HD4AR: High-Precision Mobile Augmented Reality Using Image-Based Localization

Paul N. Miranda

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

C. Jules White, Chair

Mani Golparvar-Fard

Jeffrey H. Reed

April 30, 2012

Blacksburg, Virginia

Keywords: cyberphysical systems, smartphones, android, augmented reality, construction

HD4AR: High-Precision Mobile Augmented Reality Using Image-Based Localization

Paul N. Miranda

(ABSTRACT)

Construction projects require large amounts of cyber-information, such as 3D models, in order to achieve success. Unfortunately, this information is typically difficult for construction field personnel to access and use on-site, due to the highly mobile nature of the job and hazardous work environments. Field personnel rely on carrying around large stacks of construction drawings, diagrams, and specifications, or traveling to a trailer to look up information electronically, reducing potential project efficiency. This thesis details my work on Hybrid 4-Dimensional Augmented Reality, known as HD4AR, a mobile augmented reality system for construction projects that provides high-precision visualization of semantically-rich 3D cyber-information over real-world imagery. The thesis examines the challenges related to augmenting reality on a construction site, describes how HD4AR overcomes these challenges, and empirically evaluates the capabilities of HD4AR.

Acknowledgements

I would like to thank Dr. Jules White, my advisor, for taking a chance on a guy who likes waiting in hallways. Always full of great advice and ideas, I feel like I learned something every time I got a chance to speak with him. I'll be forever grateful for getting the chance to join Team Magnum and grow as an engineer under his tutelage.

I would like to thank Dr. Mani Golparvar-Fard for his extensive guidence and support on my research. He's provided me a unique perspective on problems on many occasions, and introduced me to a whole different facet of engineering and programming.

I'm thankful to Dr. Jeffrey Reed for agreeing to help me through the last leg of my time at Virginia Tech.

My thanks go out to Hyojoon "Joon" Bae and Nikita Sharakov for their assistance on this research. Joon has already picked up the torch on this project, and will undoubtedly continue to produce excellent work from here on out. The same goes for the rest of Team Magnum, may you have nothing but success.

I wouldn't be here without my friends, both new and old, and all of their support. All of

you are fantastic, and I wish you well wherever you end up.

And finally, I have to thank my family, from which i've received nothing but love and support all my life. And though it pains me to widen the physical distance between us once more, I know that they will always be in my heart and prayers as I am in theirs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Construction Sites and Information Retrieval

Construction projects currently face the challenge of effectively accessing, interacting with, and visualizing the information necessary for project success in a physical context. This information, which is typically in the form of construction drawings, specifications, and/or expected progress information, is detached from the physical elements to which it relates, leading to reduced efficiency as engineers or field personnel attempt to reconcile cyber information against their physical surroundings. The quality and timing of transferring or exchanging hundreds to thousands of pieces of information can either delay or facilitate the execution of a project during the construction phase [1].

Current practice of construction progress monitoring includes manual and time-consuming

data collection (e.g., walking around a construction site and writing down information on paper), non-systematic analysis (e.g., rough comparisons of the notes to the 2D construction plans), and visually/spatially complex reporting [2],[3] (e.g., estimating how real-world 3D physical structures correspond to 2D building plans or virtual 3D models).

A key issue with these manual processes is that practitioners cannot access and interact with cyber-information through the physical structures that they are building. Instead, field engineers must either carry bulky stacks of drawing and documents, or make trips back and forth to construction trailers and offices to look up plans for building elements and compare them to what was seen on the construction site. Because there is no easy way to visualize and query the cyber information through the actual physical construction elements, it is difficult for field personnel to quickly detect discrepancies between the physical elements and the cyber building plans. Without quick identification of discrepancies, managers cannot easily adjust the construction plan to minimize the impact of problems.

1.2 Hybrid 4-Dimensional Augmented Reality

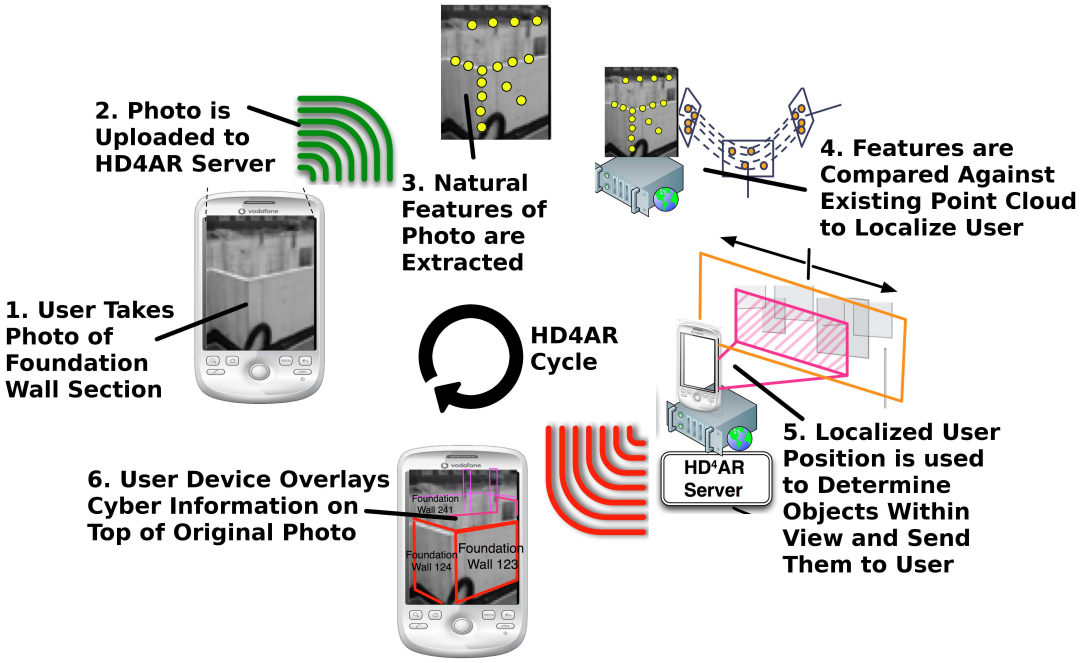
Throughout the course of my work, I have developed a system known as Hybrid 4-Dimensional Augmented Reality (HD4AR), for allowing construction personnel to visualize cyber-information on top of real-world imagery. HD4AR is an augmented reality system that merges a physical model of a construction site, built using the natural features from images, with a cyber Building Information Model (BIM, a 3D CAD model that is annotated with cyber-information).

HD4AR allows construction field personnel to use mobile devices, such as smartphones or tablets, to take pictures that include a specific construction element, see BIM elements visually overlaid on top of the real-world imagery, touch or click on a BIM element in the image, and be presented with a detailed list of cyber-information, such as plan information (e.g., budget, specifications, architectural/structural details) or actual information (e.g., cost, safety provisions, physical progress) related to the physical element.

Because mobile devices are commonly available on construction sites [4], and unite many of the components needed for an augmented reality system (e.g., camera, color display, sensors, etc.) into a single unit [5], they present a unique opportunity for bringing augmented reality to the context of construction projects.

HD4AR is based on a computer vision approach that uses image feature detection, matching, and Structure from Motion (SfM) algorithms to extract 3D geometry from a set of overlapping photographs. Initially, HD4AR is bootstrapped by extracting 3D geometry from daily construction photographs using an offline process. Once the system is bootstrapped, field engineers can take new photographs with a mobile device, wirelessly upload them to an HD4AR server, have them localized in the extracted 3D geometry, and then receive an augmented photograph with BIM elements precisely overlaid on the real-world imagery. Our empirical results show that this end-to-end process can be completed in roughly 60 seconds or less. An overview of the capabilities of HD4AR is shown in Figure 1.1 and screenshots from the Android HD4AR client are shown in Figure 4.2.

Figure 1.1: The HD4AR Cycle



1.3 HD4AR Over Time

Over the course of work on HD4AR, its form and function has greatly changed in order to continually expand and improve its capacity to localize and augment a user with only an image.

Through my research, it was deemed possible for a user to be localized via a single image. However, this process was impacted by a range of factors, such as the texture of building materials (e.g., concrete walls vs. brick walls vs. structural steel elements). While localization failure is desired in the case where an image is taken from a location not represented in the cyber-physical model (i.e., we do not want a false localization from a picture not depicting the modelled environment), failure in the case where the captured image does include a

section of the model is undesired.

Repeated failure of attempted registration leads an engineer to waste time attempting to take a new picture that will result in a successful localization. With efficiency being such a high concern on construction sites, the robustness of the system in localizing the user is a critical issue, as the benefit in time expended in using our mobile augmented reality system is lost through repeated unsuccessful localization attempts. Furthermore, the time for a successful registration was also a concern, since a long registration process would once again cut into efficiency benefits.

These concerns provided the catalyst to further develop HD4AR, and build a system that in not only operational on a basic level, but effective and efficient as well.

1.4 Research Contributions

My work on HD4AR provides the following contributions to the ongoing study and execution of augmented reality in a construction context:

- It prevents a novel means of localizing a user without the need for external hardware, relying only on natural features.
- It provides a general purpose algorithm for mobile image augmentation using static images.
- It empirically proves the robust potential of this system for localizing a user.

1.5 Research Scope and Overview

The remainder of this thesis is organized as follows: Chapter 2 presents a motivating example that illustrates the challenges of associating cyber-physical information on construction sites; Chapter 3 describes the research challenges of cyber-physical information association; Chapter 4 discusses HD4AR and its approach to high-precision cyber-physical information association; Chapter 5 presents empirical results from experimentation with HD4AR; Chapter 6 compares HD4AR to related work; and finally Chapter 7 presents concluding remarks.

Chapter 2

Motivating Example

As a motivating example, we use a scenario where a field engineer is assessing the construction progress and/or quality of a basement concrete foundation wall. With current approaches, the field engineer would return to a construction trailer or office and open 2D construction drawings (at best a 3D BIM), project specifications, and the schedule to find out when the construction of this element is expected to be finished and what is the required quality of the outcome. Once the drawings and/or 3D building model are opened, the field engineer must navigate the model to determine which element, of possibly hundreds or thousands of walls, is the foundation wall of concern. Moreover, once the information is obtained, the field engineer may need to return to the construction site to compare the information that was retrieved to the actual construction status of the real foundation wall. Because there is no way to directly query the cyber information for the wall, the field engineer may not notice a discrepancy and as a result will not be able to decide on a corrective action to minimize

the impact of the discrepancy. It is much better to find and correct a mistake before it has been made than attempt to fix it afterwards.

Thus it is beneficial for an engineer to quickly and easily associate cyber information with physical elements. Rather than using traditional methods to retrieve and associate the building information with the foundation wall, it would be useful if the engineer could remotely query the information. This could be done directly from the site using a picture of the foundation wall as the basis for the query. This picture provides all that is needed to localize the user with respect to their environment, and thus reduce the information available down to what is relevant to the current scene. Given the close proximity of construction elements and their similar appearances, the location and orientation of the picture needs to be accurately estimated to return the related information to the field engineer. Furthermore, this process should not affect the constructions workflow by requiring tagging of elements, which, considering the number of elements on a typical residential building construction site (1000-30,000 elements), makes tagging time consuming and often impractical.

Chapter 3

Challenges

This section details specific challenges relating to mobile augmented reality on a construction site.

3.1 Challenge 1: Attaching Sensors to Objects at a Construction Site is Time-Consuming, and Requires Significant Cost and Maintenance

Many of the efforts to improve the control of construction and maintenance of heavy equipment involve attaching sensors to construction objects to track them. Field engineers can then use these sensors or tags to query information related to the physical construction element. For example, [7],[8] have used GPS systems to track the locations of heavy equipment

and provide map-based displays of equipment locations. The integration of GPS with cyber decision support systems has been shown to help engineers in safely monitoring geotechnical construction processes [9],[10]. Similarly, active RFID tags mounted on specially designed helmets can track workers and machines positions [11],[12].

