stephane Lafon)

2.2 Android Programming Environment

Though many of the results derived are applicable to any mobile platform, our tests and implementations were done solely on Android G1 devices. This is a brief introduction of how applications in Android work.

Android applications are written in Java. There are four types of Android application components, but Street View uses only one: Activities. "An ***activity*** presents a visual user interface for one focused endeavor the user can undertake." (Android Application Fundamentals) Application components are activated by the launch of a specific type of message called an intent. When the system has received the

activity launch intent, it calls the activity's onCreate() method which handles the object's initial setup. Activities can be shutdown by calling *finish()*. If the activity had a parent, the parent is notified that its child activity has completed. Calling finish causes onDestroy() to run, which releases all resources.

For instance, the Street View application is a single Activity where the 'focused endeavor' is the viewing of images for a sequence of street locations. An intent is launched when the user selects to enter Street View from Maps, causing its onCreate() method to be called. This method could contain initialization work such as obtaining the location's imagery or preparing the renderer. When the user has finished with Street View and presses the back button, the onDestroy() method is called. Here Street View releases resources no longer necessary, such as memory used to store images.

Obviously there is much more to the Android programming environment. This is only a brief summary of the portions necessary to understand the work described in the following sections.

3 User Experience

It is helpful to have an understanding of how the application is used before addressing performance and alterations of the application. This section discusses the current Mobile Street View application from a user's perspective. Then it addresses some of the major drawbacks of the application. Finally Section 3.3 looks into how incorporating video can potentially improve the application.

3.1 How Street View Works

First this section describes how a user may launch and interact with the Mobile Street View application. Figure 2 shows a sequence of screenshots that illustrate what users may see when they invoke MSV from the Google Maps on Mobile application.

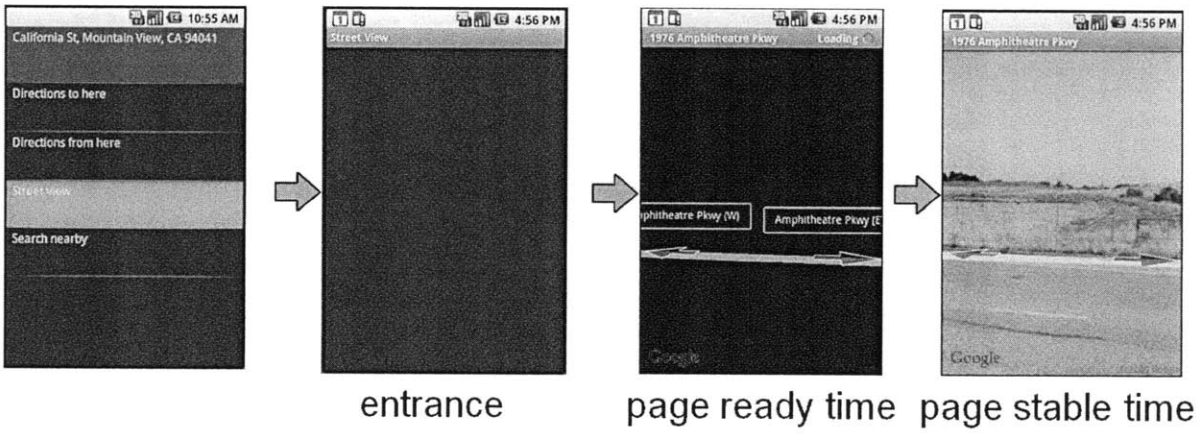

Figure 2: Screenshots depicting an example Street View session's stages.

A Street View session begins when the user enters Street View from the Maps application. Street View can be entered in one of three ways: from a particular location, from a location search, or from driving directions. Once selected, the Street View application launches and the user sees a blank screen for a short period. The time that it takes to launch the Street View application is called the **entrance time (ent)**.

The application begins at the location provided by the Maps application (usually the same location the user was viewing when the Street View application was launched). When a location is loaded the user first sees a street *movement control overlay*. This overlay displays an approximation of the road(s) at the given location, as well as a set of arrows which indicate the directions that the user can move in. The time it takes for the application to reach a useable and visible state is called the **page ready time** (prt).

Shortly after the overlay is rendered, the street imagery for the location is rendered. The lowest resolution imagery is rendered first while the application waits for higher resolution images to arrive. Once the highest resolution imagery for the current zoom level is rendered, the application then waits for further user input. The time it takes for the application to reach a state where no further data is expected from the server is called the **page stable time** (pst).

At this point the user can begin interacting meaningfully with the application. He can pan, zoom in and zoom out, and move or 'step' to adjacent locations. Panning in Street View is equivalent to the observer turning while remaining in a fixed position varying the compass direction and vertical angle of the observer; moving is equivalent to repositioning the observer up or down the street while maintaining the same viewing direction. Pan and zoom operations work with currently loaded imagery. Moving requires fetching new imagery from the street view server for the new location.

Figure 3 is a state transition diagram showing how the application behaves in response to user interactions.

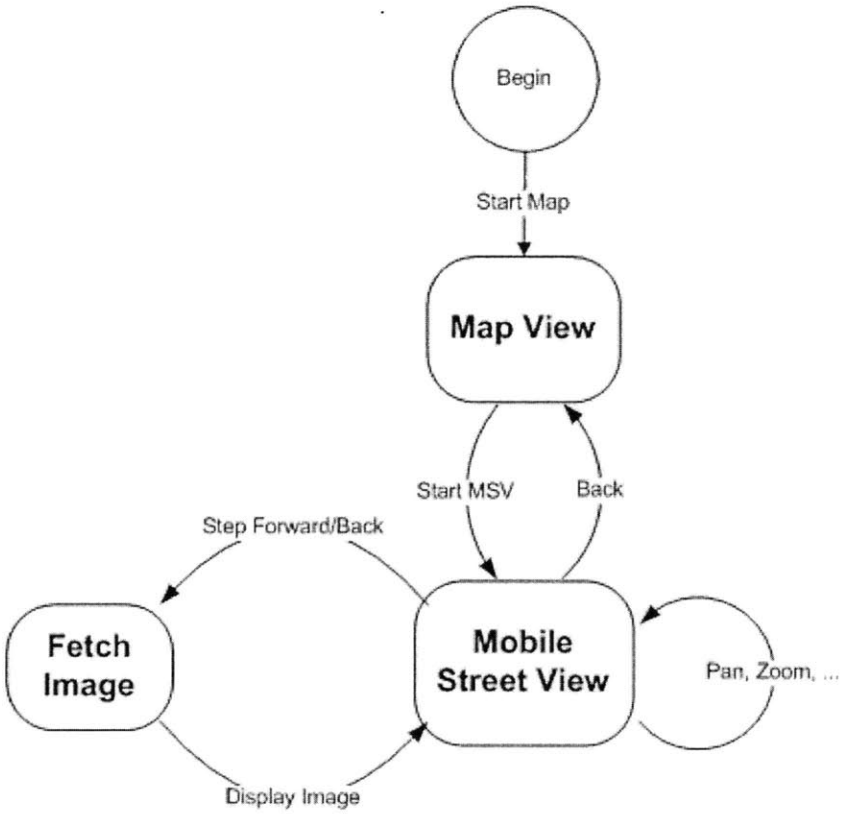

Figure 3: State diagram of current Street View application.

The user has three possible interactions: zooming, panning, and moving. Panning is done via a finger swipe across the touch screen or movement of the trackball. It causes the overlay and imagery to pan as directed. Zooming is initiated by tapping the screen to activate the zoom controls and then a tap on the + or – symbols that appear. Zooming resizes the imagery and may cause higher resolution images to be downloaded. Zooming in also causes the movement control overlay to disappear at any zoom level higher than the initial level. The zoom and pan operations do not cause the location to change so the user can remain on the same panorama. Moving is done by tapping one of the overlay arrows or by pushing the "menu" button and then selecting "step forward" or "step backward" from the menu. This causes the application to move approximately 10m to the adjacent panorama location, initiate the data retrieval (causing a brief blank screen while the data is fetched), and render the new overlay and imagery when it arrives from the server. When the user is finished using the Mobile Street View application, he may exit by pressing the back button or selecting "Go to map" from the menu. He is then returned to the Maps application at the original location.

3.2 Issues

Street View has a number of potentially interesting use cases: viewing a single location, exploring a city block, or previewing a route to a travelling destination. The current design has focused on optimizing the first use case, but to improve the application Street View developers should consider how to provide users with a good experience for other possible use cases.

When using the application the first noticeable issue is how long the user must wait for each new location's panorama to display. (See Section 7.2 for quantitative measurements of this wait time.) While waiting the screen is blank, this provides no value to the user beyond an indication that the application is working to fulfill the user's request. The delay is noticeable even when the connection to the network is using Wi-Fi (802.11 wireless local area networking), the fastest supported connection. On slower connections such as 3G and Edge networks (IMT-2000) the delay can be significant, discouraging the user from moving to new locations, or even from using Street View at all.

In addition to the long wait times, the interactions provided by the current user interface are not well suited for any use case other than the viewing of a few adjacent locations. If the user wants to explore a section of a street, then at each location he will need to navigate to the next panorama (take a 'step'). This may be reasonable for short distances, but consider that since navigating a city block means issuing around 15 step commands, the user will have to endure 15 page ready times to view just a single block. To support the use case of viewing a street, it would be better to reduce the interactions to a few commands such as specifying direction, speed, and distance or specifying an intermediate point along a route to a destination. Mobile Street View's user interface could easily be improved to become an appealing application for alternate use cases.

Obviously performance is a major factor for interactive programs. In order to reduce the page ready time, Street View developers should consider whether the current image resolutions are optimal. For viewing a single location, users may need a fairly high-resolution image in order to read store names; but for previewing a route, often the user is less concerned with the details of each individual location and thus would tolerate a much lower resolution in exchange for a faster travel speed.

In addition to optimizing the resolution of the imagery, the interface should eliminate the "dead" time between locations. The current location should remain visible until new imagery has been received. The use of asynchronous pre-fetch techniques could reduce the wait times between locations. Changing locations could also be smoother using fade-in and fade-out transitions.

3.3 Incorporating Video

The next section explores how using video could improve the user experience on Mobile Street View. Videos could be used to show traveling down the street and provide the same information using less bandwidth because of the nature of the images Street View is serving. Since adjacent locations on a road look similar, video compression stands to reduce the amount of data downloaded by the client. Reducing the data size could reduce or even eliminate visible delays, thus improving the experience.

In addition to potentially providing faster service, video allows for alternate navigation controls. By automatically playing the next segment of a video instead of waiting for the user's step command, the application can deliver information more seamlessly. This has the potential to improve the experience when a user wants to get a sense of a large region, such as a city's downtown. Video would also provide the user a visual preview of traveling along a route to a destination allowing him to identify landmarks ahead of time and showing him a turn-by-turn sequence.

