Virtual Zoom: Augmented Reality on a Mobile Device

Sergey Karayev
University of Washington
Honors Thesis in Computer Science

June 5, 2009

Abstract

The world around us is photographed millions of times a day, and a lot of images find their way online. We present a way to use this data to augment reality through a mobile phone. With our application, the user can zoom in on a distant landmark using other people's photographs. Our system relies on a 3D scene modeling back end that computes the viewpoint of each photograph in an unordered large photo collection. We present and discuss the overall system architecture, our implementation of the client application on the iPhone, our approach to picking the best views to offer a zoom path, and the complexities and limitations associated with mobile platforms.

Contents

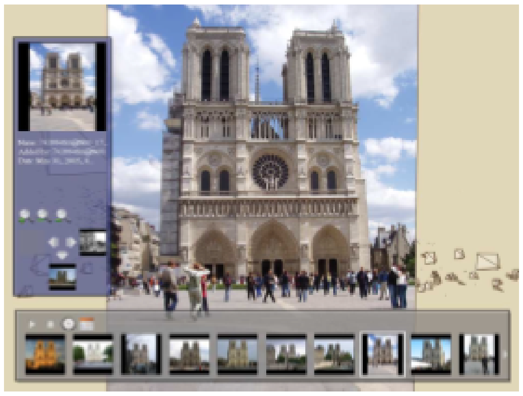
Figure 1: Related work: Photo Tourism browser

1 Introduction

The world around us is becoming increasingly well represented by photographs on the Internet. In the last year, the popular photo-sharing website Flickr has received over 2.8 million new photos per day, on average[1]. Many of the world's landmarks are represented on such Internet photo repositories by hundreds and sometimes hundreds of thousands of photos. Views of significant sites exist from almost every possible viewpoint, under a wide variety of lighting, weather, and occlusion conditions. In addition to the proliferation of consumer photography, projects exist that aim to photographically capture entire cities and countries from bird's eye view or street level[2].

Recent developments in efficient registration of images, figuring out correspondences between images of the same physical place, allow us to represent the three-dimensional structure of the real world with photographs. Applications such as Photo Tourism [10], or Microsoft's consumer-oriented PhotoSynth[3] seek to provide a way to browse through photos in an interactive three-dimensional interface. These applications are a good way to interactively tour a city or a museum from your own home, or to spice up a slideshow of a vacation, but they are missing real-time, on location applicability.

In recent years, a higher and higher proportion of consumer cameras

[1]calculated from Flickr photo ids assigned to photos uploaded on June 3 2008 and June 3 2009

[2]The most well-known is Google StreetView, `http://maps.google.com`

[3]`http://photosynth.net/`

are found embedded into cell phones. Their optical capabilities are usually worse than the cheapest dedicated cameras, but their overwhelming availability means that often they are the only camera around. A parallel trend is the improvement of bandwidth for cell phones, due to new generations of telecommunication standards and increasing demand for mobile computing. The "smartphone"–a device that combines a phone, an Internet-enabled mini computer, and a camera–is rapidly increasing in market share of all mobile devices.

The increasing coverage of the physical world by consumer photographs and the increasing availability of powerful, Internet-connected, camera-equipped mobile devices leads to our objective. We seek to access the rich representation of tourist landmarks on the Internet on a mobile device, in a useful application that can serve as a tourist's handheld companion. Our specific objective is to provide the experience of zooming in on a distant landmark by using other photos of it, having taken a picture of it on a mobile phone. Our goal is a smooth, native-feeling interface that will provide the illusion of actually zooming into an image.

1.1 Related work

There is a growing body of work on using mobile phones for augmented reality. One approach tries to match an image taken on a mobile device against a small database of landmarks tagged with keywords [12]. The retrieved keywords are used to search the Internet through a text search for more related images. Those images are then matched to the original image, and the relevant ones are returned, along with information about the landmark.

A team from Nokia Research has developed a fast outdoor augmented reality mobile phone system that does feature extraction on the device, and matches it an efficient database of highly relevant features that is optimized for the user's location [11]. A focus is made on speed and efficiency, in terms of data size and matching algorithm. They use the SURF feature descriptor [1], which is both more compressible and faster than the feature descriptor used in our system. Their work achieves near real-time augmentation of buildings with their retrieved names.