Unfortunately, while the cost of acquiring sensors makes tracking heavy equipment and workers reasonable, tracking thousands of joints, columns, and beams is prohibitively expensive in terms of sensor cost and overall time for sensor placement [2],[13],[14]. Furthermore, after sensors are attached to a construction object (e.g., a column) the object will be under continual modifications. The sensor may have to be moved continually to avoid being painted, covered with finishing materials, or protected from vibration. The constant evolution of the construction site and maintenance of tags makes tracking items other than equipment challenging [14].

Other approaches use cheap tags, such as barcodes and RFID tags [3],[15] or fiducial markers [16],[17], to associate a cyber-identity with a physical object. These approaches do not require charging sensors or significant cost. However, they are still subject to the same maintenance challenges of being exposed to a construction site. They may be covered by dust, finishing materials, or hidden and must be continually relocated and/or reapplied. Moreover, these systems are read-only; they allow information to be associated with a construction object, but they do not allow cyber-information to be updated by sensing the change in a construction object. For example, a barcode cannot differentiate similar construction elements and hence cannot accurately detect construction progress.

3.2 Challenge 2: Mobile GPS/sensor Augmented Reality Techniques are Too Imprecise for Construction Applications and Rely on GPS, which May Be Inaccurate Inside Structures.

In prior research, we and many others, have looked at using GPS-based augmented reality (AR) techniques to overlay cyber-information on user views of the physical world [18],[19]. These approaches rely on calculating the users position and field-of-view in order to map cyber-information into physical imagery. Cyber-items are tagged with latitude, longitude, and altitude. The mobile device determines its location and calculates which cyber-information items should be in view using the camera focal length.

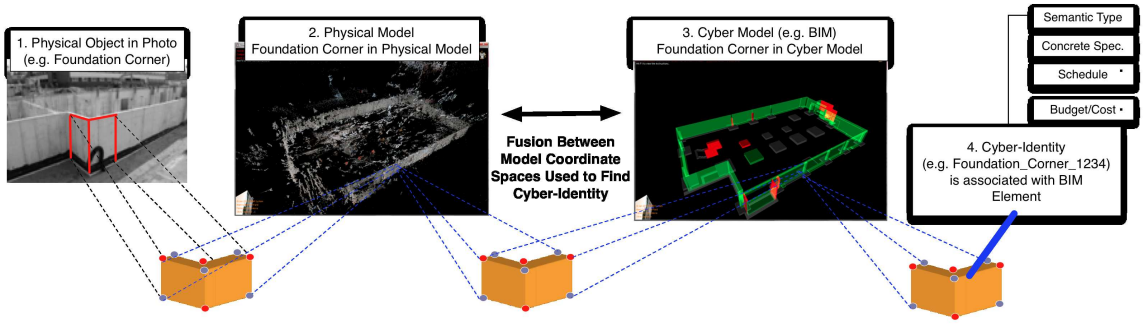
The use of GPS and geomagnetic sensors in commodity mobile devices are, however, accompanied by signicant challenges, such as the limited accuracy of the GPS sensors and the noise present in sensor data [20]. For example, the noise in geomagnetic heading values can cause jitter in onscreen information presentation. Algorithms, such the Savitzky-Golay smoothing lter [21],[22], or Finite Impulse Response [23],[24], can be used to reduce jitter, but sensor noise can still have a substantial impact on accuracy. For safety-critical applications, such as building construction, this level of error can be catastrophic. Moreover, inside of buildings under construction, GPS signals may be significantly reduced or eliminated.

3.3 Challenge 3: Computer Vision Approaches for Building Fused Cyber-Physical Models are Offline and Have Not Been Used to Create Mobile Read/Write/Feedback Capabilities.

In the past decade, advances in digital photography and image processing have led to much research into the application of image-based reasoning for various construction management tasks and into techniques that can manually, semi-automatically, and automatically interpret them [2],[10],[25],[26],[33]. It is common for jobsite images to be gathered periodically, stored in central databases, and utilized in communication and coordinating project tasks [12]. Because of the low cost of digital camera equipment, computer vision techniques are an appealing approach for extracting 3D geometry from a construction site when compared to other approaches, such as laser scanners [27],[28],[33].

As shown in Figure 3.1, these computer vision approaches can be used to create fused cyber-physical models that can map cyber-information to physical objects in photographs. To date, the complexity of handling large data sets, lack of mobile interfaces, and lack of techniques for querying information on demand have prevented these techniques from being used for dynamic information retrieval on construction sites. Applications have mostly focused on site status and data archiving [29] tasks that can be performed offline. Computer vision techniques work extremely well for creating fused cyber-physical information models,

Figure 3.1: Cyber-Phsical Model Combining Physical Structure with Cyber Identity



but need to be adapted and provided with mobile and cyber-information association and querying interfaces for near real-time access to cyber data [2].

In prior work [27],[33], we have developed computer vision and machine learning algorithms, called D4AR – 4 Dimensional Augmented Reality, for automated tracking of construction progress from photographs. Our research has shown that existing daily construction images are usually taken far from one another and may not have enough visibility for accurate progress monitoring. In some cases, building progress monitoring estimates based on the 3D reconstructions from views with very large baselines will be incorrect [2]. A specific aim of the proposed work is the investigation of the science and supporting tools for a method of detecting these situations where there is insufficient knowledge of the physical world objects, and providing feedback to the site engineer. The feedback should take the form of confidence levels in the data, together with proposed measurement steps to improve the estimate of the fused cyber-physical model.

Chapter 4

Solution

Using computer vision, researchers have shown that a set of overlapping images can be used to extract the 3D geometry of a stationary object, such as a building under construction [2],[10],[27]. As shown in Figure 3.1 the 3D geometry extracted using computer vision can then be overlaid and aligned with manually created BIM models of the construction project to create a fused cyber-physical model. Numerous research approaches have looked at the applications of these techniques [14],[27],[33]. After the cyber (e.g., BIM elements) and physical (e.g., extracted 3D geometry) models are aligned, they can be compared to determine how the actual state of the physical object compares to the representative cyber-model. Researchers have shown that the fused cyber-physical model is accurate to within millimeters [27],[33] and can be used to predict with high accuracy the actual construction progress versus the planned cyber-model, even when visual obstructions are present [25],[33]. Moreover, the fused cyber-physical model can be used to precisely predict where cyber-

information, such as BIM elements, should appear in the original photographs and where physical objects actually appear in the cyber 3D space. This ability to track where BIM elements (i.e., cyber-identities) should appear in photographs serves as the foundation of HD4AR.

4.1 HD4AR Usage Overview

HD4AR extends and integrates our previous research efforts in mobile AR, mobile device software optimization, and fused cyber-physical model construction to create a novel hybrid approach that is focused on fusing new photographs with existing cyber-physical models. HD4AR provides augmented photographs, rather than real-time augmented camera imagery. As shown in Figure 1.1, using our approach, a field engineer is able to take a photo of a physical object and the mobile device works with a backend server to produce an updated photo augmented with the cyber-information associated with the physical objects in view. The entire end-to-end process after the photograph is taken requires roughly a minute. The augmented photographs, with cyber-physical information associations, are accurate to within a few millimeters (e.g., 1-5mm) due to the empirically demonstrated high precision that computer vision algorithms provide [27]. Moreover, after a field engineer has taken a photograph, he or she is able to select physical objects in view as the context and the system uses the fused cyber-physical model to identify the cyber-identity of those objects.

4.1.1 HD4AR Bootstrapping Process

In order to augment a given photograph with cyber-information, the HD4AR system requires that construction field personnel first take overlapping photos of the construction site. The HD4AR system uses the Bundler package implementation of Structure from Motion (SfM) [30] to produce an initial 3D point cloud of the base scene, estimating the 3D position of the features through image feature matching and an optimization process called bundle adjustment. This can be reduced to the following steps:

1. Extract Visual Features

   A feature-detection algorithm is run on each image to create a key file of feature descriptors to be used as the basis for matching images to one another. Bundler by default uses the SIFT (Scale Invariant Feature Transform) algorithm [32] for feature extraction.

2. Match Visual Features

   The set of visual features for each image are iteratively compared against each other, developing matching tracks. These tracks consist of specific feature points of multiple images, which are believed to represent the same physical feature point.

3. Estimation of Camera Parameters

   The camera parameters are estimated for each image, and the 3D positions of their feature points are estimated. Then, each additional image attempts to register with

the 3D cloud. This registration fails in the event that the estimated 3D layout of its matched points does not mesh with the set positions of the previous images.

4. Bundle Adjustment

   In addition, while the images are being added, the 3D reconstruction pipeline is run through a sparse bundle adjustment module to minimize the error in the predicted 2D positions of the feature points in the photographs given their assigned 3D positions and where the feature points actually appear in the image.

4.1.2 Photograph Augmentation Process

Figure 4.2 shows screenshots from the HD4AR client running on an Android device with Android version 2.3.4. Once the point cloud is generated, the system is prepared to augment photos sent from the HD4AR client running on a users mobile device. From a high-level perspective, this process operates as follows:

1. The field engineer, upon finding a section of the worksite he/she wishes to query, takes a picture of the area using a mobile device.

2. The device uploads the captured image to the HD4AR server.

3. The server runs a reduced form of the structure from motion algorithm to localize the image with the base point cloud.

4. Using the localized image as input, the server determines what BIM objects are within the images field of view, and where they appear.

5. The objects are sent back to the user device with positional information.

6. The user device renders the captured image, overlaid with the returned objects.

4.2 Progression of HD4AR

4.2.1 Initial Implementation

The initial implementation of HD4AR was far different from its current state. At this point in time, greater concern was placed on proving the viability of location-based augmentation via user photographs than demonstrating its effectiveness or efficiency.

Full Bundling

In the first version of HD4AR, user localization was accomplished through a full run of Bundler, adding the user-submitted image to the list of inputs consisting of the base set of images. By doing this, the new image would have its features extracted and matched to the other images in the set directly, allowing Bundler to generate a fresh point cloud based on this composite set of inputs.

If the image was included in the final bundle, points within the generated point cloud that

were visible within the image would indicate so by listing the image among its list of views, as well as the 2D coordinates of the point in the view.

This solution provided an easy means of localizing the user image with respect to the base set of images. However, full bundling is a computationally expensive operation, which increases exponentially as the number of images in the original set increases (see Figure **??**)

An improvement was made by modifying the algorithm to re-use previously generated SIFT key files (if present), thus eliminating redundant generation of keyfiles for the base set on each matched user image. While this adjustment reduced the overall time necessary to localize a user, the problems persisted.

Bundler's add_image

Rather than a full bundling approach, the option was explored to use Bundler's built-in add_image method to mesh a new image to an existing point-cloud. Thus, instead of performing a full bundling procedure and matching the entire set of images, only the new image needed to be matched. This method makes more logical sense for our objective of localizing a user with respect to an existing cyber model of information.

The problem we ran into with this approach is that, while a successful adding would provide the translation and rotation matrices for the added image with respect to the transformed point cloud, the points list would not be updated with view information. Each augmented point would have to have its 2D position with respect to the user image calculated to deter-

mine whether it is within view, after being found in the list.

Limited Augmentation

Originally, augmented data was limited to being assigned to points already tracked within the point cloud generated by Bundler. The reasoning behind this was two-fold: One, this made it easy to determine which points appeared in the submitted image, as well as their positional data, since this is automatically calculated by Bundler. Two, when an image was added to an existing set, the resulting point cloud generated by Bundler would be transformed from the original cloud. Thus, having not yet developed the reverse-transformation process that would allow us more freedom of augmentation (see below), the only established lines of connection between the point clouds were natural features specifically being tracked by Bundler. Thus, augmentation consisted of connecting semantic information with a specific point index in the original point cloud. Post-matching, the same point would then be found in the new point cloud (by comparing the views in which the point was visible) This was accomplished through a multistep process consisting of:

1. Parsing the point cloud for points that could be seen within the view of the added image

2. Matching augmented points with the parsed list of visible points (by comparing the other views that could see the point)

3. Extracting the X and Y position of the augmented points in the view of the added

image (contatined within Bundler's output)

This method not only restricts the ability to augment custom defined positions and shapes in 3D space, it also adds to general computational overhead. For these reasons, it was apparent that HD4AR would need to be expanded upon to allow the capability for augmentation that was not restricted to points tracked in the point cloud. An example of the output of this method is shown in Figure 4.1.