Thus, I explored how video could be incorporated into Mobile Street View to improve both responsiveness and ease of navigation without compromising the simplicity of the UI. The approach I took was to augment the current application to use video segments in addition to images. The Street View flow depicted in Figure 3 still applies but now in addition to panning, zooming, and moving, the user can also choose to enter a video mode (see Figure 4). Ideally video mode and normal image based Street View will be integrated seamlessly, so that the only thing the user notices is an enriched set of navigation controls.

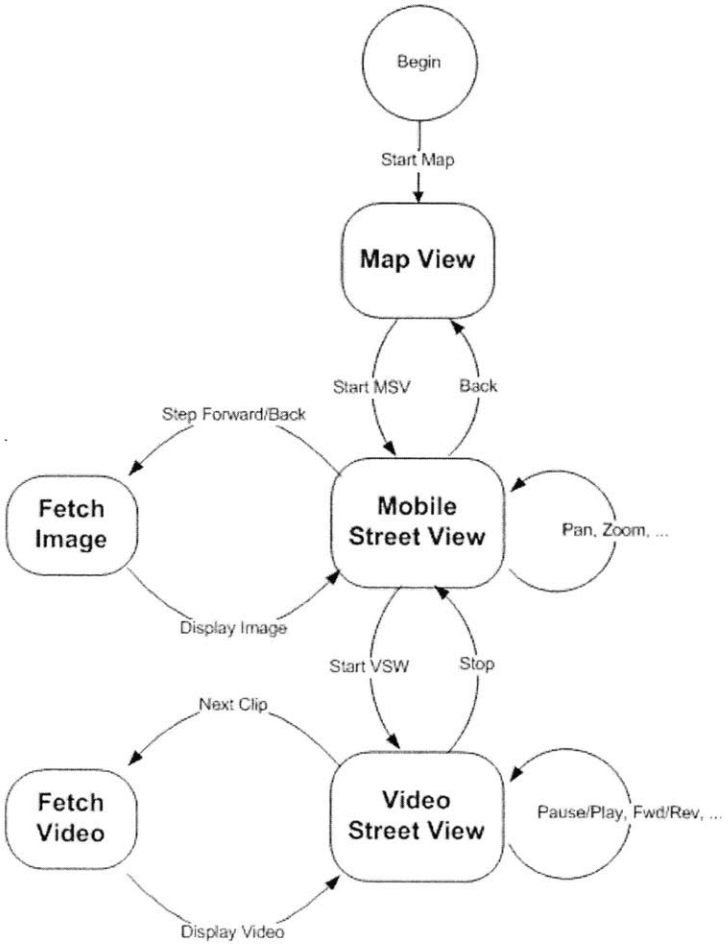

Figure 4: State diagram of Street View application augmented with a video mode.

In video mode, the application would display video segments joined together without user interaction until the end of the route or until there is an intersection or fork. At this point the application would not know in which direction to continue in so the user must provide direction. Additionally, at any point the user may choose to pause, increase/decrease speed, or pan. Panning simply alters the viewing angle, in the same manner as for still imagery. Changing the speed increases or decreases the perceived movement along the street. This can be accomplished by either playing the frames faster (increasing the frame rate) or by skipping intermediate frames. Pausing halts the video at the current location and allows the user to interact with the panorama as in the normal Street View application: he can zoom, switch direction, or choose to continue playing.

4 Video Analysis

One of the objectives of this thesis was to assess the potential of using video on a cell phone platform in general. The analysis in Section 4.2 is a combination of quantitative calculations of the size of the video clips along with a qualitative assessment of the quality of the resulting video. I used the calculations to produce video clips used in a video prototype implementation that empirically validates the analysis results.

This section describes how I determined the parameters of the video format. The obvious compromise is between the quality of the video and the amount of data transmitted over the cellular network. Then I look into the feasibility of using videos to transmit Street View data on a mobile platform. The primary concern was whether the processing and network bandwidth of the device were sufficient to provide a decent user experience.

4.1 Constructing Street View Videos

The Street View team produces master panoramas for each geographic location as discussed in Section 2.1. Now the backend reassembles lower resolution versions of those panoramas into video files. The client wants videos that display frames with a fixed-distance separation. Street View's original data was captured with images at a fixed-time frequency. Thus, the backend would rather stitch the processed panoramas together than use the original images captured. Google doesn't need to acquire additional imagery to support adding video features since the existing panorama locations are nearby enough to provide a smooth video experience. Jiajun Zhu wrote a script to assemble Street View videos from panoramas. The script took Street View specific video parameters as inputs and produced flash videos as output.

4.2 Determining Video Parameters

The first step in incorporating video into Mobile Street View was to determine the optimal video parameters to ensure quality of the display while minimizing data transfer times. My goal was to determine the best parameters for viewing a city block while keeping video file sizes small. Small file sizes are preferable on mobile devices with relatively low bandwidths and low screen resolutions typical of current cellular "smart phones". I chose the city viewing use case because of the higher density of interesting landmarks. A metropolitan area also offers more intersections, and thus more navigation opportunities, than a highway or neighborhood would.

This project was done using the Android G1 phone. At the time of the research the G1 supported only 3GP (3GPP file format (3GP)) video format and had a screen resolution of 480x320. I evaluated the following alterable parameters:

- Resolution
- Framerate
- Bitrate
- Frame spacing/speed
- Cropping
- Blurring

The first three are standard video parameters; the latter three are Street View specific parameters. Ideally, spacing and cropping would be handled on the device to give the user finer control over the display; however, due to platform limitations discussed in Section 4.3.3, the videos were processed offline for the experimental phase. All tests were done in landscape mode. I evaluated the quality of the videos by playing them on G1 devices and judging their quality myself.

I created video segments by manipulating the data from a panoramic Flash video segment. This baseline video had a resolution of 600x300 and 25 fps frame rate. The size of the baseline was 872.2 KB for a four second clip. Appendix A describes the exact process and the tool used for generating videos with the various parameter values.

4.2.1 Resolution

The goal here was to determine if the quality degradation from using a lower resolution image in order to reduce the file size was acceptable. The 3GP format supports four possible resolutions: 128x96, 176x144, 352x288, and 704x576. The baseline sample was converted to the 4 different resolutions using the *ffmpeg* utility program. I padded the video to prevent any cropping or stretching of the imagery during the conversion. With the full panorama the warping of the street is very noticeable. Table 1 below shows the file size and quality assessment of each of the four tested resolutions.

Resolution	Size (kB)	Observations
128x96	152.4	Features visible, but notably grainy
176x144	234.9	In between
352x288	352.0	Crisp and detailed
704x576	471.1	Won't display because larger than screen resolution

Table 1: video parameter test results - resolution (panoramic)

In addition to the full panoramic view, I also tested a version where the image was cropped so that only the forward facing portion of the street was visible. I determined that this was a more realistic test case because the user is normally shown a small field of view in Street View. Also, with only half the field of view, the image warping was much less noticeable. This time I began with a 300x300 panoramic flash video with 25 frames/second of size 368.5kB. Table 2 below shows the results obtained for the cropped videos.

Resolution	Size (kB)	Observations
128x96	128	Slightly more tolerable than for panoramic video
176x144	225.6	In between
352x288	357.3	Crisp and detailed
704x576	513.2	Won't display because larger than screen resolution

Table 2: video parameter test results - resolution (cropped)

In general, I found that the lower resolutions were more tolerable than anticipated. I decided to use the 352x288 resolution, but users can tolerate smaller resolutions.

4.2.2 Frame rate

The framerate is the number of frames per unit time at which the video player displays the video frames to the user. Higher frame rates provide a smoother video experience, but also require more frames per

second and thus a greater bandwidth to transmit. My goal was to find the lowest framerate that still provided a reasonably smooth video. I tested frame rates between 1 and 8 frames per second and found that a framerate of 3 or lower was noticeably jumpy, so I decided to use a framerate of 4 frames per second.

4.2.3 Bitrate

Bitrate measures the number of bits per unit time the video requires. A low bitrate video will use less bandwidth than a high bitrate video. Holding resolution and framerate steady and altering the bitrate will determine the aggressiveness of frame compression. A low bitrate will cause more compression artifacts such as blurring. My goal was to find the lowest tolerable bitrate without signification quality degradations.

I first did a preliminary test, shown in Table 3, to determine where to focus more detailed efforts, shown in Table 4. I tried to test bitrate differences of about 100kb/s.

Resolution	Bitrate (kb/s)	Notes
128x96	~100	At maximum bitrate already and quality too low.
176x144	~200	Reasonable, can try decreasing quality
352x288	~300	Reasonable, can try both decreasing and increasing

Table 3: video parameter test results - preliminary bitrate test

Set	Resolution	Bitrate (kb/s)	Size (kB)
1	176x144	106	132
2	176x144	181	225
3	352x288	126	157
4	352x288	182	227
5	352x288	287	357
6	352x288	386	480
7	352x288	564	703

Table 4: video parameter test results - bitrate parameters

While parameter set 2 is preferable to set 3, set 4 is preferable to set 2. The difference between videos sets 4 through 7 is subtle. Therefore set 4 is the best compromise between quality and size. Thus, for the resolution and framerate parameters chosen, I found a bitrate of approximately 200kb/s to be best.

4.2.4 Spacing/speed

In the current Street View application adjacent panoramas are spaced approximately 10m apart. The Stret View team determined this distance was the best compromise between granularity and speed of movement down the street. I want to similarly determine the ideal spacing between video frames. A low spacing (with constant framerate) will show images close together and thus feel smoother and allow the user more time to observe the street, but will move along the street slowly. Ideally the user would have control over the spacing (i.e. slow motion and fast forward), but because I was not able to modify the video player (see Section 4.3.3) I constrained this project to work with a predetermined value.

Changing the spacing between every two frames alters the driving speed. For instance playing images that were taken 4m apart will appear twice as fast as playing images that were taken 2m apart. Thus the

greater the distance between frames, the faster the user appears to travel down the street. A spacing of 1 represents a 1-2m distance. I tested a resolution of 352x288 at approximately 200kb/s and 4 frames per second. The spacing used in the previous tests was 4.

Spacing (m)	Observations
1	Very slow
2	Ok
3	Ok
4	Fast but tolerably so
5	Fast

Table 5: video parameter test results – spacing/speed

In the end I choose to stick with the frame rate of 4, which equates to about 6m between frames, or a observer speed of 54mph.

4.2.5 Cropping/direction/viewing angle

Ideally the video Street View application would like to give the user control over their viewing angle, direction of motion, and cropping (zoom level). Since the application is unable to modify video frames as they are playing and the player is unable to play videos backwards, I needed to predetermine each of the parameters. Each viewing angle/direction/cropping combination requires a separate preprocessed video. Thus I want to limit the number of options to as few as possible while maintaining the intended functionality.

The direction was limited to forward (along the street segment, in the direction of motion that the original imagery was taken) and backward. I decided to also limit the viewing angle to be in line with the direction of motion. So for a forward video the viewing angle was 0, and for a backward video it was 180 degrees. The original imagery was a warped panoramic view. I decided that when a user is moving in a particular direction, he or she often does not need the imagery behind the viewpoint. So I cropped imagery to show 90 degrees to the right and left of the viewing angle. This allowed the application to display the relevant portion of the imagery more prevalently. Also, when cropped to half the size of the full panorama the warping was only noticeable at the edges of the image. I considered unwarping the images to show the street in a similar manner to how the current street view application renders it, but this would have limited the field of view. Since the viewing angle was fixed, limiting the field of view would have meant that some areas would never be accessible to the user.