Adding to this work, Hile *et al.* [5] implement a landmark-based pedestrian navigation system. On a mobile phone, directions to a destination are presented as a sequence of photographs of landmark buildings, augmented with superimposed directional arrows. The landmark images are retrieved

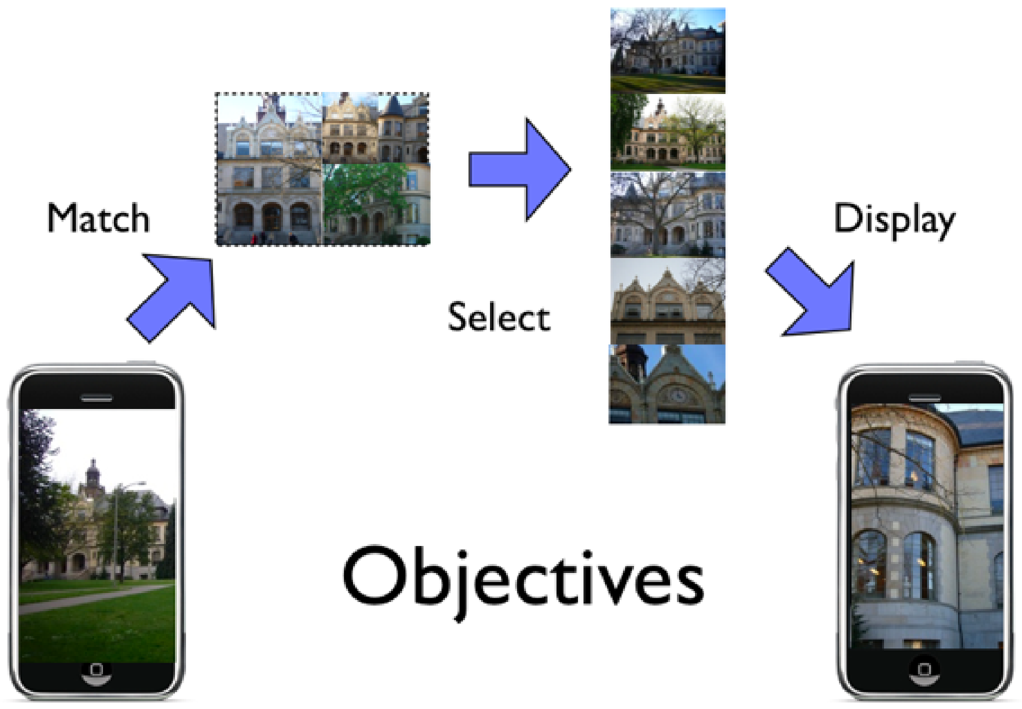

Figure 2: Our system objectives

from a geotagged database of photos. This is a concretely practical application of photo-based augmented reality on a mobile phone, following in the vein of such projects as PhoneGuide [3]. PhoneGuide is a mobile museum guide that performs object recognition on the device using a neural network.

In the domain of assembling unordered photo collections into a user-navigable 3D environment, the mentioned Photo Tourism and PhotoSynth applications have covered a lot of ground. In the mobile space, there is an iPhone client for displaying PhotoSynth data, but it does not attempt to interact with the world in any way [7].

2 System overview

Our system consists of a client-server architecture. The client is a mobile Internet-connected device with a camera. We implemented the client on the