4.2.2 Version 2

While the initial approach proved that it is possible to augment based only on a user-supplied image through localization of its natural features, the limitations on what could be augmented eliminated its usefulness. For if only what was already present could be augmented, it would be impossible to augment a view with models of construction elements still in progress or not yet started. According to our original intention, the point cloud should be the tool used for localizing the user and aligning their physical environment with the cyber model; it should not be the cyber model itself.

To this end, effort was made to redefine the augmentation and localization process to enable more freedom. As was previously mentioned, when Bundler's add_image method is invoked to mesh a new image with an existing point cloud, the result is a transformed and optimized version of the point cloud. This transformation disrupts the alignment of the cyber model with the physical model, which prevents accurate augmentation unless the coordinates of

Figure 4.1: Limited Augmentation Based on Tracked Points (Source: Paul N. Miranda)

cyber elements can be reconciled with the new coordinate space of the transformed physical model. The updated version of HD4AR is based around solving this problem by reverse-engineering the transformation of points that occurred when the new image was added, then applying this transformation to cyber elements to determine their new coordinate positions. Once transformed, these cyber elements can be flattened down to 2D coordinates and, if within view, transmitted to the user for display.

Examining more closely, the steps can be broken down into the following procedures:

Localization of Physical Elements in Cyber model.

As shown in Figure 3.1, a physical construction object (1) can be located in the 3D geometry model (2) (actual physical model), extracted with computer vision algorithms, and then mapped to 3D BIM elements in the (3) cyber model (e.g., BIM). This determination of the cyber-identity relies on mapping between the coordinate spaces of the models. If a physical object is at the same coordinates in the extracted 3D point cloud as a specific cyber-identity in the BIM, then it is associated with that cyber-identity. The (4) cyber-identifier, in turn, is associated with a BIM element and cyber-information items that contain data about the concrete foundation wall, such as concrete specifications or expected quality of the finished surface, are associated with the cyber-identity.

Reduced Single-Image SfM.

The users photograph must be localized with respect to the 3D point cloud, using a reduced sequence of SfM algorithm steps used to create the point cloud itself. An initial estimate of the focal length of the camera used to take the image is extracted from the image's EXIF tags, or cross-referenced from a list of camera types. Next, feature point extraction is done to create a key file for the image that lists all of the images feature points. The key file and the focal length are then used to mesh the image into the point cloud. The images feature points are matched against the feature points of other images in the set. If matching succeeds, the pose of the camera that took the image is known, resulting in rotation and translation values for the camera that took the newly added image, and adjustment performed on the new 3D point cloud to refine its accuracy.

Building the Transformation Matrix.

When an image is added to an existing point cloud, the bundle adjustment will optimize with respect to the relative locations of all images in the reconstructed scene, including the newly added image, resulting in a new coordinate system. To determine visible cyber-elements, the 3D positions must be transformed into this new coordinate space. This is done using a transformation matrix built using four tracked points from both the original and transformed bundles, used to solve Ax=B, where Matrix A is a 12×12 matrix formed from using the positions of four points, shown in Table 4.1.

Table 4.1: Matrix A

x_1	y_1	z_1	1	0	0	0	0	0	0	0	0
0	0	0	0	x_1	y_1	z_1	1	0	0	0	0
0	0	0	0	0	0	0	0	x_1	y_1	z_1	1
x_2	y_2	z_2	1	0	0	0	0	0	0	0	0
0	0	0	0	x_2	y_2	z_2	1	0	0	0	0
0	0	0	0	0	0	0	0	x_2	y_2	z_2	1
x_3	y_3	z_3	1	0	0	0	0	0	0	0	0
0	0	0	0	x_3	y_3	z_3	1	0	0	0	0
0	0	0	0	0	0	0	0	x_3	y_3	z_3	1
x_4	y_4	z_4	1	0	0	0	0	0	0	0	0
0	0	0	0	x_4	y_4	z_4	1	0	0	0	0
0	0	0	0	0	0	0	0	x_4	y_4	z_4	1

B is a 1×12 matrix composed of the x, y, and z positions of the translated points from the new bundle, shown in Table 4.2. Solving for X produces a 1×12 matrix, which produces the transformation matrix T when parsed as in Table 4.3.

Table 4.2: Matrix B

x_1
y_1
z_1
x_2
y_2
z_2
x_3
y_3
z_3
x_4
y_4
z_4

Table 4.3: Matrix X (1×12), reconstructed to form the transformation matrix T (4×3)

X[0]	X[1]	X[2]	X[3]
X[4]	X[5]	X[6]	X[7]
X[8]	X[9]	X[10]	X[11]
0	0	0	1

Transformation of the BIM Overlay Positions to New Coordinate Space and Projection into 2D Pixel Space of User Image.

The following formulas are used to transfer the BIM overlay positions to new coordinate system and project them into 2D pixels of the user image. In the following, equations (2), (3), (4), and (5) are assuming that a pinhole camera is used (typical assumption for SfM algorithms).

BIM object coordinates transformed to new coordinate system

$$C = T \times \begin{bmatrix} c \\ \\ 1 \end{bmatrix} \tag{4.1}$$

Item position in camera coordinates, where R is the rotation matrix of the camera, t is the translation matrix, and C is the 3D coordinate of the point.

$$P = R \times C + t \tag{4.2}$$

Flatten to Z-axis (due to homogenous coordinate systems).

$$p = -P/P.z \tag{4.3}$$

Resolve to 2D position, where f is the estimated focal length, p is the point in pixel coordinates

$$p = f \times r(p) \times p \tag{4.4}$$

Supplemental function r, where k1, k2 are the estimated distortion parameters.

$$r(p) = 1.0 + k1 \times ||p||^2 + k2 \times ||p||^4 \tag{4.5}$$

4.2.3 Improved Matching Capability

Once testing began, less-than-optimal results appeared with respect to localizing a user image to the physical model. Since this aspect is key to the functionality of HD4AR, a solution was needed to increase the robustness of the system. This solution involved adding and modifying Bundler's runtime constraints, resulting in a massive increase in localization potential. The following chapter provides additional details on these changes.

4.3 Mobile Augmented Photo Viewer and Touchscreen Interface.

To demonstrate the usability of HD4AR in a mobile capacity, a client-server model was developed to faciliate the uploading of images from a mobile device to be localized and augmented. The client works by communicating with the HD4AR server over HTTP, enabling engineers to use the system anywhere with a connection to the internet. OpenGL was used for augmenting the user photograph with the received polygonal information from the cyber model. Cyber-information items with 2D positions within the bounds of the photograph are sent back to the mobile device along both their calculated 2D locations in the photograph

and any data associated with the item. Figure 4.2 shows screenshots from the Android HD4AR client overlaying BIM information related to a window on a photograph. In the screenshots, the 3D BIM information is precisely aligned with the real-world imagery despite changes in the position and orientation of the users device. Clicking on elements can show additional cyber-information, or potentially be used to open more detailed views for more complex data sets.

4.4 Source Code

The source code for the Android client and java servlet server implementations of HD4AR can be found in the thesis appendix.

Figure 4.2: Polygon Based Client Display (Source: Paul N. Miranda, using [34] to add frames)

Chapter 5

Experimentation

This section describes experiments conducted to assess HD4ARs ability to localize a user based on a submitted image and display appropriate BIM information.

5.1 Experimental Platform

The server-side components of our experiments were run on a workstation with 24 gigabytes of 667 MHz DDR3 1333 RAM, a 4-core Intel i7 CPU 960 (@ 3.20 GHz) processor running Ubuntu version 11.04 Natty Narwhal, and using Bundler version 0.4. The pictures used to create the 3D point clouds came from the construction site for the new Virginia Tech Center for the Arts building construction project. Images for bootstrapping the system and creating the initial 3D point cloud were taken using a NIKON D300S 13.2-Mpixel camera, but to reduce computational cost of the 3D reconstruction pipeline, were resized to 50% scale

in order to create the physical model. Images for localization were taken using an iPhone 4 and HTC Incredible S. Since the camera sensor length of the Incredible was not readily available, we performed additional camera calibration steps to estimate sensor width, plus focal length, and thereby increase the probability of successful localization. The camera calibration toolbox of Bouguet [35] was used for calibration purposes.

5.2 Experiments

5.2.1 Localization Robustness

This experiment was designed to determine on average how many photographs a construction engineer would need to take in order to be localized and display BIM information on the photograph.

Setup

From the base image set taken with the NIKON, four subsets of images were selected. First, a set consisting of 25 images, attempting to cover the entire scene with a minimal number of photographs and thus, less feature points to match against. The process was repeated for 50, 75 and 100 image sets. Each of these subsets was run through Bundler to generate the initial reconstruction point cloud of the scene. After finishing initial bundle reconstruction, test images were taken using an iPhone and HTC Incredible S, to represent a users smaller, more

mobile interface device. These images cover roughly the same area as the NIKON photos. From this set, 138 images from iPhone and 109 images from HTC Incredible S were selected, which displayed the same area as selected for the initial reconstruction bundles. Each image from this set was then individually run through HD4AR to determine if the image could be matched and the camera position that took the image derived.

Hypothesis

We hypothesized that it is possible to use user-generated images to localize against an existing point cloud using HD4AR, and that as the number of images used to create the initial point-cloud increases (thus increasing the area of overlap between images), the number of user images matched to the point-cloud would also increase in a linear fashion.

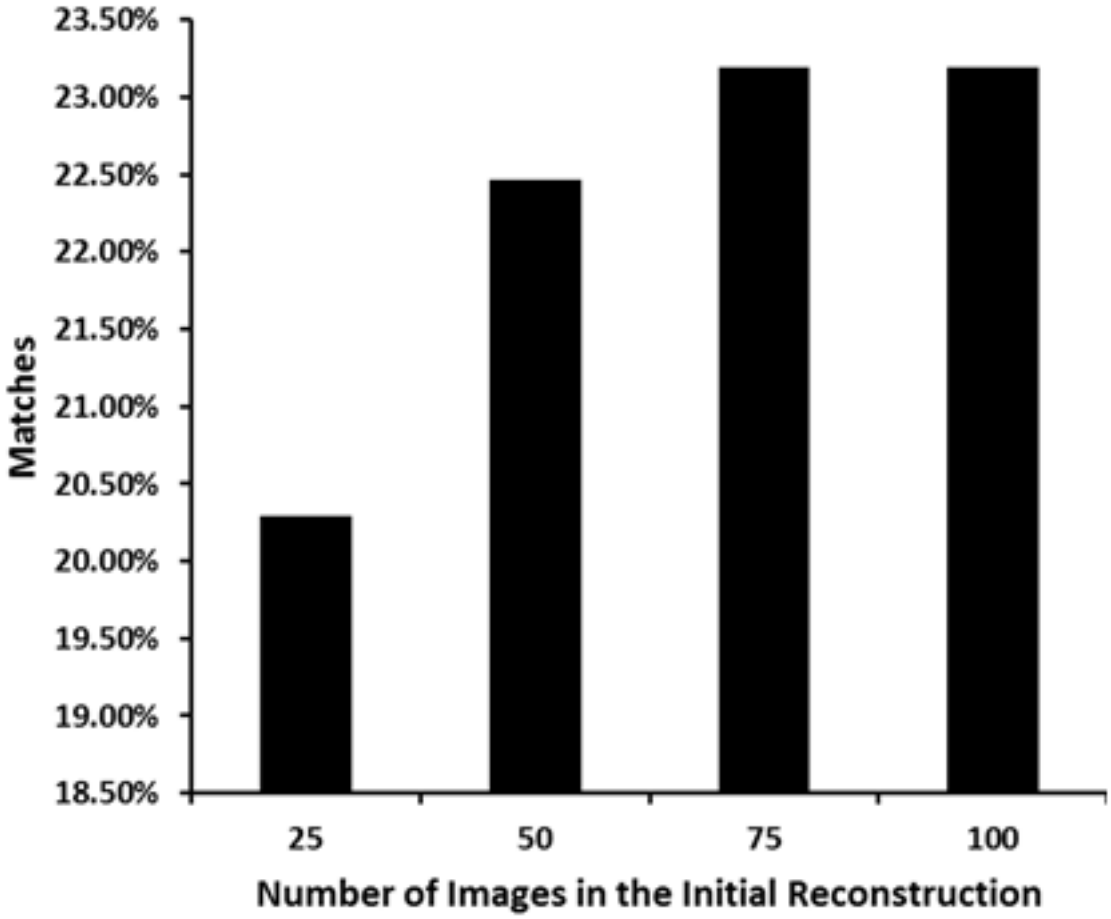
Results First Run

Upon executing our experiments for HD4AR for the first time, the results returned promising for an inital implementation, as shown in Figure 5.1. Using an initial reconstruction image set size of 25 netted a matching percentage of about 20% of photos. Increasing the size of the original set provided improvements, but seemed to reach a threshold of about 23% of pictures being matched.