In summary, for each road segment the server hosts two videos: one forward and one backward, each showing half of a panorama. See Figure 5 below for what an example frame looks like.

4.2.6 Blurring

An additional measure Jiajun took to reduce file size was to blur the upper and lower regions of the image. This area contains the sky and road data and has little valuable content; therefore this blurring was acceptable

4.2.7 Final Version

Figure 5 displays an example of a single frame of our Street View video. I would like to identify some of the features to note about the image.

- The street is in the center of the frame. This is the constraint of the viewing angle mentioned before.
- This is a half panorama soothe region 'behind' the viewer is not seen.
- There is noticeable warping of the street at the edges of the frame but it does not detract from the overall viewing of the image.
- While blurring is noticeable, for the most part it does not interfere with the viewing of the region.

Figure 5 : Example Frame

In summary, I examined six parameters and determined the following optimal parameters:

- Resolution - 352x288
- Framerate – 4 fps
- Bitrate – 200 kb/s
- Frame spacing/speed – 4m / 54mph
- Cropping – front facing, half panoramic view
- Blurring – top and bottom of image

Together these produce a video with the optimal compromise between size and quality for our purposes.

4.3 Video Feasibility

This section examines the time required for downloading and playing Street View videos on a mobile device.

4.3.1 Server Requirements

Although not the focus of this project, a server was needed to provide the video data. Jiajun was able to extend the current Street View server to provide video to satisfy this need. When data for a location is requested, the server first provides a metadata file for the panorama that includes the identifiers for the related imagery. The framework used by the current mobile Street View for serving metadata files could be reused, but the metadata itself needed to be reworked to describe the video clips instead of still imagery. See section 5.3 for details on the original metadata and section 6.3 for details on the new metadata. This augmentation increased the size of the metadata to as much as 8kB per file. This is a fairly large increase from the previous 1kB per file, but the files are still small enough that serving them doesn't represent a significant performance factor (especially when compared to the 100kB video files).

Hosting video files could have significantly increased the servers' storage requirements. However, Jiajun found that with the chosen video parameters, he could serve the required videos with the same storage capacity as required for static imagery tile levels 0-2. This was determined to be acceptable since Street View was already capable of serving higher, more expensive tile levels. For the purposes of this project Jiajun only rendered videos in a limited test region, but the feasibility analysis was done for the entirety of Street View coverage.

4.3.2 Device Bandwidth

The next step in feasibility analysis was to see if the device was capable of downloading the Street View video segments in a reasonable amount of time. I obtained expected download rates from the Android emulator specifications (Emulator Network Speed) and the G1 handset specification (HTC Products). The Android G1 specification was ambiguous, stating that it uses either an 802.11b or 802.11g connection. I assumed that ours used the 802.11b protocol (IEEE Standards Association) since this aligns more closely with our measured download rates. I was able to get data on the download rate of our G1 device and the emulator for a WIFI connection, but could not obtain rates for other connections. For this experiment, my test phone did not use a real cellular service provider so I did not get empirical measurements of the 3G performance. On the emulator, a bug in the network speed emulation capped the download rate at the upload rate.

A 4 second video formatted in the manner described above is around 100kB in size and traverses about 100m worth of street imagery. The download times were calculated assuming 10% of the nominal network capacity was used, and then assuming 25%.

Table 6 shows the calculated time to obtain the video file and the results of our tests of the download speed of the G1 device and Android device emulator.

	connection type	Nominal download rate (kb/s)	measured download rate (kb/s)	Download Time 10%	Download Time 25%
emulator	GSM/CSD	14.4		555.56	222.22
emulator	HSCSD	43.2		185.19	74.07
emulator	GPRS	80.0		100.00	40.00
emulator	EDGE/EGPRS	236.8		33.78	13.51
emulator	UMTS/3G	1920.0		4.17	1.67
device	WIFI 802.11b	11000	2378	0.73	0.29
emulator	WIFI 802.11b	11000	1198	0.73	0.29
emulator	HSDPA	14400.0		0.56	0.22
	802.11g	54000		0.15	0.06

Table 6: Device bandwidth tests. Times are for the download of a 100kB file assuming 10% to 25% of the bandwidth actually delivers payload data.

In order to provide a smooth experience, the application must be able to download a video in less time than it takes to play it. The results in Table 6 show that many mobile connections are too slow to support downloading videos. The current leading carrier connection, UMTS/3G, is right on the border of viability at 1.67s-4.17s to obtain a 4 second video clip, but slower networks such as EDGE are not fast enough to support a smooth experience. Though, with the constant improvement of mobile capabilities, programmers can expect devices to reach the necessary download speeds in the near future.

I did not to take samples of Street View images sizes for comparison to the video sizes. I do know that the video tiles are 512x512, which is 33kB uncompressed. Street View needs to download level 0 and level 1 tiles (at most 5 images) for reasonable resolution. The total size of the data for 5 images is 165kB. Though this is an overestimate of the amount of data needed, it does show that the video file sizes are in the same range as the current image tiles. However, the video files cover a greater distance than the static imagery.

4.3.3 Platform Capability

Mobile devices have limited processing power. Since the platform could handle the current Street View image renderer as well as the current video player it should be able to handle the additional processing power required by a video mode. Strain on the processing power can be seen by delays in the application, which were measured in our instrumentation tests in section 7.

Video file storage must be considered in this feasibility study. The space used by the current Street View tile cache was sufficient to store several video chunks as well as the existing tiles. This space is automatically cleared by the system, but can also be cleared by the user manually. Since the video files are roughly similar to the still panorama, storage was not deemed to be a major issue.

Another consideration was that since only 3GP video was supported we needed to create a separate channel for mobile videos on the server since the desktop video channel was encoded in flash.

Additionally, the small screen and touch controls distinguish the mobile application's controls from the desktop application's controls. Recent desktop navigation improvements involving hover mouse controls and distinguishing between single and double clicks aren't feasible on a touchscreen/trackball interface.

There are two aspects we hoped to gain from introducing video to the Street View application. The first is improved data transmission. Decoded video frames are no different from the image tiles Street View currently downloads and renders. So if the application could get access to the decoded video frames as bitmap images (which the video player must produce to play the video) then it could render them in its custom framework. This application of video would be entirely transparent to the user. Unfortunately developers cannot currently use the video decoder in isolation and so I could not implement and test this. It was beyond the scope of this project to make the changes necessary to the video player to implement this feature.

The second is improved navigation controls. This application, while hampered by the lack of access to decoded frames, could still be implemented in a limited form using the existing video player. Without access to the decoded frames the application would need to play the videos as-is therefore we had to do more processing on the server side. For each street segment the server had to host an additional video for the reverse direction since we couldn't play the videos backwards. Since the video player couldn't adjust (shift, warp, or crop) the video in any way we constrained the user to a single vantage. This limitation turned up in places I hadn't expected. For instance, we used the positions of two adjacent panoramas to calculate direction of the street, which we used to crop the videos. But occasionally there were single frame videos. For these videos the direction of motion is not well defined so we couldn't determine beforehand the proper cropping to use. Thus our prototype had to ignore all single frame videos. So while constraints by the video player meant we had to be more cautious, the overall idea could still be implemented.

4.4 Conclusions

This section analyzed the plausibility of using video first by determining how the video should be formatted, and then by considering current server, device, and platform display capabilities. I found a suitable combination of parameters for viewing Street View video on the G1 while keeping the file size small. Jiajun determined that Street View could serve video data with the existing technologies. On the client side, download capabilities are on the border of viable, meaning that devices need faster connections to make video viable. Lastly, I found it feasible to implement video in the mobile Street View application to test its potential for improvements in navigation controls and reduced latency.

5 Mobile Street View Application Design

One of the goals of my thesis research was to review the design of Mobile Street View and to analyze its performance. This required me to reverse engineer the design from the source code because the design was not documented when the initial work was done to port the application to the Android environment. Google also wanted to measure the performance of MSV in accordance with its internal policies. Google has a standard package used to instrument source code and collect performance

metrics. This section describes the work done to understand the operation and design of the existing Mobile Street View Application.

5.1 Requirements

The goal of the Street View application is to display the imagery of a street location to the user and to allow the user to navigate to adjacent locations. To achieve this, the application must meet some basic display and interaction requirements. In the display, it must show the user the street imagery as well as the provide navigation controls to move to adjacent panoramas. The image display is done through a custom renderer that requests data from the imagery server, manipulates it, and renders the necessary image tiles. The navigation controls are accomplished by displaying navigation arrows as an overlay on top of the imagery. Each arrow represents an option to navigate to an adjacent panorama. Figure 6 shows one example of how this appears to the user.

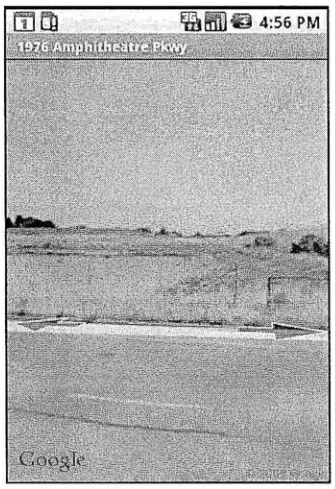

Figure 6 : example Street View display

The second requirement of the application is to allow the user to interact with the panorama. The main class of the application, StreetView, handles all user interactions. There are two types of interactions. The first is panorama interactions such as panning and zooming. These events can be triggered by touch, trackball, or menu. They are processed in StreetView and the resulting viewpoint changes are forwarded to the renderer to be handled. The second types of interactions are those that alter the location. StreetView also interprets the user action to determine the next panorama to load. It handles the fetching of the new data, and then asks the renderer to refresh with the new location.

These summarize the core requirements that need to be met for the application to function as intended.

5.2 Usage

Figure 7 diagrams an example for how a street might be traversed using Street View.

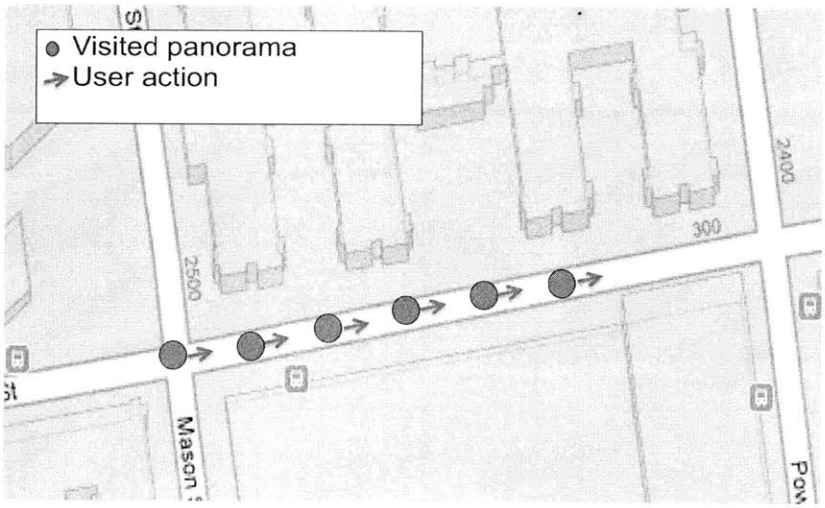

Figure 7 : bird's eye diagram depicting how a user navigates down a road in the existing Street View application.