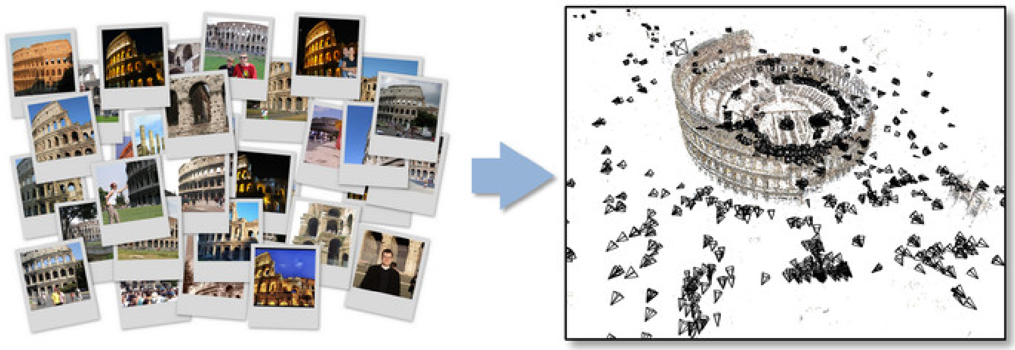

Figure 3: The structure from motion system registers unordered images and produces a sparse point cloud.

Apple iPhone 3G. The client takes a picture of the object of interest, and sends it to the server as a query. The server processes the image, matches it to a bundle, and selects a set of images that offer views of increasing proximity to the object. The client receives this set, and displays it in a 3D browser that seeks to emulate the experience of actually zooming into the image.

2.1 The Back End

The fundamental source of data for our application is a set of geometrically registered images and their recovered cameras, along with the feature points they see. We call this set of data a *bundle*. We use an existing structure from motion system that takes an unstructured, diverse collection of photographs, and recovers the camera parameters for each image, along with a sparse point cloud of feature points [10]. The photos are matched using SIFT features [6]. In the current system, these images are collected semi-manually from such websites as Google Image Search, or added by users playing a sort of game (more info in section 4).

2.2 Match

The first part of our system, the matching of an image taken by the mobile client to a bundle containing other views of the object of interest, was not implemented in our project. Instead, the user has to manually specify which bundle they expect their photo to match to. It would not be hard to implement matching, for efficient matching methods exist[[8]], and the geolocation information supplied by the mobile phone would reduce the number of potential matching bundles to a very small number. It was never our goal to do real-time matching, unlike some related projects (see section 1.1).

When the image is matched to a bundle, it is also incrementally added to it, using fast pose estimation [9]. The bundle is thus updated with the new image. In this way, users using the client application not only receive data but also contribute more images to the system.

2.3 Select

A bundle can easily consist of several hundred images and tens of thousands of points. For each image, we need to describe the camera parameters, and list the points visible in the image. The compressed size of this data for a bundle of several hundred images, which is a medium size for a bundle, is over 10MB. Transferring this much data over a cell phone connection, or even simply loading it on a memory-limited mobile device, is not feasible. Our solution is to pre-select 10-20 images from the bundle, and transfer only this subset of the bundle to the client.

We want to create a subset of the bundle that includes cameras on the path from this query image to the object of interest, defined as the set of feature points in the middle region of the query image. This is done by constructing a graph of cameras, with weighted edges representing the cost of moving from one camera to the other, with cost defined as the viewpoint score (see section 3.2) of going to one camera from the other. We then find the most zoomed-in image for the query image, and compute the shortest path from it to the query image. This is the source set of our bundle subset. The most zoomed-in image has the highest ratio of the area containing the feature points of interest projected into the image to the total area of the image.

2.4 Display

The main part of our system is the client application that provides the zoom interface to the user. Our goal in this Display part of the system was to closely mimic the native user interface way of zooming and moving around an images, in order to provide an illusion of actually zooming into the pixels of the image the user just took.