Three key conclusions can be derived from these results:

1. HD4AR can successfully localize a user based solely on images taken from a mobile

Figure 5.1: Inital Test Run Results



device. While it is not guaranteed that the first image submitted for localization will succeed, this experiment has proven that it is indeed possible to sucessfully localize a user using no external modifications to an area.

2. There is a threshold for the number of base images used in reconstruction, after which, successful localization no longer experiences large gains. From our initial experiments, we estimate the threshold to lie at around 50 images of saturation. This phenomenon coincides exactly with the trend of average matching points per image in initial bundle reconstruction. Therefore we can use this statistic to decide how many base images are required to bootstrap the system to provide reliable localization. Practically this threshold number may vary based on the texture of building material (e.g., concrete vs. steel) or the distance from camera to the building. The reason for this threshold is believed to lie in the idea that there is a limited number of feature points for a given area. Thus, despite adding additional images to the base reconstruction, once you have enough images to reflect all available natural feature points in the area, subsequent images will not actually add any complexity to the physical model, but rather just redundantly project the same features already modeled.

3. Engineers may need to take multiple photos before achieving a successful localization. Based on our results, this will average out to around 3-4 images using this version of the system.

5.2.2 Execution Time Analysis

In addition to how effective the system is at achieving user localization, we were also interested in the computational time required to localize the user in 3D with HD4AR.

Setup

These experiments were run concurrently with those detailed in Experiment 1. For each image run through the localization process, the time expended was recorded. Then, for each 3D reconstruction, the time spent to localize each image (whether successful or not), was averaged out to determine the overall time estimate for localizing an image at a given density of the 3D reconstruction.

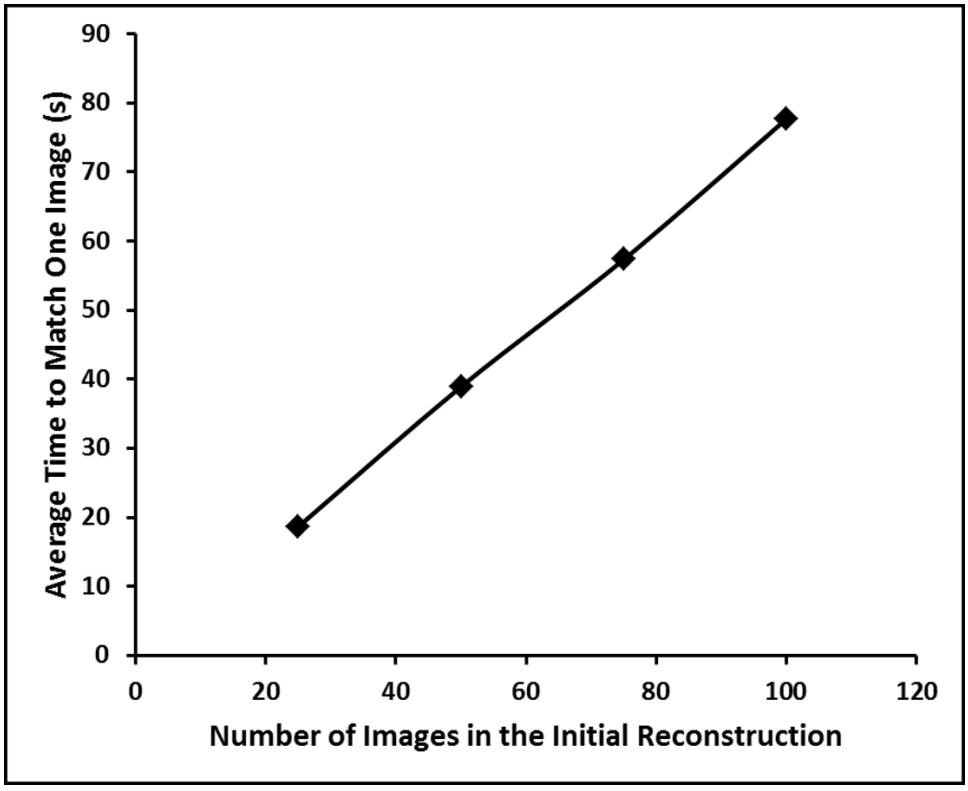
Hypothesis

Since the localization process is based around matching the features of a single image against the features of all images in the initial 3D reconstruction, we expected that the time required to localize an image to a pre-existing set would scale linearly based on the number of images in the initial set.

Results

As can be seen in Figure 5.2, the hypothesis for the linear growth of image adding times proved correct. As the size of the initial 3D reconstruction increases, so does the time

Figure 5.2: Average time needed to localize an image from iPhone 4, image resolution: 960x720



needed to compare the new image against the original set. In addition, overall execution time depends on the image resolution as well. For the current implementation of HD4AR, each photograph takes between 10 to 80 seconds to be localized. In future work, we plan to investigate methods of limiting the scope of the image matching using the coarse location information available in mobile phone to speed image localization.

Chapter 6

Related Work

There have been several previous research efforts to associate sematic data with the physical environment and contextual data with the user. The following section compares these techniques with HD4AR, by classifying them by the type of technology used.

6.1 Tagging Construction Objects with Sensors.

[7],[8],[9] have used GPS systems to track the locations of heavy equipment and provide map-based displays of equipment locations. The integration of GPS with cyber decision support systems has been shown to help engineers to safely monitor construction processes [9],[10]. Similarly, active RFID tags mounted on specially designed helmets can track workers and machines positions [36] and could be used to provide location estimation. Attaching physical sensors or markers to construction objects has several limitations. Tracking building

elements such as joints, columns, and beams, is impractically expensive due to their multitude [14],[15]. Furthermore, the sensor may have to be moved and reinstalled in the event that project progress results in the sensor being covered or hidden. The constant evolution of the construction site and maintenance of tags makes tracking items other than equipment challenging and to some extent impractical [14]. Cost effective tags, such as barcodes and RFID tags [3], [15] or fiducial markers [16],[17] can be used to associate a cyber-identity with a physical object. These approaches do not require charging sensors or significant cost. However, they are still subject to the same maintenance challenges of being exposed to a construction site. They may be covered by dust, finishing materials, or hidden and must be continually relocated and/or reapplied. In the case of fiducials, the large number of construction elements introduces the additional problem of running out of unique identifiers with which to tag. HD4AR does not require construction practitioners to tag construction objects with sensors, thus eliminated the problems associated with them. Rather than applying external sensors to construction elements, HD4AR demonstrates localization using only what is already present the natural features of the elements themselves.

6.2 Wireless Sensors.

GPS and other wireless sensor solutions in mobile devices suffer from the noise present in sensor data [20]. For applications such as building construction, which require high levels of accuracy for the purpose of safety and efficiency, this level of error can be catastrophic.

Moreover, inside of buildings under construction, GPS signals may be significantly reduced or eliminated. For indoor environments, the usage of GPS indoors, WLAN, and UWB is proposed. The main drawback of indoor tracking systems is their reliance on pre-installed network infrastructure making them unsustainable in on-going construction and facility environments [1]. There has recently been proposed an inertial measurement unit Personal Dead Reckoning (PDR) system [31] which is accurate for tracking smooth site walkthrough, nonetheless its accuracy degrades with path complexity, time spent indoors, and abrupt movements such as running. Unlike image-based tracking in HD4AR this system requires a user to wear a PDR sensor. The PDR sensor also needs RTK-GPS for calibration, which inconveniences the user to walk outside.

6.3 Computer Vision.

Due to the price of digital cameras, computer vision is an affordable alternative to other 3D reconstruction technologies, such as laser scanners [2],[28],[33]. Computer vision can be used to create cyber-physical models that can map cyber-information to physical objects in photographs, without the need of installing tags or sensors. Current applications have mostly focused on site status and data archiving [29], which can be performed offline. Current computer vision implementations have some drawbacks, such as complexity of handling large data sets, lack of mobile interfaces, and lack of techniques for querying information on demand. This prevents current computer vision techniques from being used for dynamic in-

formation retrieval on construction sites. HD4AR aims to address these drawbacks. Another advantage of computer vision is that taking images on a construction site is now common practice [12].

Chapter 7

Conclusion

The construction industry stands to gain much through the integration of cyber-information into regular site operations. The HD4AR system was designed with the intent of bringing augmented reality to construction sites, to better facilitate the use of cyber-information around worksites. It takes vital project information, (such as expected quality of building materials or location of elements, project schedule, and cost) which traditionally has been difficult to access on a jobsite (yet remains very important to project specification and completion), and makes it mobile, accessible, and possible to visualize over real-world imagery. HD4AR works using visual features as the basis for user localization, using Structure from Motion techniques to build and match a 3D geometric model from regular camera images. Users can use a smartphone outfitted with a camera, screen, and wireless communication to upload a captured image, localize it, determine relevant cyber information, and then overlay the returned cyber information on the physical objects to which it pertains. From my

work developing mobile Augmented Reality for construction sites, I learned the following important lessons:

1. Localization of Users can be Accomplished Using Only 2D Images as Input.

   HD4AR demonstrates that it is possible to develop augmented reality systems without the need for external modifications, signal tracking, or GPS/wireless tracking modules, avoiding the pitfalls inherent to such techniques on construction sites.

2. Image-based Queries can be Performed within a Reasonable Amount of Time, but Increases as Base Image Set Grows As shown in the results, localization based upon an image-based query takes only around 40 seconds using 50 photographs for the base point cloud. However, as the base set increases in size, the time needed to localize a photo also increases. Thus, there exists a tradeoff between the size of the base set and the time needed to localize. While an increased base set does provide some improved measure of localization potential (due to including more images, and thus more features to be matched against), a side effect is that the time necessary for localization is increased linearly with the increase in base images.

3. Reliability of the Localization is Promising with Small Number of Initial Images Results from our experiments show that, given the current implementation of HD4AR, localization is indeed possible, but may require multiple executions before success. For this reason, a key component of future work is increasing the potential for localization. Some work has already been completed towards this goal, with early results showing

up to 97% chance of localization, a massive increase over the current implementation. Therefore, with small effort bootstrapping the system, reliable mobile localization and augmentation is possible.

Chapter 8

Bibliography

[1] Y. Chen, and J.M. Kamara, "A framework for using mobile computing for information management on construction sites," Journal of Automation in Construction, vol. 20, 2011, p. 776.