It gives a sense of the actions that must be taken by a user to browse along the street. Each visited panorama, indicated in red, shows a display akin to the one shown in Figure 6. Each arrow indicates that the user has selected one of the overlay arrows. For each location, the user must wait for the panorama to load, so he can see that browsing an entire street incurs many data request and image processing delays. Our goal in section 7.2 is to measure these delays and recommend modifications to improve the performance.

5.3 Server Communication

This section describes how the data needed by the client is obtained from the server.

To view a given location, the application first requests the **panorama configuration** from the server. It can specify which panorama it wants by providing a latitude/longitude or a panorama ID. In return the application gets an XML document like the one shown in Figure 8 below. This XML contains all the metadata for the requested panorama.

```
<panorama>
    <data_properties image_width="3328" image_height="1664"
      tile_width="512" tile_height="512" pano_id="0pE1VEM2x4AODWtEiJEfUQ"
      num_zoom_levels="3" lat="37.808929" lng="-122.414931" >
        <copyright>Â© 2009 Google</copyright>
        <text>The Embarcadero</text>
        <street_range>2004</street_range>
        <region>San Francisco, CA</region>
        <country>United States</country>
    </data_properties>
    <projection_properties projection_type="spherical"
      pano_yaw_deg="274.62" tilt_yaw_deg="51.52" tilt_pitch_deg="0.86" />
    <annotation_properties>
        <link yaw_deg="277.96" pano_id="HPM5DBzWg51pAxLNLBDagw"
          road_argb="0x80ffffff" >
            <link_text>The Embarcadero</link_text>
        </link>
        <link yaw_deg="96.52" pano_id="F1SMFkVnc0XTbKdbU5LVeA"
          road_argb="0x80ffffff" >
            <link_text>The Embarcadero</link_text>
        </link>
    </annotation_properties>
</panorama>
```

Figure 8 : Example panorama configuration. Significant elements are bold.

The most important piece of information is the pano_id, or **panorama ID**. This string is a unique identifier for this panorama location, and is needed to request image tiles. Image tiles are requested by providing a panorama ID, zoom level, and x and y values. The client sends a separate request for each tile, so that no resources are wasted downloading unnecessary data. The tile requests are made asynchronously after the panorama metadata has been received.

The **link** data in the configuration file is also important. Links represent a connection, or 'link', to adjacent panoramas. Traditionally this information is displayed to the user in the form of a street overlay with a clickable arrow. When the user clicks on the arrow, they are moved to the adjacent panorama, specified by the direction and pano_id field within the link. Figure 9 shows a graphical summary of the link data available in the panorama configuration.

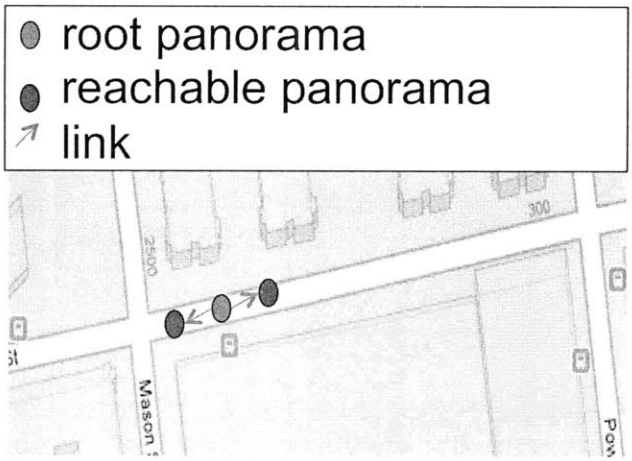

Figure 9 : diagram of the link data contained in a panorama configuration file.

5.4 Software Structure

This section briefly describes the Street View application's classes and their purposes. At a high level the application is rather simple. For each location, the application needs to fetch the configuration file for the desired location. This gives it the data necessary to render the overlays as well as telling it which image tiles to fetch. Once the tiles are fetched the client needs to organize and unwarp them. Finally, it renders the imagery and overlays and handles any user interactions.

5.4.1 Main Classes

- Street – handles the activity related events such as menus, pausing the application, and status text. It is the main Android activity and where the application is entered. It contains a StreetView object, which it uses to handle the more application specific events.
- StreetView - handles the main flow of actions. Its main role is to interpret user actions and convert them into actions to be handled by the Renderer or other classes. This class also maintains and updates the configuration data for the current panorama location. While it handles touch events directly, it uses TrackballGestureDetector to help interpret trackball events.
- Renderer – handles the display of data to the user. It knows how to fetch tiles, and manages them using an ImageScoreboard.

5.4.2 Drawing

The classes responsible for display of data to the user are:

- ImageScoreboard, TextureCache – handles the caching/fetching of tiles,
- Sphere, Grid, GeometryDrawer, LevelInfo –handle the placement and warping of tiles for rendering,
- Overlay, LabelMaker – responsible for the street, arrow, and street label overlays. These also determine whether a given touch action activates the 'move' or 'zoom' events, and
- GLMatrixWrapper, MatrixStack – wrappers for GL.

5.4.3 Threading

Much of the work of the application is done asynchronously through message passing. Both the Renderer and StreetView classes use messaging extensively.

- AsynchRunner – runs workers from a RequestQueue asynchronously.
- RequestQueue – a queue of requests. Also allows for the requester to check if a request is already in the queue.
- RequestPanoramaConfiguration – an object that knows how to request a panorama configuration. Also has a hashcode. (The Renderer knows how to request individual tiles given the panorama configuration)

5.4.4 Abstract Data Types

- PanoramaConfig – stores the panorama configuration file. The most important fields are mPanoId (the ID of this panorama) and mLinks (a list of panoramas reachable). This class also knows how to parse the XML panorama configuration file.
- PanoramaLink – stores the data relevant to linking to an adjacent panorama
- PanoramaImageKey – a hash key for a panorama image
- LevelInfo – information about the image and zoom level necessary to for the base grid.
- MapPoint – represents a latitude/longitude.
- UserOrientation – represents the user's direction. This determines the user's field of view.

5.4.5 Caching / Storage

There are essentially only two types of data that need to be stored: Panorama configuration files and image tiles. Because they are used frequently and are smaller, configuration files are kept in an in memory cache. All data requested is kept in the HttpCache. The Android system takes care of clearing out the data cache when it gets too large.

- HttpCache – a cache of URLs to filenames. Fetches the data from the server if not already in the cache.
- PanoramaConfigCache – an in memory cache of the panorama configuration files.
- TextureCache –an in memory cache of tiles.

5.4.6 Misc

- StreetUrl – a set of static strings needed to request data from the server.

This section looked into the design of the existing mobile Street View application. First it considered at a high level the application requirements and how they were met. Then it described the data provided by the server and how to request it. Finally, it summarized the software structure and described the principal purposes of the classes.

6 Video Street View Prototype

Section 5 detailed the breakdown of the existing mobile Street View application. This section describes the requirements, design, and implementation of the Street View video prototype I built. Jiajun designed and implemented the server side video support (including adding panorama metadata and video chunks), and I handled the client side support.

6.1 Requirements

The goal of the video prototype is to display video imagery of a street and to allow the user to navigate through several locations by only specifying a direction.

A video prototype has some additional requirements that must be met. On the display side, in addition to the imagery and overlays, the application needs to play video. I left the custom renderer in place to handle the tile imagery. I altered the overlays to display the video link data instead of normal links. Then I added a video mode to handle the requirement to display video clips. After obtaining the necessary data, the video mode calls an activity dedicated to playing the videos. Unlike the renderer, the video player displays the data it receives as-is. Thus the street is displayed warped and with a fixed viewing angle. The shift from the unwarped imagery to the warped video imagery is hidden by a brief blanking of the screen. Additionally, before shifting into video mode the client aligns the user's vantage with the video's start vantage to avoid confusion caused by a swift change in viewing angle.

The second requirement of the application is that it should allow the user to interact with the panorama by only specifying a direction of motion. Again, the main class StreetView handles all user interactions. While panning and zooming the imagery was done in the same manner as before, the interaction indicating movement (selection of the arrow overlay) was directed towards the video handler rather than the renderer. The handler then takes care of obtaining and displaying the relevant video chunks. Once the video begins playing, the application does not allow any user interaction until an intersection is reached, or the 100m limit is reached. This was done because of time constraints; the player could have handled pausing/stopping of the video at any time. When video playback is complete the user is dropped back into the normal Street View at the terminating location.

These summarize the core requirements that need to be met for the application to function as intended and to allow us to validate the video analysis in section *3*.

6.2 Usage

Figure 10 illustrates how a street might be traversed using our video mode. Notice that in comparison to the previous diagram, Figure 7, there are many fewer user actions required. Also, delays while waiting for imagery/footage to download only occur when entering and exiting the visited panoramas. Thus, this design would potentially provide a much smoother experience.

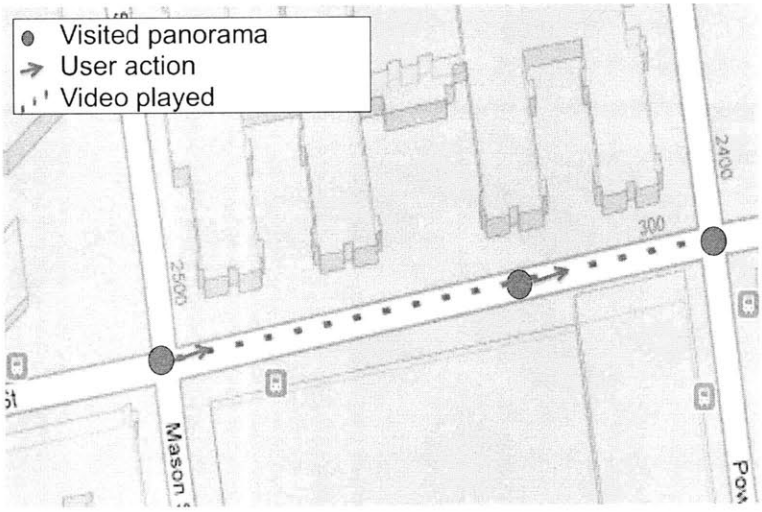

Figure 10 : birds eye diagram depicting how a user navigates down a road in the *video* Street View application.

6.3 Server Communication

This section describes how the data needed by the client is obtained from the server for Street View with video incorporated.