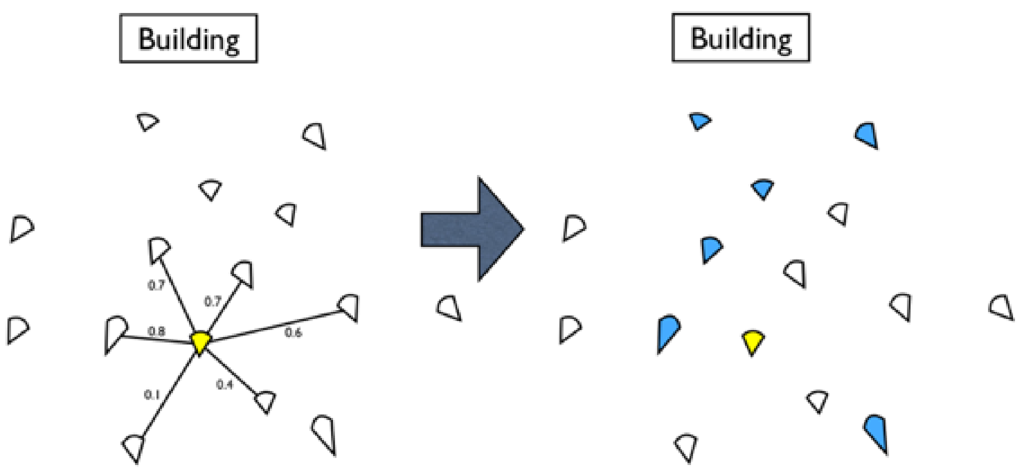

Figure 4: We construct a graph of cameras and find the shortest path to the most zoomed in camera.

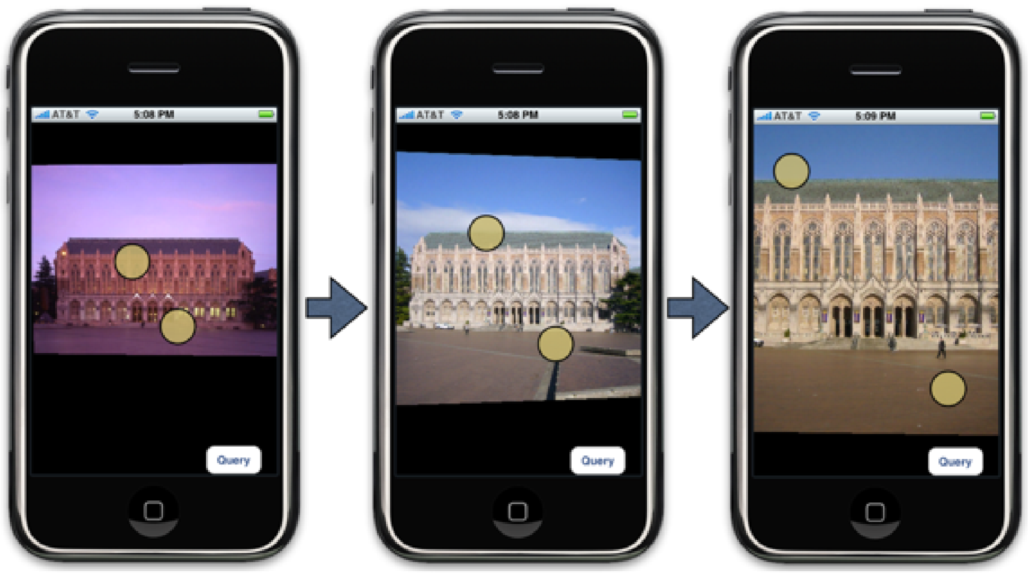

Figure 5: The user moves the viewpoint forward by spreading their pinching motion.

The target client was the Apple iPhone, a popular smartphone with excellent hardware capabilities and a multi-touch screen. The native photo viewer on the iPhone allows the user to zoom in and out of an image with a "pinch" motion, and translate the image with a one-finger swipe. Our application mirrors this interface, with pinching motions translating the viewpoint forward and backward (our equivalent of zooming in/out, details in section 3.2), and one-finger swipes translating the viewpoint up/down and left/right. The resulting user interface feels like the native iPhone photo viewer.

The viewpoint of the client is free-moving, not tied to an existing camera viewpoint. If the client viewpoint is not at an existing camera, the best image for that viewpoint is picked, and is reprojected to the desired viewpoint. This permits the smooth experience of zooming in: as long as a given image is still good, it will simply take up more and more screen space, until a better image replaces it. This results in a more continuous user interface experience than applications like PhotoSynth, or its mobile version iSynth (mentioned in section 1.1), which limit potential user viewpoints to existing cameras, with morphs between them.

It must be noted that while consistency with the native interface was a goal that we achieved, our user interface also allows another degree of freedom for the viewpoint: two-fingered swipes pan and tilt the camera. Our data is three-dimensional, and this feature lets the user experience that. For example, one could zoom close to the front of the building and then look up. This degree of freedom is of course not possible with standard photos.