[2] M. Golparvar-Fard, F. Pea-Mora, and S. Savarese, "D4AR-A 4-dimensional augmented reality model for automating construction progress data collection, processing and communication," Journal of Information Technology in Construction (ITcon), Special Issue Next Generation Construction IT: Technology Foresight, Future Studies, Roadmapping, and Scenario Planning, vol. 14, 2009, p. 129-153.

[3] R. Navon and R. Sacks, "Assessing research issues in automated project performance control (APPC)," Automation in construction, vol. 16, 2007, p. 474-484.

[4] ENR (Engineering News Record), "Gadgets Are Hot And More Are At Hand", McGraw-Hill.

[5] D. Wagner; D. Schmalstieg, "History and Future of Tracking for Mobile Phone Augmented Reality," Ubiquitous Virtual Reality, 2009. ISUVR '09. International Symposium on , pp.7-10, 8-11 July 2009

[6] P. Miranda, "Hybrid 4-Dimensional Augmented Reality". Pervasive and Embedded Computing and Communication Systems, 2012, Conference on . February 2012.

[7] L. Bernold and X. Huang, "CAD-integrated real-time control for robotic excavation and pipe-Laying: Development and field testing," Final Report Draft, FHWA Office of Advanced Research, 1996.

[8] J. Seo, C.T. Haas, K. Saidi, and S.V. Sreenivasan, "Graphical control interface for construction and maintenance equipment," Journal of construction engineering and management, vol. 126, 2000, p. 210.

[9] C.H. Caldas, D.G. Torrent, and C.T. Haas, "Integration of automated data collection technologies for real-time field materials management," Proceedings of the 21st ISARC, Jeju, Korea, 2004.

[10] M.Y. Cheng and J.C. Chen, "Integrating barcode and GIS for monitoring construction progress," Automation in Construction, vol. 11, 2002, p. 23-33.

[11] V.R. Kamat and J.C. Martinez, others, "Dynamic 3D visualization of articulated construction equipment," Journal of computing in civil engineering, vol. 19, 2005, p. 356.

[12] L. Soibelman, J. Wu, C. Caldas, I. Brilakis, and K.Y. Lin, "Management and analysis of unstructured construction data types," Advanced Engineering Informatics, vol. 22, 2008, p. 15-27.

[13] F. Bosch, "Automated recognition of 3D CAD model objects in laser scans and calculation of as-built dimensions for dimensional compliance control in construction," Advanced engineering informatics, vol. 24, 2010, p. 107-118.

[14] S. Kiziltas, others, "Technological assessment and process implications of field data capture technologies for construction and facility/infrastructure management", ITcon, 2008.

[15] E.J. Jaselskis and T. El-Misalami, "Implementing radio frequency identification in the construction process," Journal of Construction Engineering and Management, vol. 129, 2003, p. 680-688.

[16] D.J. Moseley, E.A. White, K.L. Wiltshire, T. Rosewall, M.B. Sharpe, J.H. Siewerdsen, J.P. Bissonnette, M. Gospodarowicz, P. Warde, and C.N. Catton, others, "Comparison of localization performance with implanted fiducial markers and cone-beam computed tomography for on-line image-guided radiotherapy of the prostate," International Journal of Radiation Oncology* Biology* Physics, vol. 67, 2007, p. 942-953.

[17] M. Fiala, "ARTag, a fiducial marker system using digital techniques," 2005.

[18] R.T. Azuma, others, "A survey of augmented reality," Presence-Teleoperators and Virtual Environments, vol. 6, 1997, p. 355-385.

[19] D. Schmalstieg and D. Wagner, "Experiences with handheld augmented reality," Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007, p. 1-13.

[20] J.B. Gotow, K. Zienkiewicz, J. White, and D.C. Schmidt, "Addressing Challenges with Augmented Reality Applications on Smartphones," Mobile Wireless Middleware, Operating Systems, and Applications, 2010, p. 129-143.

[21] P.A. Gorry, "General least-squares smoothing and differentiation by the convolution (Savitzky-Golay) method," Analytical Chemistry, 62, 1990, p. 570-573.

[22] H.H. Madden, "Comments on the Savitzky-Golay convolution method for least-squares-fit smoothing and differentiation of digital data," Analytical Chemistry, vol. 50, 1978, p. 1383-1386.

[23] T. Saramki, "Finite impulse response filter design," Handbook for Digital Signal Processing, 1993, p. 155.

[24] Y. Neuvo, D. Cheng-Yu, and S. Mitra, "Interpolated finite impulse response filters," Acoustics, Speech and Signal Processing, IEEE Transactions on, vol. 32, 1984, p. 563-570.

[25] M. Golparvar-Fard, S. Savarese, and F. Pe na-Mora, "Automated Model-Based Recognition of Progress Using Daily Construction Photographs and IFC-Based 4D Models," 2010.

[26] O. Abudayyeh, "Audio/visual information in construction project control," Advances in Engineering Software, vol. 28, 1997, p. 97-101.

[27] M. Golparvar-Fard, F. Pena-Mora, and S. Savarese, "Integrated sequential as-built and as-planned representation with D4AR – 4 dimensional augmented reality - tools in support of decision-making tasks in the AEC/FM industry," ASCE Journal of Construction Engineering and Management, 2011.

[28] Y. Furukawa, "High-fidelity image-based modeling," University of Illinois at Urbana-Champaign, 2008.

[29] P. Newton, K. Hampson, and R. Drogemuller, "1 Transforming the built environment through construction innovation," Technology, Design and Process Innovation in the Built Environment, 2009, p. 1.

[30] N. Snavely, S. M. Seitz, and R. Szeliski. "Photo tourism: exploring photo collections in 3D". In ACM SIGGRAPH 2006 Papers (SIGGRAPH '06). ACM, New York, NY, USA, 835-846, 2006.

[31] Akula, M., Dong, S., Kamat, V.R., Ojeda, L., Borrell, A., and Borenstein, J. (2011). "Integration of Infrastructure Based Positioning Systems and Inertial Navigation for

Ubiquitous Context-Aware Engineering Applications", Advanced Engineering Informatics, ICCCBE 2010 Special Issue on Computing in Civil and Building Engineering, Elsevier Science, New York, NY.

[32] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints" University of British Columbia Vancouver, B.C., Canada . 2004

[33] M. Golparvar-Fard, F.,Pea-Mora, and S. Savarese "Automated model-based progress monitoring using unordered daily construction photographs and IFC as-planned models." ASCE Journal of Computing in Civil Engineering, 2012. `http://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000205`.

[34] Android Asset Studio, "Device Frame Generator" `http://android-ui-utils.googlecode.com/hg/asset-studio/dist/device-frames.html`

[35] J.-Y. Bouguet, "Camera calibration toolbox" `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`

[36] J., Teizer, B.S., Allread, C. E., Fullerton, and J., Hinze (2010). "Autonomous Pro-Active Real-time Construction Worker and Equipment Operator Proximity Safety Alert System". Journal of Automation in Construction, Elsevier, vol. 19, no. 5, p. 630-640.

Appendix A

Server Code

A.1 ImageAdder.sh

```
1 #!/bin/bash
2 IMAGE=$1
3 LIST=$2
4 BUNDLE=$3
5 OUTPUT=$4
6 BUNDLER=$5
7
8 echo "[Extracting Focal Length]"
9 echo $IMAGE"jpg" > $OUTPUT"/temp.txt"
```

```
10 $BUNDLER"/bin/extract_focal.pl" $OUTPUT"/temp.txt"
11 rm -f $OUTPUT"/temp.txt"
12
13 echo "[Extracting Features]"
14 if [ ! -e $IMAGE"key.gz" ];
15 then
16 mogrify -format pgm $IMAGE"jpg"
17 $BUNDLER"/bin/sift" < $IMAGE"pgm" > $IMAGE"key"
18 rm $IMAGE"pgm"
19 gzip -f $IMAGE"key"
20 fi
21
22 echo "[Adding Image to Bundle]"
23 $BUNDLER"/bin/bundler" $LIST --bundle $BUNDLE --add_images $OUTPUT
      "/prepare/list.txt" > $OUTPUT"/add_out"
```

A.2 HD4ARServer.java

```
1 package org.miranda.hd4ar;
2 import org.mortbay.jetty.Server;
3 import org.mortbay.jetty.webapp.WebAppContext;
```

```
4
5 public class HD4ARServer extends Server{
6
7   public HD4ARServer() throws Exception{
8     super(8888);
9
10    WebAppContext webapp = new WebAppContext();
11          webapp.setContextPath("/");
12          webapp.setWar("./war");
13          this.setHandler(webapp);
14
15    this.start();
16    this.join();
17  }
18
19  public static void main(String[] args) {
20    try {
21      new HD4ARServer();
22    } catch(Exception e){
23      e.printStackTrace();
24    }
```

```
25    }
26
27 }
```

A.3 ImageServlet.java

```
 1 package org.miranda.hd4ar.servlet;
 2
 3 import java.io.File;
 4 import java.io.FileOutputStream;
 5 import java.io.IOException;
 6 import java.io.InputStream;
 7 import java.util.UUID;
 8 import java.util.logging.Logger;
 9
10 import javax.servlet.ServletException;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14
15 import org.apache.commons.fileupload.FileItemIterator;
```

```
16 import org.apache.commons.fileupload.FileItemStream;
17 import org.apache.commons.fileupload.servlet.ServletFileUpload;
18 import org.miranda.hd4ar.ImageHandler;
19
20
21 public class ImageServlet extends HttpServlet {
22
23   private static final long serialVersionUID = 1L;
24   private static final Logger log = Logger.getLogger(ImageServlet.
       class
25       .getName());
26
27   public void doPost(HttpServletRequest req, HttpServletResponse
       res)
28       throws ServletException, IOException {
29     try {
30
31       System.out.println("Standing by...");
32       String name = null;
33
34       ServletFileUpload upload = new ServletFileUpload();
```

```
35        res.setContentType("text/plain");
36
37        File file = null; //= new File("./",fname);
38
39
40        FileItemIterator iterator = upload.getItemIterator(req);
41        while (iterator.hasNext()) {
42          FileItemStream item = iterator.next();
43          InputStream stream = item.openStream();
44
45          if (item.isFormField()) {
46            log.warning("Got a form field: " + item.getFieldName());
47
48          } else {
49            log.warning("Got an uploaded file: " + item.getFieldName
                ()
50               + ", name = " + item.getName());
51
52            name = "upload_" + randomImageName() + "_" + item.
                getName();
53            file = new File("./",name);
```

```
54            FileOutputStream fout = new FileOutputStream(file);
55            byte[] buff = new byte[2048];
56            int read = 0;
57            while((read = stream.read(buff)) != -1){
58              fout.write(buff,0,read);
59            }
60
61          }
62
63        }
64        String s = "";
65
66        ImageHandler ih = new ImageHandler();
67        if(file != null){
68          System.out.println("Handling Image");
69          s = ih.handleImage(file);
70        }
71        if(name != null) {
72          System.out.println("Printing back to client: \n" + s);
73          res.setContentType("text/plain");
74          res.getWriter().write(s);
```

```
75        } else {
76          writeError(res, "No image was found in the POST!");
77        }
78
79      } catch (Exception ex) {
80        throw new ServletException(ex);
81      }
82    }
83
84    public String randomImageName() {
85      return UUID.randomUUID().toString();
86    }
87
88    public void writeError(HttpServletResponse res, String error)
89        throws IOException {
90      res.setContentType("text/plain");
91      res.getWriter().write(error);
92    }
93 }
```