To view a location the first thing the application needs is the panorama configuration. Jiajun augmented the configuration metadata with information necessary to retrieve and play the relevant video chunks. Requesting the panorama configuration from the server is the same as before, except the requester needs to set a flag to request the additional video data. Conceptually, the video link data is represented as a tree. The current panorama is the root of the tree, and each node in the tree is a reachable panorama. Each non-root node knows its parent node, and how to transition from its parent via video chunks. Figure 11 illustrates the video link data contained in the panorama configuration.

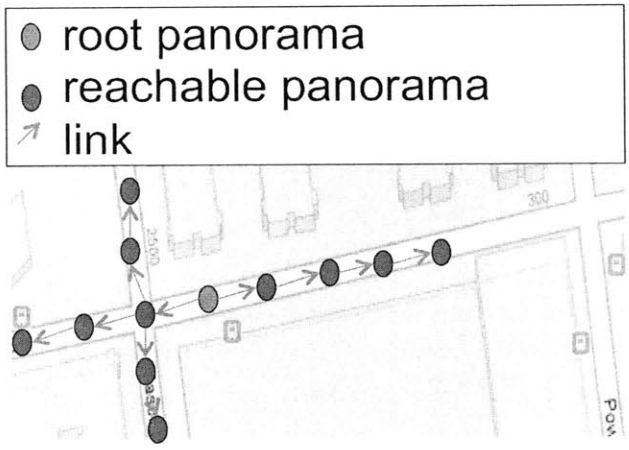

Figure 11 : diagram of the video link data contained in the new panorama configuration file.

The transition from one node to another, or **video link**, consists of a list of video chunks and the start and end frames for each. Given this information, the developer can backtrack from any node to the root to determine the entire video sequence that needs to be played to reach that panorama.

Figure 12 shows an example of how the configuration file actually incorporates these additions.

```
<panorama>
    <data_properties>
    … Same as Figure 8…
    </annotation_properties>
    <model>
       <transitions>
          <chunks>
             <chunk id="WBW57iB3f7cr4FMZhezyxQ"/>
             <chunk id="2bLK58BtUo-wNs4fSt3wjw"/>
             <chunk id="kkC0ZRqCUlrSocasHJEDvw"/>
          </chunks>
          <panos>
             <pano id="F1SMFkVnc0XTbKdbU5LVeA" lat="37.808919" lng="-
             122.414817">
                <sequence chunk="1" begin="36">
                   <frame yaw="-85.53"/>
                   <frame yaw="-85.60"/>
                </sequence>
             </pano>
             <pano id="OADG8m8n5reObVaQX73H2Q" lat="37.808911" lng="-
             122.414674" prev="0">
                <sequence chunk="1" begin="41">
                   <frame yaw="-85.60"/>
                   <frame yaw="-85.51"/>
                   <frame yaw="-84.96"/>
                </sequence>
                <sequence chunk="2" begin="15">
                   <frame yaw="-85.60"/>
                </sequence>
             </pano>
             …
             <pano> … </pano>
          </panos>
       </transitions>
    </model>
</panorama>
```

Figure 12 : Example of panorama configuration with video transition metadata added.

The "chunks" section contains a zero indexed list of all the video chunks, and their unique ID strings, that appear in video link tree. In the remainder of the configuration the video chunks are referred to by the order in which they appear in the list. For instance, in the example configuration provided, chunk 0 refers to chunk ID `WBW57iB3f7cr4FMZhezyxQ`. Chunks are requested from the server using the chunk ID.

The <panos> element contains a zero indexed list of all panoramas (nodes) reachable from the root. The node contains its panorama ID and location. The tree structure described before is encoded by the "prev" attribute, which refers to the list index of the parent panorama. If no "prev" field is specified then the panorama's parent is the root panorama. So in the example above the first panorama (`F1SMFkVnc0XTbKdbU5LVeA`) is a child of the root, and the second (`OADG8m8n5reObVaQX73H2Q`) is a child of the first. The **sequence** field contains the data relevant to video chunk playback. Each sequence

pertains to the playing of a single chunk, so a node may have several sequences. The sequence field contains the index of the chunk to play, as well as the frame to begin playing at. Afterwards is a list of frame yaws, the length of this list indicates how many frames to play before stopping playback. The yaw indicates the difference between the view direction and due north. This information would have been necessary if the prototype wanted to support the ability to have the viewing angle be independent of the direction of motion. Since the video prototype did not support that feature, it ignored the yaw angle.

After these modifications, the server provides three types of file for download: augmented configuration files, image tiles, and video chunks.

6.4 Software Structure

Adding video to the existing application is simple conceptually. Previously when the application detected a user had activated the arrow, it used the PanoramaLink information to move to the adjacent panorama and reset the renderer. Now, the application uses the video link information instead.

I altered the overlays so that instead of displaying the normal panorama links, it displays the adjacent video links (the children of the root panorama). Thus by clicking on a particular arrow, the user has indicated the panorama that is in the direction he would like to travel in. To determine which chunk sequences to play, I can follow that branch of the tree until a node has either no children (the metadata has reached the 100m limit) or multiple children (it's reached an intersection). That list of sequences is preprocessed to consolidate sequences and eliminate single frame chunks. Then the application requests the video chunks from the cache. This list of sequences is sent to the video player (VideoActivity) and displayed. When complete, the user is returned to the normal Street View application at the final position and the application awaits further user input.

6.4.1 Design decisions

- I placed the video player in its own Activity. This ensured that the video player didn't interfere with the existing Street View renderer in any way. Unfortunately it made passing data to and from the player much more difficult and expensive since all the data had to be passed through intents. This difficulty occurs because in Android class data is not readily shared between Activities so must be parceled and passed through intents – which is expensive.
- The existing Android MediaPlayer did not support seeking by frame so the application had to convert frame counts into times. The player also did not support playback for a fixed time period so Is had to set a separate timer that would stop playback when triggered. Thus, the start and stop point of our video playback was approximate.
- Videos are downloaded synchronously for simplicity. I wanted the video player not have to worry about whether a video had been downloaded or not, so I ensured that all the videos were obtained before calling the video player. This designed could be improved by using asynchronous fetch requests for future video chunks while playing the current one.
- The preprocessing of the video sequence serves mainly to filter out single frame videos, which the video player cannot handle because of their ill-defined directions. The secondary purpose of

this processing was to combine two continuous sequences such as "chunk 1 frames 10-15" "chunk 1 frames 15 – 20" into "chunk 1 frames 10-20"

- In addition to playing until the next fork in the tree, the video applcation also support playing to any node in the tree. Though not used in the final prototype this feature helped to test intermediate versions.

6.4.2 Modified classes

- Street – Because this is the main Activity for StreetView, using this class is the only way to determine whether the VideoActivity has terminated. The Street class needs to pass that information back to StreetView to let it know that video playback has completed.
- StreetView – I overrode the "move" command to "play" which calls VideoMode.
- Overlay – I altered the overlays to display the direct children of the root panorama in the video link tree, rather than the normal panorama links.
- PanoramaConfig – I needed to add support to the configuration file for parsing and storing the new video link data.

6.4.3 Added classes

- VideoMode –responsible for parsing the video link tree to determine which video chunks to play. It uses a VideoChunkCache to get the required videos. It actually has two modes: play to fork and play to node, though I only use the former in the demo.
- VideoActivity –responsible for the actual playing of video chunks.
- VideoChunkCache - a cache of video chunks. Because the HttpCache already provides a layer of caching, the main purpose of this class was to provide a cache capable of providing the full filepath, which the MediaPlayer requires.

6.4.4 New Data Structures

- VideoPanoNode – a data structure that stores a node from the video link tree. This is the video equivalent of PanoramaLink.
- Sequence – A data structure to store the information in a "sequence" field. Instances of this class are kept in a VideoPanoNode.

I was able to successfully construct a working version of this video Street View prototype.

This section described the operation of the Street View application augmented with video. It considered at a high level the new application requirements and how I decided to address them. Then it described the changes to the configuration file to encode video link data. Finally, it summarized the design decisions, software structure, and principle purposes of the classes.

The video prototype verified that mobile Street View could provide video with an acceptable display quality. The analysis in section 3 showed that the size of the videos is actually smaller than the still imagery and that the data transmission requirements fall within the capabilities of the devices and the networks currently available.

7 Instrumentation

This section addresses the goal to determine quantitatively the performance of both the existing Street View application and our video prototype.

To determine how an application is performing, developers must first instrument it. **Instrumentation** is the process of adding timing measurements to an application. In this work I perform client-side instrumentation, which means I only take performance measurements in the client application. Almost all applications at Google perform some form of instrumentation, hence the need for a Client Side Instrumentation (CSI) team. Since most applications are web-based, the tools for collecting instrumentation measurements are able to accept remote reports through HTTP requests. Because my instrumentation is solely on the client side, I do not get information about how our servers are performing.

For my experiments I use a previously developed Google Client Side Instrumentation (CSI) aggregator. It provided a simple reporting interface via HTTP and handled the collection and storage of reported measurements. CSI also provides tools to aid visualization and analysis of instrumentation measurements. The standard Google CSI library had support for recording variables, actions, and experiments. **Variables** are the specific timing values that we want to measure. **Actions** represent a grouping of variables related to particular sequence of events. And **experiments** organize related results. For instance, instrumenters may group measures into experiments based on what version of an application the user is running.

I implemented a simple Timer class to handle the management of instrumentation values and report them to the CSI server. Each Timer manages the variables for a single action. The Timer measures the time elapsed between the start() and end() calls for each variable. The Timer was also responsible for generating and sending report messages to the CSI server when requested. To allow for non-time related variables, the Timer class also supported the ability to manually set the values associated with a given string. The string values for variable names and server URLs was kept in the utility class CSINames. Also, for code simplicity, if a variable was repeatedly measured it selected the first (or shortest) time.

7.1 Definition of Terms

7.1.1 Actions

I defined 3 actions for our experiments. The third action was only used for the video experiment.

- entrance - This action's variables measure the timing for the initial entrance into Street View. The user's initial entrance is when he is in maps, launchs Street View and views the first panorama. It is important to differentiate the initial viewing from subsequent viewings since the initial timings often incur setup costs due to the initialization of objects such as the Cache or the OpenGL renderer. Furthermore, some code paths are only followed during entrance.
- next - This action is for the viewing of any panorama other than the initial panorama.
- video -This action is for the viewing of a sequence of videos along a street.

7.1.2 Variables

My instrumentation variables fell into three categories: (1) those measured as a time from the start of the variable's action, (2) round trip times to download data from a server, or (3) the number of occurrences of some event. Categories (1) and (2) are measured in seconds.

The start times for the actions are as follows:

- (for action entrance) the start of the SV client
- (for action next) the time when the user clicked an arrow to move to the next panorama
- (for action video) the time when the user clicked an arrow to start playing videos.

I omit some of the less relevant measurements from this section. To see the full list please refer to Section 11.1.