3 Client Implementation

In terms of the System Objectives in Figure 2, the client application deals with Match and Display. We implemented the client application on the iPhone OS 2.2 platform, running on the iPhone 3G.

3.1 Loading data

As was mentioned, true bundle matching was not implemented in this project, and so it is the responsibility of the user to pick the bundle that their object of interest will be added to. The user takes a picture (or selects one from the device's photo library) by pressing the Query button in the user interface. This image is sent as an HTTP POST request to our server, along with the id of the bundle that the user has selected.

As the server processes and attempts to add the image to the bundle, the client repeatedly requests the bundle subset information from a specific server URL (more details in section 4). When it receives the data, in plain text format, it reads it in and populates its list of images and points seen by their cameras. It sets up a queue to download the image files that are referenced in the bundle subset. These images are in the PVR compressed OpenGL texture format, proprietary to the iPhone's PowerVR graphics chip. This format significantly reduces the amount of image data that needs to be transferred, as images are encoded with a maximum of 4 bits per pixel. Because the iPhone has a maximum texture size of 1024x1024, the maximum size an image can be is 528KB. We do not use mipmapping, for it results in images that appear too blurry when they occupy less than half of screen space.

Based on the user's viewpoint, an image may need to be displayed before it has been downloaded. In that case, it goes to the front of the download queue, and the user interface is frozen until it is downloaded. This may not be good design if the device is not on a fast connection, because loading an image can take up to 7 seconds on slow cell phone data connections. Another design decision could be to simply not consider images that have not been downloaded. The problem of download rates will be discussed further in section 5.1

Once an image has been downloaded, it is cached on disk that should persist between application starts. In case the same image is part of another subset, it will be read from disk and not requested for the server.

3.2 Viewpoint evaluation

Our implementation of the rendering engine is based on Snavely's work in Finding Paths through the World's Photos [9]. There are two components to our approach: reprojection of images from their original cameras to new viewpoints, and evaluation of potential reprojections with respect to quality.

The evaluation of candidate images to be reprojected to a new viewpoint is done with a viewpoint scoring function, which is a combination of three measures: angular deviation, field of view, and resolution. Angular deviation is the average angle between rays from the image through a set of points in the scene and rays from the viewpoint. Field of view is the area of the view that is covered by reprojecting the image into it, weighted toward the center of the view. Resolution is the ratio of the number of pixels in the image to the the number of pixels it would occupy on the screen when reprojected to the new view.

The viewpoint has 5 degrees of freedom: forward/back, left/right, up/down translations, pan, and tilt. The sixth degree of freedom, zoom through changing field of view, is not accessible through our interface. This may seem curious for a project entitled Virtual Zoom, but the way we implement "zoom" is through forward/back translations, while keeping the field of view constant. We found that this looks better and feels more natural than changing the field of view. This design decision may merit further investigation, and a more intelligent approach to picking the zoom subset on the server should consider the degrees of freedom we chose.

3.3 Rendering

All images that are part of the loaded bundle subset are thus evaluated, and the top image is the one that is used to render the viewpoint. If no image has a non-zero score for a given viewpoint, only the point cloud is rendered. The points contain color data, and a large number of them can convey a good impression of the object's structure. However, in our bundle subset we only include points that are visible in the images, and so the point cloud is too sparse to convey detailed structure.

On top of the point cloud, the best image is rendered. An image is reprojected to match the desired viewpoint by projecting it onto a planar proxy. The proxy plane for each camera is pre-computed and is included with the bundle subset data. It is computed by fitting a plane to the set of points visible in the image using RANSAC. When the viewpoint scoring function picks a different top image, the old top image begins fading out using the texture's alpha value, and the new image starts fading in. In this way, the transitions between images are smoothed.