A.4 ImageHandler.java

```
1 package org.miranda.hd4ar;
2 import java.awt.image.BufferedImage;
3 import java.io.BufferedReader;
4 import java.io.DataInputStream;
5 import java.io.File;
6 import java.io.FileInputStream;
7 import java.io.IOException;
8 import java.io.InputStreamReader;
9
10 import javax.imageio.ImageIO;
11 public class ImageHandler {
12
13   public String handleImage(File f) throws IOException{
14     if(f.exists()){
15       try {
16         long time = System.currentTimeMillis();
17         String line = "";
18
19         String category;
20         BufferedReader b = new BufferedReader(new
               InputStreamReader(
```

```
21              new DataInputStream(new FileInputStream("HD4ARServer.
                    conf"))));
22
23          String pathtobundler = "";
24          String listfile = "";
25          String datafile = "";
26          String outfile = "";
27          boolean testing = false;
28
29          while((line = b.readLine()) != null){
30            int i = line.indexOf(" ");
31            category = line.substring(0, (i != -1) ? i : line.length
                  ());
32            if(category.equalsIgnoreCase("pathtobundler")){
33              pathtobundler = line.substring(line.indexOf(" ")+1);
34              System.out.println("pathtobundler: " + pathtobundler);
35            } else if (category.equalsIgnoreCase("listfile")){
36              listfile = line.substring(line.indexOf(" ")+1);
37              System.out.println("listfile: " + listfile);
38            } else if (category.equalsIgnoreCase("datafile")){
39              datafile = line.substring(line.indexOf(" ")+1);
```

```
40              System.out.println("datafile: " + datafile);
41            } else if (category.equalsIgnoreCase("outfile")){
42              outfile = line.substring(line.indexOf(" ")+1);
43              System.out.println("outfile: " + outfile);
44            } else if (category.equalsIgnoreCase("testing")){
45              testing = true;
46              System.out.println("testing: " + testing);
47            }
48          }
49
50          b.close();
51
52
53
54          //DO NOT DELETE!!! COMMENTED OUT FOR TESTING ONLY!!!
55          if(!testing){
56            File local = new File("./");
57
58            String filename = f.getCanonicalPath();
59            String fn = filename.substring(0, filename.lastIndexOf("
                .")+1);
```

```
60
61          Process p = Runtime.getRuntime().exec("./ImageAdder.sh "
                + fn + " " + listfile + " " + outfile + " " + local.
                getCanonicalPath() + " " + pathtobundler);
62          BufferedReader waiter = new BufferedReader(new
                InputStreamReader(p.getInputStream()));
63          BufferedReader err = new BufferedReader(new
                InputStreamReader(p.getErrorStream()));
64
65          String w = "";
66          while((w = waiter.readLine()) != null){
67            System.out.println(w);
68          }
69          while((w = err.readLine()) != null){
70            System.out.println(w);
71          }
72
73        }
74
75        System.out.println("Image Adding Completed");
76        BufferedImage img = ImageIO.read(f);
```

```
77          System.out.println("Image Dimensions: " + img.getWidth() +
                "," + img.getHeight());
78
79          StringBuilder outString = new StringBuilder();
80
81
82          PointTranslator pt = new PointTranslator(outfile,"./bundle
               .added.out",datafile,img.getWidth(),img.getHeight());
83          if(pt.isValid()){
84            System.out.println("VALID");
85            pt.processPointsFile();
86            BufferedReader pointsReader = new BufferedReader(new
                 InputStreamReader(new DataInputStream(new
                 FileInputStream("./transformed_points.dat"))));
87            line = pointsReader.readLine();
88            int numObjects = Integer.valueOf(line);
89            outString.append(line + "\n");
90            System.out.println(line);
91            for(int i = 0; i < numObjects; i++){
92              for(int j = 0; j < 5; j++){
93                line = pointsReader.readLine();
```

```
94                System.out.println(line);
95                outString.append(line);
96                outString.append("\n");
97              }
98            }
99            pointsReader.close();
100
101           System.out.println("CAMERA POSITION");
102           System.out.println("===============");
103           double[][] cam_pos = pt.getCameraPosition();
104           for(int i = 0; i < cam_pos.length; i++){
105             for(int j = 0; j < cam_pos[i].length; j++){
106               System.out.print(cam_pos[i][j] + " ");
107             }
108             System.out.println();
109           }
110           System.out.println("===============");
111         }else{
112           outString.append(0);
113         }
114
```

```
115
116
117             float end = (System.currentTimeMillis() - time)/1000f;
118             System.out.println("Elapsed time: " + end + " seconds");
119             return outString.toString();
120         } catch (IOException e) {
121             e.printStackTrace();
122         }
123     }else{
124         System.out.println("No file found");
125     }
126     return "";
127   }
128
129 }
```