Entrance action only:

- uent (1) - the time between a user requesting to enter SV and the SV client to start up.
- ucnt (3) – a count of the number of panoramas viewed before exiting the SV client

General:

- sc (2) - the round trip time to receive the panorama configuration file
- sl (2) - the round trip time to receive an image tile
- prt (1) – page ready time, the time a panorama is ready to be interacted with, i.e. when the road overlays are first rendered
- pst (1) – page stable time, the time a panorama is fully loaded, i.e. the screen is stable until the users interacts with it
- unxt (1) - the time a user clicks on an arrow to step to another panorama

Video action only:

- sv (2) - the round trip time to receive a video chunk
- vcnt (3) - a count of the number of video chunks played during this video session
- fcnt (3) - a count of the number of video frames played during this video session
- pcnt (3) - a count of the number of live panoramas passed during this video session (including the starting panorama)
- vrt (1) - the time videos begin playing
- vst (1) - the time videos complete playing
- gap (1) - the time between videos where the screen goes black

7.2 Original Street View Application

I conducted the first experiment to get a sense of the existing Mobile Street View's timing.

7.2.1 Instrumentation Locations

This section details where and how instrumentation was inserted for each of the measured variables. It was sometimes difficult to find the exact location where a particular variable's event began or ended, so some of the values are approximate. I also tried to keep the measurements within as few classes as possible to avoid passing the Timer objects as much. I took measurements with the cache activated.

Street View Entrance (uent) – Because this variable spans two applications (Maps and StreetView) I had to add instrumentation into the Maps application to handle it. When the user requests to enter Street View a timestamp is taken. This timestamp was then passed through the intent to start the Street View application. I also had to ensure that the timer would measure all variables in the 'entrance' action relative to this provided timestamp. I deemed that the application was entered once the onCreate method was completed. (See Street->onCreate())

Configuration Request (sc) – I used to StreetView status updates to determine the roundtrip time to get the configuration from the server. I started the Timer before any panorama configuration request was sent to the queue and ended it when the PanoramaConfig object was no longer null. (see StreetView->getStatus(), checkStatus())

Tile Request (sl) – I were able to isolate the exact request to the server and timed around it for this variable. (See Renderer->ImageFetcher->fetch())

Overlay Rendered (prt) – I used the existing render status updates to determine when the overlay was finished rendering. When a panorama's status passed 0 something must have been rendered, and since the first thing rendered is the overlay I used this as our time. (See Renderer->draw())

Imagery Rendered (pst) – I also used the render status updates for the page stable time. (See Renderer->draw())

Next Action (unxt) – I declared the point where the user requests the next panorama to be after the touch event was processed, but before the request for the next panorama configuration was sent. This is also when the current action finishes, results are reported to the server, and the next action's Timer is initialized. (see StreetView->goto())

Panoramas Visited Count (ucnt) – this variable was reported under entrance action, even though it did not truly belonged to any of the actions. The count had to be kept independent of the timer because Timer objects were reset with each new panorama/action. I simply kept a count within Street View and incremented it each time the user requested to move to a new panorama. Its value was reported whenever the application's onDestroy() method was called. (See StreetView->goto())

7.2.2 Experiment

For this experiment, eight users installed an instrumented version of MSV which reported to a centralized CSI server to collect the results. Users were asked to interact with the application for 15 minutes a day for 3 consecutive days while connected to Google WIFI. Users were allowed to use the

application in any way they wished, but were also provided with usage suggestions such as finding a new restaurant, exploring a famous landmark, or looking at the street where they live on.

In total we received about 25 reports of the action 'entrance' and 100 reports of the action 'next'.

7.2.3 Results

The purposes of this instrumentation of the Street View application were to

1. Quantify the user-visible delays
2. Determine the source of client side delays where possible
3. Quantify aspects of user/application interactions

Section 3 described the application flow as observed by the user, particularly in Figure 2 (reproduced below as Figure 13). Figure 14 shows a timeline associated with each of the events in the application flow.

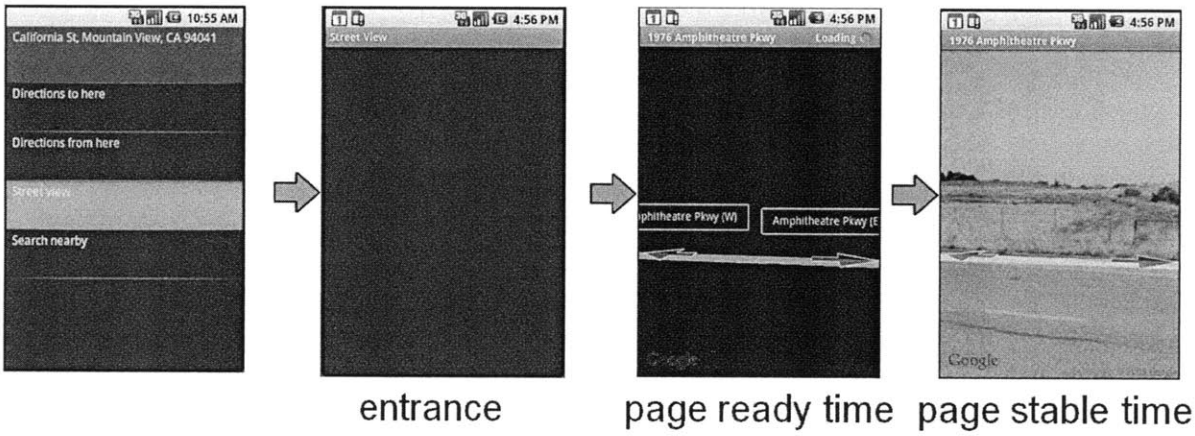

Figure 13: Screenshots depicting an example Street View session's stages.

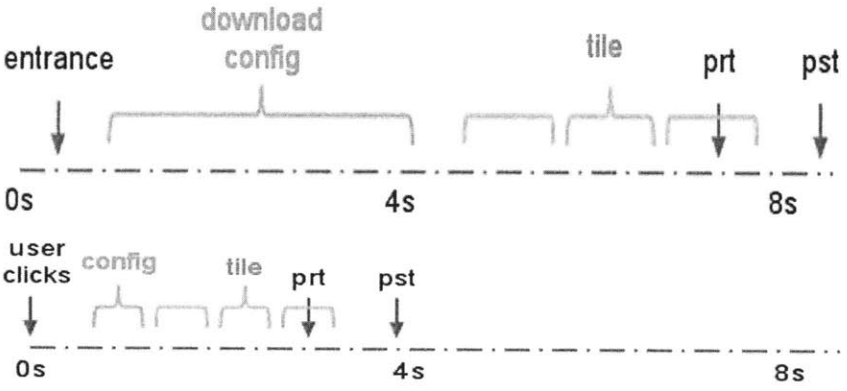

Figure 14 : A timeline of the application flow.

The top timeline is for the 'entrance' action and the bottom for the 'next' action. The configuration and tile times are to scale, but placed arbitrarily, since we only obtained the round trip times. Also remember that the tiles are requested asynchronously. The choice of 3 tiles was arbitrary. The times used were average measurements of the variables.

First to consider is the transition from the Maps application to the Street View application as indicated by the text "Street View" in the status bar in the second frame of Figure 14. The typical (median) wait was 0.6 seconds, but occasionally users saw much larger delays. I am unsure what caused the larger delays, but I presume that normal delays were composed of application launch costs. For the remaining variables in the entrance action, I use the median values since the outlier entrance times affect their measurement (excluding the variables measuring as round trip times).

Every event during the 'entrance' action is potentially affected by the application's startup costs and so I would expect the times to be larger than for the 'next' action. I expect that the first connection with the server in particular may take much longer than subsequent connections because of the DNS lookup. Additionally, later events relying on HTTP requests may hit cached values, significantly speeding them up. I do not believe the latter explanation is sufficient to explain the timing differences. Caching alone would not improve the times for when a cache miss occurs, so I would still expect to see some comparably large measurements for configuration/tile downloads in the next action. Since these measurements are missing, I know some initialization costs are also affecting the entrance server request times.

The next event visible to the user is the display of the overlays and then the display of the images. There is a large difference between actions for the timing for these variables. Most of this difference can be accounted for by the variations in download times. Note that not all of the delay in the rendering of the overlay is due to the downloading of the configuration file.

In conclusion, I found that the wait time between panoramas was 3.5 seconds with most users experiencing between 2.5s and 6s delays. While this was less than expected, it is important to remember that the experiment was performed on WIFI, which is the fastest available connection. However, the wait time for the first panorama was almost double that at 6.5 seconds. The variance was very large with most users experiencing between 4.5s and 13.5s delays. The download times are significant, but do not explain all of the delays. These results show that there is significant room for improvement, especially in the entrance action.

I also took some data on user behavior in addition to information about the application. Users usually view only one or two panoramas in a session. The data shows that users typically waited until the panorama was fully loaded before beginning interaction and they delayed even longer before requesting the next panorama. This means that the experience for moving to an adjacent panorama could be improved by prefetching.

7.3 Video Street View Application

The second experiment aimed to measure latencies in our video prototype, and compare its performance to the performance of the existing application.

7.3.1 Instrumentation Locations

In addition to the measurements detailed previously, I also added some new instrumentation specific to video mode.

Chunk Request (sv) - I measured how long it took to obtain video chunks when there was a cache miss. (See VideoChunkCache->get())

Video, Frame, and Panorama Count (vcnt, fcnt, pcnt) – I determined these parameters during the additional processing of the sequences before chunk requests were sent to the server. (see VideoMode->getSequences(), playtoFork())

Video Begins Playing (vrt) - I timed this at just after the intent was launched to avoid passing the timer into the VideoActivity. (See VideoMode->playSequences())

Video Completes Playing (vst) – I measured this at the point where the application reenters the original Street View mode to avoid passing the timer around. (See StreetView->finishVideoLink())

7.3.2 Experiment

The second experiment aims to measure the performance of the Mobile Street View application augmented with video features in order to compare its performance to the existing application's performance. This experiment was less rigorous due to time constraints on the project. I only tabulated result from one user on the emulator. I could not use the G1 device because of issues reaching the test server from a wireless connection. I am also careful not to compare values between experiments since the setup for this experiment was fundamentally different from the first experiment.

Our test of one user on the emulator produced about 45 entrance reports, 75 next reports, and 100 video reports.

7.3.3 Results

The purpose of this instrumentation of the video Street View application was to

1. Quantify the user-visible delays
2. Determine, where possible, the source of client side delays
3. Quantify aspects of user / application interactions

I was concerned with the performance of video Street View in comparison with the traditional Street View application. Since this experiment was conducted on the emulator it is not comparable to the previous experiment, I collected data from all the previous variables in addition to the added video variables.

Section 3 described the altered application flow as observed by the user. Figure 15 depicts the flow of the video application after a user has chosen to step, and Figure 16 shows a timeline associated with each of the events in the application flow.