Reprojecting an image to a new viewpoint requires a technique called projective texturing. Because we apply a flat texture to a plane that is not orthogonal to the user's viewpoint, if the texture coordinates remain in the 2D affine plane, the texture will be applied to the surface incorrectly. Texture coordinates must also be specified in the full 3D projective space [2]. On the iPhone, it seems that the interpolation of texture pixels to screen pixels is not performed correctly when not at the vertices, even if the texture coordinates are correctly specified in projective space. Our solution to this is to subdivide the quadrangle we are projecting the texture onto into more triangles (32 vs. the minimum value of 2), which results in more vertices and thus better-looking interpolation of the texture. We have not found that this tessellation results in decreased framerates.

3.4 Area of improvement

The user interface has a flaw in that a translation of the user's viewpoint has a greater effect on the apparent change in perspective the closer the viewpoint is to the scene points. So, a one-finger swipe when the viewpoint is up close to the scene points moves the viewpoint more than the same swipe performed when the viewpoint is far away from the scene points. This results in an inconsistent navigating experience. The cause is the way the viewpoint is moved: if a multi-touch event triggers one of the movement direction thresholds, the viewpoint receives an impulse to move in that direction. A continuous one-finger swipe to the left, for example, means that the viewpoints receives many impulses to move left. The way navigation should be done, to be completely true to the native iPhone photo browser UI, is that once a finger touches a point on the image, that point should stay under the finger throughout the touch event.

4 Server Implementation

Our server architecture relies almost completely on a system developed by Kathleen Tuite. Her project, PhotoCity, is a capture-the-flag game in which virtual flags are automatically planted on a map of a small region (such as a college campus), and are captured by taking and uploading a picture of that place. The goal of the game is to collaboratively build up photographic representation of an area. The photographs are organized into bundles of the same format that our system uses, and our application hooks into her server system with minimal modifications.

Briefly, the client device sends a query image to the web server, which assigns it an id, stores it on the filesystem, extracts SIFT features, and attempts to register it to the bundle specified by the client. The client, meanwhile, queries the webserver for the subset for its query image. When the bundle server finished registering the new image and writes out the selected subset, the web server returns it to the client.

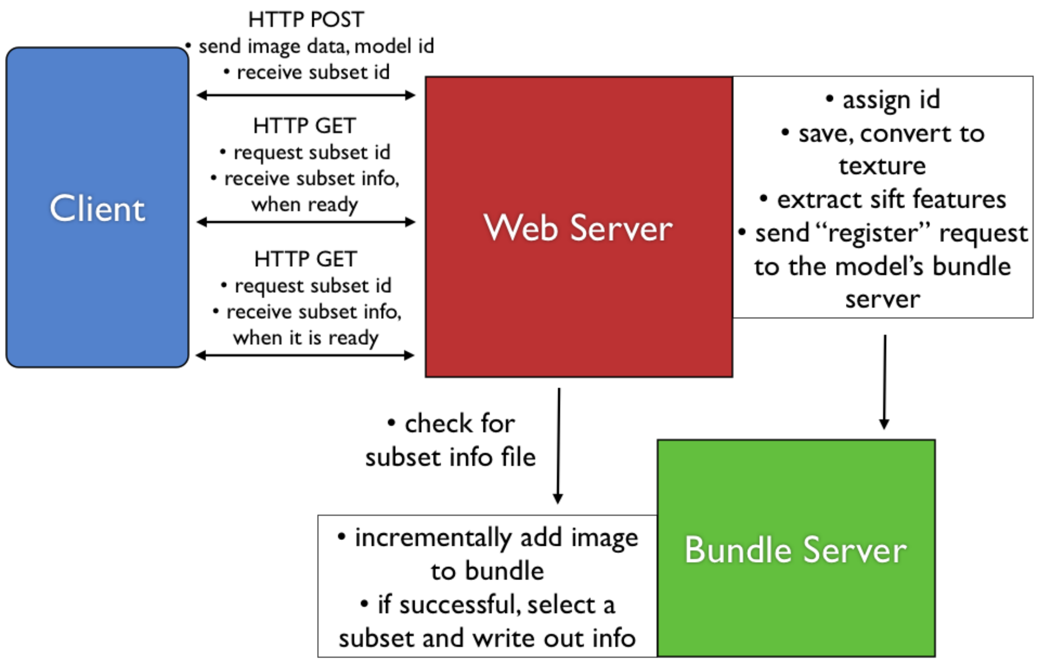
HTTP POST
• send image data, model id
• receive subset id
HTTP GET
• request subset id
• receive subset info, when ready
HTTP GET
• request subset id
• receive subset info, when it is ready
Client
Web Server
• assign id
• save, convert to texture
• extract sift features
• send "register" request to the model's bundle server
• check for subset info file
• incrementally add image to bundle
• if successful, select a subset and write out info
Bundle Server

Figure 6: An overview of the server.