A.5 PointTranslator.java

```
1 package org.miranda.hd4ar;
2
3 import java.io.BufferedReader;
```

```
 4 import java.io.DataInputStream;
 5 import java.io.FileInputStream;
 6 import java.io.FileNotFoundException;
 7 import java.io.IOException;
 8 import java.io.InputStreamReader;
 9 import java.io.PrintStream;
10
11 import Jama.Matrix;
12
13 public class PointTranslator {
14
15   private String oldBundle_;
16   private String newBundle_;
17   private String dataPoints_;
18   private Matrix cameraRotation_;
19   private Matrix cameraTranslation_;
20   private Matrix pointTransform_;
21   private double focalLength_;
22   private double distortionOne_;
23   private double distortionTwo_;
24   private TempPoint[] oldPoints_;
```

```
25  private TempPoint[] newPoints_;
26  private boolean valid_;
27  private int maxX_;
28  private int maxY_;
29
30  /**
31   * @param bundle - Path to the old bundle.out file
32   * @param addbundle - Path to bundle.added.out, generated when a
         new image is added
33   * @param pointsfile - Path to dataPoints.dat
34   * @param width - Width of the image
35   * @param height - Height of the image
36   */
37  public PointTranslator(String bundle, String addbundle, String
       pointsfile,
38      int width, int height) {
39    oldBundle_ = bundle;
40    newBundle_ = addbundle;
41    dataPoints_ = pointsfile;
42    maxX_ = width/2;
43    maxY_ = height/2;
```

```
44      valid_ = this.initialize();
45      this.getTransform();
46    }
47
48    public boolean isValid(){
49      return valid_;
50    }
51
52    public double[][] getCameraPosition(){
53
54      Matrix invR = cameraRotation_.inverse();
55      Matrix negInvR = invR.times(-1.0);
56      Matrix mult = negInvR.times(cameraTranslation_);
57      return mult.getArray();
58    }
59
60
61    /*
62     * Read in "number" of TempPoints from bundle "bFile"
63     */
64    public TempPoint[] getPoints(String bFile, int number){
```

```
65      try {
66        TempPoint[] points = new TempPoint[number];
67        String line = "";
68        String[] div;
69        BufferedReader b = new BufferedReader(
70            new InputStreamReader(
71                new DataInputStream(
72                    new FileInputStream(bFile))));
73        line = b.readLine();
74        line = b.readLine();
75        div = line.split(" ");
76
77        int numCams = Integer.valueOf(div[0]);
78        int numPts = Integer.valueOf(div[1]);
79        if(numPts < number){
80          System.out.println("Less than " + number + " points");
81          return null;
82        }
83        /*
84         * Skip over Camera values
85         */
```

```
86        for(int i = 0; i < numCams; i++){
87          for(int j = 0; j < 5; j++){
88            line = b.readLine();
89          }
90        }
91        /*
92         * Select first "number" points
93         */
94        String position;
95        for(int i = 0; i < number; i++){
96          /*
97           * Read the point values
98           */
99          position = b.readLine();
100         b.readLine();
101         b.readLine();
102         double[] p = new double[3];
103         div = position.split(" ");
104         for(int j = 0; j < 3; j++){
105           p[j] = Double.valueOf(div[j]);
106         }
```

```
107           points[i] = new TempPoint(p);
108         }
109         return points;
110
111       } catch (FileNotFoundException e) {
112         e.printStackTrace();
113       } catch (IOException e){
114         e.printStackTrace();
115       }
116
117       return null;
118     }
119
120     private boolean initialize() {
121
122       String line = "";
123       String[] div;
124       try {
125         BufferedReader b = new BufferedReader(new InputStreamReader(
126             new DataInputStream(new FileInputStream(oldBundle_))));
127         line = b.readLine();
```

```
128         line = b.readLine();
129         div = line.split(" ");
130
131         int numCamsOld = Integer.valueOf(div[0]);
132
133         b.close();
134
135         b = new BufferedReader(new InputStreamReader(new
                DataInputStream(
136                 new FileInputStream(newBundle_))));
137         line = b.readLine();
138         line = b.readLine();
139         div = line.split(" ");
140
141         int numCamsNew = Integer.valueOf(div[0]);
142         if(numCamsNew == numCamsOld){
143           System.out.println("Camera was not added.");
144           return false;
145         }
146
147         for (int i = 1; i < numCamsNew; i++) {
```

```
148         for (int j = 0; j < 5; j++) {
149           b.readLine();
150         }
151       }
152       line = b.readLine();
153       div = line.split(" ");
154
155       focalLength_ = Double.valueOf(div[0]);
156       distortionOne_ = Double.valueOf(div[1]);
157       distortionTwo_ = Double.valueOf(div[2]);
158       System.out.println("Acquiring Camera data");
159       System.out.println(focalLength_ + " " + distortionOne_ + " "
              + distortionTwo_);
160
161       double[][] r = new double[3][3];
162       for (int i = 0; i < 3; i++) {
163         line = b.readLine();
164         div = line.split(" ");
165         for (int j = 0; j < 3; j++) {
166           r[i][j] = Double.valueOf(div[j]);
167           System.out.print(r[i][j] + " ");
```

```
168           }
169           System.out.println();
170         }
171         cameraRotation_ = new Matrix(r);
172
173         line = b.readLine();
174         div = line.split(" ");
175         double[] t = new double[3];
176         for (int i = 0; i < 3; i++) {
177           t[i] = Double.valueOf(div[i]);
178           System.out.print(t[i] + " ");
179         }
180         System.out.println();
181         cameraTranslation_ = new Matrix(t, 3);
182
183         b.close();
184
185         return true;
186       } catch (FileNotFoundException e) {
187         e.printStackTrace();
188       } catch (IOException e) {
```

```
189         e.printStackTrace();
190       }
191       return false;
192     }
193
194     /*
195      * Forms transformation matrix using points from original and
           added bundle files.
196      * In the event that the transform matrix is nonsingular (
           usually occurs when two
197      * selected points are very close together in 3D space), will
           select additional
198      * points to use.
199      */
200     private void getTransform() {
201
202       Matrix Am = new Matrix(12,12);
203       int count = 1;
204       while(!Am.lu().isNonsingular()){
205         System.out.println("Count: " + count);
206         oldPoints_ = this.getPoints(oldBundle_, 4*count);
```

```
207
208         double[] pt1 = oldPoints_[0+4*(count-1)].getPosition();
209         double[] pt2 = oldPoints_[1+4*(count-1)].getPosition();
210         double[] pt3 = oldPoints_[2+4*(count-1)].getPosition();
211         double[] pt4 = oldPoints_[3+4*(count-1)].getPosition();
212
213         for(int i = 0; i < 4; i++){
214           System.out.println("Old Point " + i);
215           for(int j = 0; j < 3; j++){
216             System.out.print(oldPoints_[i+4*(count-1)].getPosition()
                  [j] + " ");
217           }
218           System.out.println();
219         }
220
221         double[][] A = { { pt1[0], pt1[1], pt1[2], 1, 0, 0, 0, 0, 0,
              0, 0, 0 },
222             { 0, 0, 0, 0, pt1[0], pt1[1], pt1[2], 1, 0, 0, 0, 0 },
223             { 0, 0, 0, 0, 0, 0, 0, 0, pt1[0], pt1[1], pt1[2], 1 },
224             { pt2[0], pt2[1], pt2[2], 1, 0, 0, 0, 0, 0, 0, 0, 0 },
225             { 0, 0, 0, 0, pt2[0], pt2[1], pt2[2], 1, 0, 0, 0, 0 },
```

```
226             { 0, 0, 0, 0, 0, 0, 0, 0, pt2[0], pt2[1], pt2[2], 1 },
227             { pt3[0], pt3[1], pt3[2], 1, 0, 0, 0, 0, 0, 0, 0, 0 },
228             { 0, 0, 0, 0, pt3[0], pt3[1], pt3[2], 1, 0, 0, 0, 0 },
229             { 0, 0, 0, 0, 0, 0, 0, 0, pt3[0], pt3[1], pt3[2], 1 },
230             { pt4[0], pt4[1], pt4[2], 1, 0, 0, 0, 0, 0, 0, 0, 0 },
231             { 0, 0, 0, 0, pt4[0], pt4[1], pt4[2], 1, 0, 0, 0, 0 },
232             { 0, 0, 0, 0, 0, 0, 0, 0, pt4[0], pt4[1], pt4[2], 1 },
233
234         };
235         System.out.println("Array constructed");
236         for(int i = 0; i < 12; i++){
237           for(int j = 0; j < 12; j++){
238             System.out.print(A[i][j] + " ");
239           }
240           System.out.println();
241         }
242         Am = new Matrix(A);
243         count++;
244       }
245
246       count--;
```

```
247
248     System.out.println(4*count);
249     newPoints_ = this.getPoints(newBundle_,4*count);
250     for(int i = 0; i < newPoints_.length; i++){
251       double[] temp = newPoints_[i].getPosition();
252       for(int j = 0; j < temp.length; j++){
253         System.out.print(temp[j] + " ");
254       }
255       System.out.println();
256     }
257
258     double[] B = new double[12];
259     for (int i = 0; i < 4; i++) {
260       System.out.println("Using Point " + (4*(count-1)+i));
261       double[] pt = newPoints_[i+4*(count-1)].getPosition();
262       for (int j = 0; j < 3; j++) {
263         B[i * 3 + j] = pt[j];
264         System.out.print(pt[j] + " ");
265       }
266       System.out.println();
267     }
```

```
268
269     Matrix Bm = new Matrix(B, 12);
270
271     Matrix Xm = Am.solve(Bm);
272     double[][] X = Xm.getArrayCopy();
273
274     double[][] pt = {
275         {0,0,0,0},
276         {0,0,0,0},
277         {0,0,0,0},
278         {0,0,0,1}
279     };
280     for(int i = 0; i < 3; i++){
281       for(int j = 0; j < 4; j++){
282         pt[i][j] = X[i*4+j][0];
283       }
284     }
285     pointTransform_ = new Matrix(pt);
286
287 }
288
```

```
289  /*
290   * Using the transform matrix, translates a point position into
291   * the addBundle's 3D space
292   */
293  private double[] transformPoint(double[] pos) {
294    double[] oldPt = { pos[0], pos[1], pos[2], 1 };
295    System.out.println("OldPt: " + pos[0] + " " + pos[1] + " " +
           pos[2]);
296    Matrix opm = new Matrix(oldPt, 4);
297
298    Matrix npm = pointTransform_.times(opm);
299    double[][] newPt = npm.getArrayCopy();
300
301    double[] pt = { newPt[0][0], newPt[1][0], newPt[2][0] };
302    return pt;
303  }
304
305  /*
306   * Iterates through polygons of interest, and determines if they
           are within view.
307   */
```

```
308  public void processPointsFile(){
309    String line = "";
310    try {
311      PrintStream out = new PrintStream("processed_points.dat");
312      BufferedReader b = new BufferedReader(new InputStreamReader(
313          new DataInputStream(new FileInputStream(dataPoints_))));
314      line = b.readLine();
315      int numObjects = Integer.valueOf(line);
316      int count = 0;
317      for(int i = 0; i < numObjects; i++){
318        boolean withinView = false;
319        line = b.readLine();
320        System.out.println("type: " + line);
321        int type = Integer.valueOf(line);
322        line = b.readLine();
323        System.out.println("points: " + line);
324        String[] points = line.split(",");
325        int numPts = Integer.valueOf(points[0]);
326        System.out.println("NumPts: " + points[0] + " " + numPts);
327        StringBuilder ptsLine = new StringBuilder();
328        for(int j = 1; j < numPts + 1; j++){
```

```
329         double[] pos = new double[3];
330         String[] aPt = points[j].split(" ");
331         System.out.println(points[j]);
332         for(int k = 0; k < 3; k++){
333           pos[k] = Double.valueOf(aPt[k]);
334         }
335         System.out.println("POSITION(" + pos.length + "): " +
              pos[0] + " " + pos[1] + " " + pos[2]);
336         double[] newPos = transformPoint(pos);
337         System.out.println("NEWPOSITION(" + newPos.length + "):
              " + newPos[0] + " " + newPos[1] + " " + newPos[2]);
338         double[] flatPos = point3Dto2D(newPos);
339         System.out.println("FLATPOSITION(" + flatPos.length + ")
              : " + flatPos[0] + " " + flatPos[1]);
340         if(withinBounds(flatPos)){
341           withinView = true;
342         }
343         for(int k = 0; k < 2; k++){
344           ptsLine.append(flatPos[k]);
345           ptsLine.append(" ");
346         }
```

```
347         }
348         String pL = ptsLine.toString();
349         line = b.readLine(); //color of polygon -- keep as is
350         String color = line;
351         line = b.readLine(); //indices of polygon -- keep as is
352         String indices = line;
353         line = b.readLine();
354         String id = line;
355         if(withinView){   //if at least one point of the polygon
              is visible
356           out.println(type);          //type
357           out.print(numPts + " ");     //# x1 y1 x2 y2 x3 y3 ...
358           out.println(pL.trim());
359           out.println(color);         //a b c d
360           out.println(indices);       // 0 1 0 2 1 3 0 4 1 5 2 6 3 7
                4 5 4 6 5 7 6 7
361           out.println(id);          // Whatever associated info is
                available
362           count++;
363         }else{
364           System.out.println("NOT WITHIN VIEW!!!");
```

```
365             }
366
367         }
368         b.close();
369         out.close();
370
371         out = new PrintStream("transformed_points.dat");
372         b = new BufferedReader(new InputStreamReader(
373             new DataInputStream(new FileInputStream("
                    processed_points.dat"))));
374         out.println(count);
375         while((line = b.readLine()) != null){
376           out.println(line);
377         }
378
379       } catch (FileNotFoundException e) {
380         e.printStackTrace();
381       } catch (IOException e) {
382         e.printStackTrace();
383       }
384   }
```

```
385
386   private boolean withinBounds(double[] coords){
387     if(coords.length != 2){
388       System.out.println("Incorrect length " + coords.length);
389       return false;
390     }
391     int x = maxX_ - Math.abs((int)coords[0]);
392     int y = maxY_ - Math.abs((int)coords[1]);
393     if(x > 0 && y > 0){
394       return true;
395     }
396     return false;
397   }
398
399   /*
400    * Helper function which finds the norm(maximum) of an array.
401    */
402   private double norm(double[] p) {
403     double max = p[0];
404     for (int i = 1; i < p.length; i++) {
405       if (p[i] > max) {
```

```
406           max = p[i];
407         }
408       }
409       return max;
410     }
411
412     /*
413      * r(p) is the function that captures the radial distortion: r(p
           ) = 1.0 + k1
414      * * ||p||^2 + k2 * ||p||^4 + ... k1 and k2 are radian
           distortion
415      * coefficients that are captured in the SfM process and are
           listed right
416      * after f in the out file.
417      */
418     private double function_r(double[] p) {
419       double rp = 0;
420       rp = 1.0d + distortionOne_ * Math.pow(norm(p), 2.0d) +
             distortionTwo_
421           * Math.pow(norm(p), 4.0d);
422       return rp;
```

```
423   }
424
425   /*
426    * Transform 3D position into 2D position in camera view.
427    * This implementation uses JAMA Matrix classes.
428    */
429   public double[] point3Dto2D(double[] XYZ) {
430     double[] VW = new double[2];
431     /*
432      * 1.1 P = R * X + t Conversion from world to camera
           coordinates - where
433      * X is the 3D point (x,y,z), R is the rotation matrix of the
           camera
434      * (3x3), and t is the translation matrix (3x1)
435      */
436     Matrix Xm = new Matrix(XYZ, 3);
437     // Matrix RXm = cameraRotation_.times(Xm);
438     Matrix Pm = (cameraRotation_.times(Xm)).plus(
          cameraTranslation_);
439
440     /*
```

```
441      * 1.2. p = -P / P.z = P * (-1/P.z) Perspective division -
             This is due
442      * to homogeneous coordinate system conversion to Cartesian (
             homogeneous
443      * is used to capture the concept of infinity) P.z is the
             third
444      * coordinate of the point P which was the outcome of step 1.1
445      */
446     Matrix PZm = Pm.times(-1 / Pm.getArray()[2][0]);
447
448     /*
449      * 1.3 p' = f * r(p) * p Conversion to pixel coordinates - f
             is the
450      * focal length and p is the outcome of step 1.2
451      */
452     double[] p = new double[3];
453     for (int i = 0; i < 3; i++) {
454       p[i] = PZm.getArray()[i][0];
455     }
456     double factor = function_r(p) * focalLength_;
457
```

```
458     VW[0] = factor * p[0];
459     VW[1] = factor * p[1];
460
461     return VW;
462   }
463 }
```

A.6 TempPoint.java

```
1 package org.miranda.hd4ar;
2
3 public class TempPoint {
4
5   private double[] position_;
6
7   public TempPoint(double[] position){
8     position_ = position;
9   }
10
11  public double[] getPosition(){
12    double[] p = position_.clone();
```

```
13        return p;
14    }
15
16 }
```

Appendix B

Client Code

B.1 AndroidManifest.xml

```
1<?xml version="1.0" encoding="utf-8"?>
2<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
3   package="org.miranda.hd4ar.client"
4   android:versionCode="1"
5   android:versionName="1.0">
6  <application android:label="@string/app_name" android:theme="
       @android:style/Theme.NoTitleBar.Fullscreen" android:icon="
       @drawable/hd4aricon">
```

```
7     <activity android:name="org.miranda.hd4ar.client.
          UploadPageActivity"
8         android:label="@string/intent_name"
9         android:screenOrientation="landscape">
10        <intent-filter>
11          <action android:name="android.intent.action.SEND" />
12          <category android:name="android.intent.category.DEFAULT" /
                >
13          <data android:mimeType="image/*" />
14        </intent-filter>
15    </activity>
16    <activity android:name="org.miranda.hd4ar.client.
          NetworkCommunicationActivity">
17    </activity>
18    <activity  android:name="org.miranda.hd4ar.client.
          PolygonDisplayActivity"
19        android:screenOrientation="landscape">
20    </activity>
21  </application>
22  <uses-sdk android:minSdkVersion="8" />
```

```
23   <uses-permission android:name="android.permission.INTERNET"></
         uses-permission>
24</manifest>
```