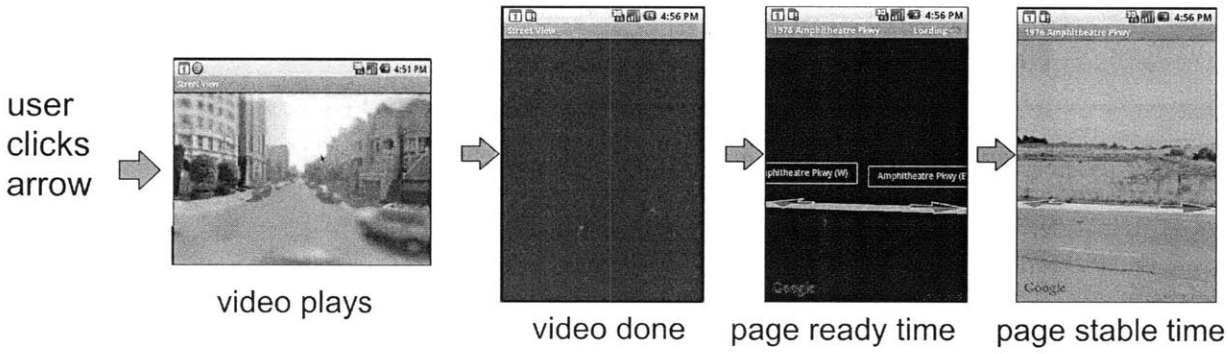

Figure 15 : Screenshot depicting stages in an example session for Street View with video.

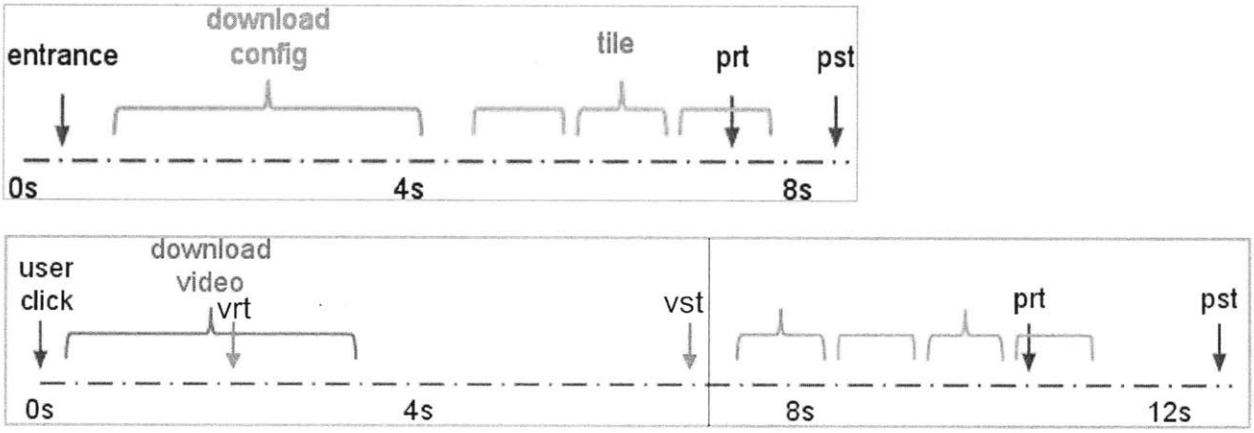

Figure 16: A timeline of the video application flow.

The top timeline is for the 'entrance' action and the bottom for the 'video' (left) and 'next' (right) action. The configuration and tile/chunk download times are to scale, but placed arbitrarily, since we only obtained the round trip times. Also remember that the tiles are requested asynchronously. The choice of 3 tiles was arbitrary.

The times measured in this experiment should be close to those in the last experiement since the entrance and next actions have not been changed. They may differ due to the difference in the two platforms, but I expect that at least the general trends remain the same. For the most part the emulator used in the second experiment produced slower times than the device.

One interesting observation to note is that the average time to download a chunk is less than the average time until video starts playing. This can be explained by a high hit rate on the video cache since in the 100 video reports, only 18 chunks were requested.

I measure download times of on average 3 seconds for a 5 second video. This means it is possible for the downloading to keep up with the playing of the videos. Thus the application could continue along a street indefinitely with little to no interruption.

Ultimately Street View is concerned with displaying data to the user. I thought that there were two important metrics to consider: the time it takes to display a panorama to the user and the fraction of the overall time that imagery or overlays are visible.

I want to compare how long it takes to view a street segment using video to how long it takes to view a street segment using traditional imagery. By measuring the number of live panoramas hit during each video session I can calculate this. Users hit on average 6 live panoramas, including the start panorama. Assuming a cache miss on retrieving the chunk, it takes the application 3 seconds to download the chunk. Then the video player takes another 5 seconds to play the chunk. The application takes 6 seconds to load a panorama in the original mode. So the time per panorama is about 1.5 seconds for viewing a street with a cache miss. Thus the video prototype gets a 4x improvement over the original Street View application.

This is a conservative estimate of the benefits for a number of reasons. First, I ignore the fact that users wait several seconds after a panorama is loaded to move on to the next panorama. So by automatically playing to the next panorama the application saves time. I assumed a cache miss, while the evidence suggests that often the application get cache hits. Also, I used the average live panorama count; the median count was closer to what I would expect given the frame count and spacing chosen. With some changes to the implementation, I could potentially drop most of the 6 seconds needed to load the end panorama by having the configuration and tiles fetched while the video is playing. Additionally, the video chunk fetching and playing was done synchronously for simplicity. The application could gain in performance by also making them asynchronous.

Another important metric to consider is the fraction of time that useful information is being displayed to the user. The time between vrt and vst is all time spent playing the video, whereas in the time before prt the screen is blank. There are occasionally some blank screens between video chunks, called gaps, but since most video sessions played only a single chunk these did not occur often. Of the 7(average case) or 9(cache miss) seconds spent in video, on average 5 seconds are spent playing video. For the original application, only 2 of 6 seconds are displaying overlay information and no imagery is seen unless the user waits longer. Through video a lot more time is spent actively displaying new data to the user.

So for viewing a sequence of locations, my video prototype gets a 4x improvement in performance over the original application while increasing the fraction of useful screen time.

This section described our instrumentation of the Street View application. I found that latencies for the first panorama were significant, and delays during the next action were lower than expected. For the video prototype I found significant improvement in both latencies and useful screen time. Specifically, I found that the time to fetch video chunks was less than the time to display them.

8 Related and Future Work

8.1 Related work

A lot of work has been done to bring video to mobile platforms. Low bit video encodings such as H.263 (Rijkse) and H.264 (T.Wiegand) have been developed which work better for limited bandwidth circumstances found on mobile devices (T. Stockhammer). New protocols for data transfer over mobile networks such as EDGE, 3G, and HSDPA (3GPP specification) have been developed to increase mobile capabilities. In addition to video formats and communications protocols, new mobile software platforms (Oliver) have also been rapidly emerging; such as the Android OS (Android) used in this project.

Additionally Street View is constantly revisiting ways to improve the application. One recently released feature that improved usage is click-to-go for the desktop application. It presents locations for the user to navigate to through additional mouse-overs. Thus a user can click to go to any location visible in the current panorama or zoom in on a façade. There has also been work to make the transitional stage of click to go (after the user has chosen a new location but before they have arrived) use video segments to smooth the transition. The Google Maps for Mobile team is unifying the code bases Maps and Street View for various mobile platforms to facilitate uniform feature and update availability across platforms. Other teams are also looking into ways to use the existing Street View data in new ways. For instance, the newly launched turn-by-turn directions has an option for seeing Street View imagery of a location while driving through it.

The Client Side Instrumentation team at Google is constantly providing new ways to collect and analyze instrumentation data. The desktop version of Street View was also instrumented during the same time period as this project.

8.2 Video formats and Mobile Viability

I was constrained to using only 3GP video file format for our tests. A full assessment would consider other video encodings to determine which format compresses Street View videos best. Future researchers should consider the Flash format in particular since Street View already serves Flash format for the desktop application. Serving only a single video channel would require fewer resources to support the same functionality. Additionally, my analysis of network connections was entirely theoretical. It would be worthwhile to test actual download rates on various connections using 3G protocols. Lastly, one should test other mobile devices and platforms for a fuller assessment.

8.3 Mobile Performance Experiments

My measurement of the application performance covered a small scope of the potential interest areas. The measurements we obtained, while useful for getting a general sense of the application, raised a number of questions that could be answered by finer grained instrumentation. Exactly how do the server response times relate to user visible latency? What are the initialization costs for each of the Street View components?

It would be worthwhile to obtain instrumentation results on lower speed connections such as EDGE or 3G to get a sense of how the application is performing in the field. Additionally, the video prototype

instrumentation was time constrained and so reported only on a single user's experience on the emulator. Once the hurdles to overcoming access to the server from the device have been overcome, a multiuser on device experiment should be performed. This experiment would also provide more feedback about the changes to the user experience.

8.4 Enhance Existing Street View Application

Several shortcomings were identified in the current Mobile Street View application. I found that the entrance to the application and the display of the first location is the area with the largest latencies. One major improvement would be to utilize asynchronous data requests and pre-fetch techniques to avoid long periods of blank screen and jumpy transitions. Pre-fetch techniques should focus particularly on obtaining configuration files in advance.

8.5 Incorporate Video into Street View

I would like to see video integrated seamlessly with still imagery into a single renderer. I was unable to do this because the media player lacked the necessary features and I did not have access to the existing video decoder. Once developers are able to decode video frames they can use the existing renderer to manipulate and display video frames in the same manner Street View does for still imagery. This would enable the application developers to allow the user fine-grained control over the experience including such features as: variable viewing angle, adjustable observer speed, and zoom while paused. I anticipate that, even with improved video players, developers will need to use our custom renderer because of the very Street View-specific needs such as unwarping panoramic views and merging still and video imagery.

Though most of the limitations were in the renderer, there are other aspects of the prototype that can be improved. I used a greatly simplified fetcher to acquire video chunks; with a more sophisticated asynchronous fetcher one could decrease wait times. Also, currently when the application runs out of video link data in the direction the user is travelling the video mode stops the player. A more reasonable behavior would be for the application to fetch the next configuration file and continue to play video until an intersection or the user requests a stop.

It could also be interesting to implement a prototype with video used solely as a compression mechanism to see how much improvement is gained over still imagery.

Additionally, developers should consider how video might interact with other emerging features. One can easily imagine integration with driving directions or turn-by-turn navigation, but other features pose more challenging integrations. For instance, how should business labels and location annotations interact with video controls? With the addition of video, we also have a currently unutilized audio channel. Perhaps that channel could be used for narrating turn-by-turn directions while a route is being previewed, or announce notable features at the locations being passed.

8.6 Video Server

For the experiments, a test server was setup with videos covering all of California. To launch a video application Street View would need to allocate very significant resources to handle additional server

storage space and bandwidth to support downloading video chunks and metadata. Also, the computation resources needed to preprocess the video segments needs to be considered.

8.7 Business Case

The value of implementing the Video Street View capabilities must be investigated to determine whether the return on investment would be justified. Most of Google's revenue is based on advertising sales so how VSV would support advertising and otherwise generate revenue is worthy of investigation.

9 Conclusions

In this paper I propose and analyze a procedure to incorporate videos into the Mobile Street View application. I began by proposing one manner of altering the application to include video from a usage perspective. I aim to improve cases where the user desires to view a large continuous road segment since we anticipate that this is where video would have the largest impact. Next, I considered how to generate Mobile Street View videos in order to optimize size, quality, and data transfer rates. I found that due to the small screen size users would tolerate greater quality degradation than on other platforms such as the desktop browser. This allowed for smaller video sizes and, consequently, faster downloads.