The incremental bundle server takes as input a small number of new images (in our case, just one), quickly estimates their pose, and updates the bundle. This is how PhotoCity grows its bundles: every image sent with the intention of capturing a flag increases the size of the bundle. The goal of the project is a fully connected, well-covered photographic representation of blocks, neighborhoods, and eventually entire cities.

For client efficiency, each image in the bundle is converted to a PVR texture ready to be loaded into the client application. Because of the iPhone's maximum texture size of 1024x1024, and the additional requirement that textures must have power-of-two dimensions, we resize and pad each image before it is converted.

5 Results

Most of the time alloted for this project was spent on implementing the client application, and the system objective of Display (see Figure 2) was fully met. The objective of Matching an image to a bundle was not met. The objective of producing a bundle Subset in a form that is efficiently transferrable and loadable on the client was mostly met.

5.1 Client

At the beginning of the project, we had significant concerns about the iPhone's computing and rendering power, and were cautious about pursuing the goal of real-time viewpoint scoring and texture reprojection. Because of these concerns, we at first pursued a different strategy toward achieving virtual zoom: doing the reprojections on the server, and sending a static stack of images to display in sequence based on the desired zoom level.

These concerns proved to be unfounded, and we have found that a smooth, robust photo navigator can run on the iPhone and be interacted with using a native-feeling user interface. Our client display application runs smoothly at framerates of over 25 frames per second on the device, with around 15 images loaded. Viewing the video of the application in action, at `http://vimeo.com/4892120`, is highly encouraged.

The biggest issue with the viability of the application is data transfer. The 3G network is offered by the three major cellular phone service providers in this country, with the average download rate of 1250kbps in one extensive

Table 1: Data transfer times for different connection rates.

Transfer Task	500 kbps	5000 kbps
200KB image upload	12.8 s	1.28 s
700KB bundle download	11.2 s	1.12 s
500KB single image download	8 s	0.8 s

test [4], and a low number of around 500kbps. We treat 500kbps as the worst case number, and 5000kbps as the best case number, on a Wi-Fi network. We assume a bundle size of 700KB, the gzip-compressed size for a bundle of around 15 images and several thousand points. We consider a corresponding upload rate as 1/4 of the download rate. The data is in Table 1.

The application spends an additional second reading the bundle data in. The table demonstrates that the cellphone data rates are a hindrance to using the application in areas not covered by Wi-Fi. It does not seem acceptable to us to wait 8 seconds for an image to be displayed. Both 3G speeds and Wi-Fi coverage should continue to improve, so the worst case scenario should be getting better with time. The amount of data to transfer could also be reduced by using streaming. The bundle file would not have to be loaded all at once, but streamed in, with the immediately needed information first. Same could be done for images: a low-resolution version could be streamed first, and then improved. This is what the iSynth client seems to be doing, at a reasonable rate of performance.

5.2 Server

We have developed an efficient format for representing the subset of a bundle for use on the client. It includes the camera information for each image, its best-fit proxy plane, the feature points that are visible for it, location and color information about these points, and initial viewpoint properties for the scene. This minimizes the amount of initial processing the client must perform.

We have developed but not adequately tested an algorithm that picks the zoom subset, as described in section 2.3. More time could be spent on making the algorithm more robust and intelligent. For client development purposes, the bundle subsets were largely hand-picked.

6 Discussion

We have developed a robust, interactive photo viewer application, and established a client-server architecture for it, but there remain many hurdles to successful field use of the application. Although extensive field testing is necessary, network data transfer seems to be a major limiting factor. An accessible Wi-Fi network appears to be a *de facto* requirement for a responsive client experience. As discussed in section 5.1, this may not have to be the case with intelligent use of streaming. Exploring this is probably a necessary direction toward a consumer-usable product.

We made a design decision in the overall system architecture of statically selecting a subset of a bundle on the server, and making only that subset available to the client. An alternative system could consist of a more intimate client-server connection, in which they remain connected for the duration of the client's runtime. In this system, the client would send its query image, receive information about where it was added to in the bundle, and display it. Upon the user moving the viewpoint, the server would be updated, select an image for the new view, and send it over along with data allowing correct projection. The server should be able to handle multiple client connections simultaneously, as they are all exploring fundamentally the same bundle. This system would be more powerful in terms of the amount of data accessible to the client, and is worth exploring. In our opinion, the network latency would prevent such an arrangement from being usable; we opted to let the device do as much of the work as it could, and for that, it needed a static bundle of its own.

Matching images was not explored in this project. There is ample related work in matching images to a database, including from and on mobile devices (see section 1.1). It would be interesting to pair our work with a robust matching algorithm that ran in real-time. This would allow near real-time virtual zooming, and is the future vision for this type of project.

It may be useful to be able to zoom in on objects from a picture on a low-resolution camera, and we focused on this application. But our system is easily modifiable to provide different sets of views. For example, perhaps the user would like to orbit around the Statue of Liberty. Our client could be modified with orbit controls and the server with different subset selection criteria. Or, the user would like to see historical photos of a building. Our system could provide that functionality server-side.

Past work such as Photo Tourism and Photosynth has started to create a virtual world out of photographs of the real one. Our system can be viewed as the first step in linking our world to this new virtual world.

7 Acknowledgements

This project was supervised by Steven Seitz and Noah Snavely. Kathleen Tuite developed the server architecture for another project and was an indispensable resource. This project was supported in part by a Mary Gates Research Scholarship.

References

[1] Bay, H., Tuytelaars, T., and Van Gool, L. Surf: Speeded-up robust features. In *9th European Conference on Computer Vision* (Graz, Austria).

[2] Everitt, C. W. Getting to know the q texture coordinate. `http://www.r3.nu/~cass/qcoord/`.

[3] Föckler, P., Zeidler, T., Brombach, B., Bruns, E., and Bimber, O. Phoneguide: museum guidance supported by on-device object recognition on mobile phones. In *MUM '05: Proceedings of the 4th international conference on Mobile and ubiquitous multimedia* (New York, NY, USA, 2005), ACM, pp. 3–10.

[4] Gizmodo. The definitive coast-to-coast 3g data test. `http://gizmodo.com/5111989/the-definitive-coast+to+coast-3g-data-test`, December 2008.

[5] Hile, H., Vedantham, R., Cuellar, G., Liu, A., Gelfand, N., Grzeszczuk, R., and Borriello, G. Landmark-based pedestrian navigation from collections of geotagged photos. In *MUM '08: Proceedings of the 7th International Conference on Mobile and Ubiquitous Multimedia* (New York, NY, USA, 2008), ACM, pp. 145–152.

[6] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60* (2004), 91–110.

[7] Pascale, G. isynth: A photosynth viewer for the iphone. `http://www.cs.brown.edu/people/gpascale/iSynth/`.

[8] Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2007).

[9] Snavely, N., Garg, R., Seitz, S. M., and Szeliski, R. Finding paths through the world's photos. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008) 27*, 3 (2008), 11–21.

[10] Snavely, N., Seitz, S. M., and Szeliski, R. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings* (New York, NY, USA, 2006), ACM Press, pp. 835–846.

[11] Takacs, G., Chandrasekhar, V., Gelfand, N., Xiong, Y., Chen, W.-C., Bismpigiannis, T., Grzeszczuk, R., Pulli, K., and Girod, B. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *MIR '08: Proceeding of the 1st ACM international conference on Multimedia information retrieval* (New York, NY, USA, 2008), ACM, pp. 427–434.

[12] Yeh, T., Tollmar, K., and Darrell, T. Searching the web with mobile images for location recognition. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on* (2004), vol. 2, pp. II–76–II–81 Vol.2.