I considered how long it would take to obtain such video files over common network connections. I found that devices are not quite capable of obtaining video data at the necessary rates, but are expected to be suitable for video applications in the very near future.

Finding satisfactory theoretical conclusions, I proceeded to implement a working prototype of Mobile Street View to demonstrate feasibility. I then instrumented both the existing application and our video prototype and obtained measurements on their performance. For the existing application the largest weakness is the time it takes to show the first location to the user. Analysis of the video prototype's performance measurements showed a 4x improvement over the existing application for the use case of viewing a continuous expanse of road.

In summary, my work found great potential for the improvement of the Mobile Street View application through the incorporation of video and demonstrated this improvement concretely on the Android version of Mobile Street View.

10 Appendix A - Generating Street View Videos

This section details how the videos were generated. We began with a script, run_to_video_clips, to convert panoramic imagery into flash videos. From there we used ffmpeg, a standard video manipulation application, to convert the videos to 3GP and manipulate the videos where necessary.

```
ffmpeg -i sample.flv -s 128x96  -vcodec h263 -acodec copy test1_1.3gp -
padtop 16 -padbottom 16
ffmpeg -i sample.flv -s 176x144 -vcodec h263 -acodec copy test1_2.3gp -
padtop 28 -padbottom 28
ffmpeg -i sample.flv -s 352x288 -vcodec h263 -acodec copy test1_3.3gp -
padtop 60 -padbottom 62
ffmpeg -i sample.flv -s 704x576 -vcodec h263 -acodec copy test1_4.3gp -
padtop 112 -padbottom 112
```

```
ffmpeg -i sample1.flv -cropright 150 -cropleft 150 sample2.flv

ffmpeg -i sample2.flv -s 128x96  -vcodec h263 -acodec copy test2_1.3gp -
padleft 16 -padright 16
ffmpeg -i sample2.flv -s 176x144 -vcodec h263 -acodec copy test2_2.3gp -
padleft 16 -padright 16
ffmpeg -i sample2.flv -s 352x288 -vcodec h263 -acodec copy test2_3.3gp -
padleft 32 -padright 32
ffmpeg -i sample2.flv -s 704x576 -vcodec h263 -acodec copy test2_4.3gp -
padleft 64 -padright 64
```

```
# no low resolution tests

# medium resolution: 100 vs 200
ffmpeg -i sample2.flv -s 176x144 -b  70k -vcodec h263 -acodec copy
test3_1.3gp -padleft 16 -padright 16
ffmpeg -i sample2.flv -s 176x144 -b 200k -vcodec h263 -acodec copy
test3_2.3gp -padleft 16 -padright 16

# high resolution: 100 vs 200 vs 300 vs 500
ffmpeg -i sample2.flv -s 352x288 -b  20k -vcodec h263 -acodec copy
test3_3.3gp -padleft 32 -padright 32
ffmpeg -i sample2.flv -s 352x288 -b 100k -vcodec h263 -acodec copy
test3_4.3gp -padleft 32 -padright 32
ffmpeg -i sample2.flv -s 352x288 -b 200k -vcodec h263 -acodec copy
test3_5.3gp -padleft 32 -padright 32
ffmpeg -i sample2.flv -s 352x288 -b 300k -vcodec h263 -acodec copy
test3_6.3gp -padleft 32 -padright 32
ffmpeg -i sample2.flv -s 352x288 -b 500k -vcodec h263 -acodec copy
test3_7.3gp -padleft 32 -padright 32
```

```
for i in 1 2 3 4 5
do
  run_to_video_clip -spacing $i
  ffmpeg -i spacing$i.flv -cropleft 150 -cropright 150 square_spacing$i.flv
  ffmpeg -i square_spacing$i.flv -s 352x288 -b 200k -vcodec h263 -acodec
copy test4_$i.3gp -padleft 32 -padright 32
done
```

```
for i in 1 2 3 4 5 6 7 8
do
  ./run_to_video_clips -framerate $i
  mv output.flv framerate$i.flv
  ffmpeg -i framerate$i.flv -cropleft 150 -cropright 150
square_framerate$i.flv
  ffmpeg -i square_framerate$i.flv -s 352x288 -b 200k -vcodec h263 -acodec
copy test5_$i.3gp -padleft 32 -padright 32
done
```

```
./run_to_video_clips -framerate 4 -spacing 4
mv output.flv final.flv
ffmpeg -i final.flv -cropleft 150 -cropright 150 square_final.flv
ffmpeg -i square_final.flv -s 352x288 -b 200k -vcodec h263 -acodec copy
final.3gp -padleft 32 -padright 32
```

11 Appendix B – Instrumentation Measurements

The measurement volumes indicating the number of occurrences of the action were reported. Not every variable is reported in every action session, so the measurement volume varies from variable to variable. 25^{th}, median (50^{th}), and 90^{th} report the respective percentiles for the variable.

11.1 Complete List of Instrumentation Variables

Variables are the specific values that we want to measure. Our variables fell into three categories (1) those measured as a time from the start of the variable's action, (2) round trip times to download data from a server, or (3) counts of the number of occurrences of something.

The start times for the actions are as follows:

- (for action entrance) the start of the SV client
- (for action next) the time when the user clicked an arrow to move to the next panorama
- (for action video) the time when the user clicked an arrow to start playing videos.

No specification indicates that the variable is in category 1. Variables in the other two categories are marked with their category number. Categories (1) and (2) are measured in seconds.

Entrance only:

- uent (1) - the time between a user requesting to enter SV and the SV client to start up.
- ucnt (3) – a count of the number of panoramas viewed before exiting the SV client

General:

- upan (1) - the time a user first pans the image using the touchscreen
- unxt (1) - the time a user clicks on an arrow to move to another panorama
- utrb (1) - the time a user first pans the image using the trackball
- sc (2) - the round trip time to receive the panorama configuration file (metadata)
- sl (2) - the round trip time to receive an image tile
- prt (1) - the time a panorama is ready to be interacted with, i.e. when the road overlays are first rendered
- pst (1) - the time a panorama is fully loaded, i.e. the screen is stable until the users interacts with it
- idle (1) - the time the application pauses due to idleness

Video only:

- sv (2) - the round trip time to receive a video chunk
- vcnt (3) - a count of the number of video chunks played during this video session
- fcnt (3) - a count of the number of video frames played during this video session

- pcnt (3) - a count of the number of live panoramas passed during this video session (including the starting panorama)
- vrt (1) - the time videos begin playing
- vst (1) - the time videos complete playing
- gap (1) - the time between videos where the screen goes black

(Variables regarding a user's requests are prefaced with 'u', while those pertaining to the server have an 's'.)

11.2 Experiment 1: existing Street View Application

Variable	measurement volume	mean	25th	median	90th
uent	29	33.5	0.239	0.625	187
sc	26	3.33	1.56	2.01	7.52
sl	26	0.814	0.323	0.727	1.43
prt	23	7.53	4.04	5.88	12.6
pst	22	8.33	4.67	6.53	13.6
upan	14	12.4	11.4	11.6	15.2
Utrb	15	11.8	6.25	9.17	19.8
unxt	20	22.4	11.3	17.5	41.0
idle	9	144	93.5	198	199
ucnt	27	5	2	2	13

Table 7: experiment 1 results – entrance variable measurements

variable	measurement volume	mean	25th	median	90th
sc	105	0.356	0.214	0.286	0.526
sl	106	0.341	0.211	0.325	0.574
prt	119	2.79	1.86	2.07	4.87
pst	103	3.81	2.57	3.36	5.87
upan	31	5.74	4.00	4.50	10.9
utrb	73	4.00	1.58	2.03	9.18
unxt	117	9.65	3.07	7.13	20.0
idle	14	16.8	7.85	12.3	42.8

Table 8: experiment 1 results – next variable measurements

11.3 Experiment 2: Street View application with video augmentation

variable	measurement volume	mean	25th	median	90th
uent	45	0.758	0.580	0.654	1.03
sc	45	2.48	1.58	1.74	5.11
sl	45	0.627	0.435	0.512	1.12
prt	45	6.73	4.30	6.65	9.82
pst	32	8.78	6.99	8.84	11.1
upan	16	8.26	5.43	6.08	13.4
unxt	43	11.1	7.62	9.98	20.8
ucnt	22	6	3	4	18

Table 9: experiment 2 results - entrance variable measurements

variable	measurement volume	mean	25th	median	90th
sc	76	0.812	0.638	0.762	1.09
sl	75	0.775	0.560	0.752	1.17
prt	75	3.53	2.61	2.85	7.62
pst	43	5.91	4.26	5.06	8.97
upan	24	4.66	3.27	4.00	6.28
unxt	96	12.7	4.38	5.87	15.0

Table 10: experiment 2 results - next variable measurements

variable	measurement volume	mean	25th	median	90th
sv	18	3.33	1.67	2.20	9.66
vrt	95	2.29	1.13	2.02	4.24
gap	27	0.300	0.259	0.285	0.372
vst	115	6.98	3.68	6.94	12.4
fcnt	115	22	8	22	42
vcnt	115	1	1	1	2
pcnt	12	6	2	8	9

Table 11: experiment 2 results - video mode variable measurements

Bibliography

3GPP file format (3GP). 2010 February <http://www.3gpp.org/ftp/Specs/html-info/26244.htm>.

3GPP specification. 2010 February <http://www.3gpp.org/Specifications>.

Android. 2010 February <http://www.android.com/>.

Android Application Fundamentals. February 2010 <http://developer.android.com/guide/topics/fundamentals.html>.

Emulator Network Speed. February 2010 <http://developer.android.com/guide/developing/tools/emulator.html#netspeed>.

HTC Products. February 2010 <http://www.htc.com/www/product/g1/specification.html>.

IEEE Standards Association. February 2010 <http://standards.ieee.org/getieee802/802.11.html>.

Oliver, E. "A survey of platforms for mobile networks research." ACM SIGMOBILE Mobile Computing and Communications Review (2008): 56-63.

Rijkse, K. "H.263: video coding for low-bit-rate communication." IEEE Communications Magazine (1996): 42-45.

Stephane Lafon, Russel Smith. Street View for Client Developers. internal design document. Mountain View, CA, 2008.

T. Stockhammer, M. Hannuksela, T. Wiegand. "H.264/AVC in Wireless Environments." IEEE Transactions on Circuits and Systems for Video Technology (2003): 657-672.

T.Wiegand, G.Sullivan,G.Bjøntegaard,andA.Luthra. "Overview of the H.264 / AVC Video Coding Standard." IEEE Transactions on Circuits and Systems for Video Technology (2003): 560–576.

---

[i] The research associated with this Masters Degree was partially supported by the Google Corporation in conjunction with MIT Course VI. Google provided an internship and access to the necessary resources while I performed the research. This research aims to provide concrete practical value to Google in addition to more theoretical analyses.