B.2 UploadPageActivity.java

```
 1 package org.miranda.hd4ar.client;
 2
 3 import java.io.InputStream;
 4
 5 import org.apache.commons.io.IOUtils;
 6
 7 import android.app.Activity;
 8 import android.content.ContentResolver;
 9 import android.content.Intent;
10 import android.database.Cursor;
11 import android.net.Uri;
12 import android.os.Bundle;
13 import android.provider.OpenableColumns;
14 import android.util.Log;
15 import android.view.View;
```

```
16 import android.view.View.OnClickListener;
17 import android.widget.Button;
18 import android.widget.EditText;
19 import android.widget.Toast;
20
21
22
23 public class UploadPageActivity extends Activity implements
       OnClickListener {
24
25   private EditText server_;
26   private Button close_;
27   private Button upload_;
28
29   private byte[] data;
30   private Uri stream;
31   private String name;
32
33   private final static String APP = UploadPageActivity.class.
         getSimpleName();
34
```

```
35  public void onCreate(Bundle savedInstanceState) {
36    super.onCreate(savedInstanceState);
37
38    setContentView(R.layout.uploadpagelayout);
39    server_ = (EditText) findViewById(R.id.server);
40    upload_ = (Button) findViewById(R.id.UploadButton);
41    close_ = (Button) findViewById(R.id.CloseButton);
42    close_.setOnClickListener(this);
43    upload_.setOnClickListener(this);
44
45    Intent intent = this.getIntent();
46
47    try {
48      if (intent.hasExtra(Intent.EXTRA_STREAM)) {
49
50        String type = intent.getType();
51        stream = (Uri) intent.getParcelableExtra(Intent.
              EXTRA_STREAM);
52        if (stream != null && type != null) {
53          Log.i(APP, "Got a stream for an image...");
54          ContentResolver contentResolver = getContentResolver();
```

```
55
56          String contentType = contentResolver.getType(stream);
57          Log.i(APP, "Image type: " + contentType);
58          int size = -1;
59          Cursor metadataCursor = contentResolver.query(stream,
60              new String[] { OpenableColumns.DISPLAY_NAME,
61                  OpenableColumns.SIZE }, null, null, null);
62          if (metadataCursor != null) {
63            try {
64              if (metadataCursor.moveToFirst()) {
65                name = metadataCursor.getString(0);
66                size = metadataCursor.getInt(1);
67              }
68            } finally {
69              metadataCursor.close();
70            }
71          }
72          Log.i(APP, "The size is: " + size);
73
74          if (name == null) name = stream.getLastPathSegment();
75
```

```
76            InputStream in = contentResolver.openInputStream(stream)
                  ;
77            data = IOUtils.toByteArray(in);
78
79          } else {
80            Log.i(APP, "Stream was null");
81            UploadPageActivity.this.finish();
82          }
83        } else {
84          Log.i(APP, "Invoked outside of send....");
85        }
86      } catch (Exception e) {
87        e.printStackTrace();
88      }
89
90  }
91
92  @Override
93  public void onClick(View v) {
94    if(v.getId()==close_.getId()){
95      UploadPageActivity.this.finish();
```

```
96      }else if(v.getId()==upload_.getId()){
97          Intent i = new Intent(this, NetworkCommunicationActivity.
                class);
98          i.putExtra("SERVER", server_.getText().toString());
99            i.putExtra("DATA", data);
100           i.putExtra("NAME", name);
101         UploadPageActivity.this.startActivity(i);
102         UploadPageActivity.this.finish();
103     }else{
104         Toast.makeText(this, "Unknown button press registered",
                Toast.LENGTH_SHORT).show();
105
106     }
107 }
108
109 }
```

B.3 uploadpagelayout.xml

```
1<?xml version="1.0" encoding="utf-8"?>
```

```
2<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
3    android:orientation="vertical"
4    android:layout_width="fill_parent"
5    android:layout_height="fill_parent"
6    >
7
8  <TextView android:id="@+id/TextView03" android:layout_width="
       wrap_content" android:layout_height="wrap_content"
       android:text="Sword Server Address"></TextView><EditText
       android:layout_height="wrap_content" android:layout_width="
       fill_parent" android:id="@+id/server" android:text="http:
       //10.0.2.2:8888/upload"></EditText>
9
10 <LinearLayout android:id="@+id/linearLayout1"
       android:layout_width="match_parent" android:layout_height="
       wrap_content">
11      <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="Upload
            " android:layout_weight=".50" android:id="@+id/
            UploadButton"></Button>
```

```
12        <Button android:layout_width="wrap_content"
              android:layout_height="wrap_content" android:id="@+id/
              CloseButton" android:text="Cancel" android:layout_weight=
              ".50"></Button>
13    </LinearLayout>
14
15    <ScrollView android:layout_width="match_parent"
          android:layout_height="wrap_content" android:id="@+id/
          scrollView1">
16      <TextView android:id="@+id/returnView" android:layout_width="
            wrap_content" android:layout_height="wrap_content"
            android:text="" android:isScrollContainer="true"></TextView
            >
17    </ScrollView>
18
19
20
21
22</LinearLayout>
```

B.4 NetworkCommunicationActivity.java

```
1 package org.miranda.hd4ar.client;
2
3 import java.io.BufferedReader;
4 import java.io.ByteArrayInputStream;
5 import java.io.IOException;
6 import java.io.InputStream;
7 import java.io.InputStreamReader;
8 import java.io.Reader;
9 import java.io.StringWriter;
10 import java.io.Writer;
11
12 import org.apache.http.HttpEntity;
13 import org.apache.http.HttpResponse;
14 import org.apache.http.client.ClientProtocolException;
15 import org.apache.http.client.HttpClient;
16 import org.apache.http.client.methods.HttpPost;
17 import org.apache.http.entity.mime.MultipartEntity;
18 import org.apache.http.entity.mime.content.InputStreamBody;
19 import org.apache.http.impl.client.DefaultHttpClient;
20
21 import android.app.Activity;
```

```
22 import android.content.Intent;
23 import android.os.AsyncTask;
24 import android.os.Bundle;
25 import android.util.Log;
26 import android.widget.LinearLayout;
27
28 public class NetworkCommunicationActivity extends Activity {
29
30   private byte[] data;
31   private String server;
32   private String name;
33
34   private final static String APP = NetworkCommunicationActivity.
       class
35     .getSimpleName();
36
37   public void onCreate(Bundle savedInstanceState) {
38     super.onCreate(savedInstanceState);
39     LinearLayout blank = new LinearLayout(this);
40     setContentView(blank);
41
```

```
42      Intent intent = this.getIntent();
43      server = intent.getStringExtra("SERVER");
44      data = intent.getByteArrayExtra("DATA");
45      name = intent.getStringExtra("NAME");
46      if(data == null || server == null || name == null){
47        Log.d(APP,"Data was null");
48        NetworkCommunicationActivity.this.finish();
49      }
50      try {
51        sendData();
52      } catch (IOException e) {
53        e.printStackTrace();
54      }
55    }
56
57    public void sendData()
58        throws IOException {
59      Log.d(APP, "sendData");
60      ByteArrayInputStream bin = new ByteArrayInputStream(data);
61
62      MultipartEntity entity = new MultipartEntity();
```

```
63      entity.addPart("thefile", new InputStreamBody(bin, name));
64
65      UploadTask t = new UploadTask(server);
66      t.execute(entity);
67  }
68
69  public HttpResponse sendRequest(String url, HttpEntity entity)
        throws ClientProtocolException, IOException
70          {
71      Log.d(APP, "sendRequest");
72      HttpClient client = new DefaultHttpClient();
73      HttpPost request = new HttpPost(url);
74      request.setEntity(entity);
75      HttpResponse resp = client.execute(request);
76
77      return resp;
78  }
79
80  private class UploadTask extends
81          AsyncTask<HttpEntity, Integer, HttpResponse> {
82
```

```
83      private String url_;
84
85      public UploadTask(String url) {
86        super();
87        url_ = url;
88      }
89
90      @Override
91      protected HttpResponse doInBackground(HttpEntity... params) {
92        try {
93          return sendRequest(url_, params[0]);
94        } catch (Exception e) {
95          e.printStackTrace();
96          return null;
97        }
98      }
99
100     protected void onPostExecute(HttpResponse resp) {
101       if (resp != null) {
102         InputStream in;
103
```

```
104         try {
105           in = resp.getEntity().getContent();
106
107           Writer writer = new StringWriter();
108           Reader reader;
109
110           reader = new BufferedReader(new InputStreamReader(in,
111               "UTF-8"));
112           char[] cdata = new char[1024];
113           int bytesread = 0;
114           String r = "";
115           while ((bytesread = reader.read(cdata)) != -1) {
116             Log.d("OMEGA",
117                 "Bytes read: " + String.valueOf(bytesread));
118             writer.write(cdata, 0, bytesread);
119           }
120           r = writer.toString();
121
122           Intent i = new Intent(NetworkCommunicationActivity.this,
                PolygonDisplayActivity.class);
123           i.putExtra("DATA", data);
```

```
124             i.putExtra("POLYGON", r);
125             NetworkCommunicationActivity.this.startActivity(i);
126           } catch (IllegalStateException e) {
127             e.printStackTrace();
128           } catch (IOException e) {
129             e.printStackTrace();
130           }
131         }
132
133       }
134
135   }
136
137 }
```

B.5 PolygonDisplayActivity.java

```
1 package org.miranda.hd4ar.client;
2
3 import java.util.ArrayList;
4
```

```
 5 import android.app.Activity;
 6 import android.content.Intent;
 7 import android.graphics.Bitmap;
 8 import android.graphics.BitmapFactory;
 9 import android.os.Bundle;
10 import android.os.Handler;
11 import android.util.Log;
12 import android.view.Menu;
13 import android.view.MenuInflater;
14 import android.widget.ImageView;
15
16 public class PolygonDisplayActivity extends Activity {
17
18   private final static String APP = PolygonDisplayActivity.class.
       getSimpleName();
19
20   private ImageView pictureView_;
21   private PolygonSurfaceView surfaceView_;
22
23   private Handler handler_ = new Handler();
24
```

```
25  //Bitmap bmp;
26  int xcenter,ycenter;
27
28  public void onCreate(Bundle savedInstanceState) {
29    super.onCreate(savedInstanceState);
30    setContentView(R.layout.surfaceviewlayout);
31
32    pictureView_ = (ImageView) findViewById(R.id.otherImage);
33    surfaceView_ = (PolygonSurfaceView) findViewById(R.id.
          mysurfaceview);
34
35    Intent intent = getIntent();
36
37    byte[] data;
38    data = intent.getByteArrayExtra("DATA");
39
40
41    Log.d(APP,"Creating Bitmap");
42    Bitmap bmp = BitmapFactory.decodeByteArray(data, 0, data.
          length);
43    ycenter = bmp.getHeight() / 2;
```

```
44      xcenter = bmp.getWidth() / 2;
45
46      handler_.post(new showPicture(bmp));
47
48      String polygons = intent.getStringExtra("POLYGON");
49
50      Log.d(APP,"returnWork");
51      String[] lineSplit = polygons.split("\n");
52      int count = Integer.valueOf(lineSplit[0]);
53      Log.d(APP,"count: " + String.valueOf(count));
54      for(int i = 0; i < count; i++){
55        String typeString = lineSplit[i*5 + 1];
56        String pointsString = lineSplit[i*5 + 2];
57        String colorString = lineSplit[i*5+ 3];
58        String indexString = lineSplit[i*5 + 4];
59        String dataString = lineSplit[i*5 + 5];
60        Polygon p = createPolygonFromValues(typeString,pointsString,
              colorString,indexString,dataString,xcenter,ycenter);
61        if(p != null){
62          Log.d(APP,"Adding Polygon");
63          surfaceView_.addPolygon(p);
```

```
64        }else{
65          Log.d(APP,"Polygon is NULL!");
66        }
67      }
68      surfaceView_.requestRender();
69    }
70
71    private int[] intListToArray(ArrayList<Integer> aL){
72      int[] theInts = new int[aL.size()];
73      for(int i = 0; i < aL.size(); i++){
74        theInts[i] = aL.get(i);
75      }
76      return theInts;
77    }
78
79    private byte[] byteListToArray(ArrayList<Byte> aL){
80      byte[] theBytes = new byte[aL.size()];
81      for(int i = 0; i < aL.size(); i++){
82        theBytes[i] = aL.get(i);
83      }
84      return theBytes;
```

```
85  }
86
87  private Polygon createPolygonFromValues(String t, String v,
        String c, String i, String data, double halfx, double halfy){
88    Log.d(APP,"createPolygonFromValues");
89    int type = Integer.valueOf(t);
90    switch (type){
91    case Polygon.TYPE_BOX_FLAT: {
92      String[] vertSplit = v.split(" ");
93      Log.d(APP,"VERTICES: " + v);
94      int numPts = Integer.valueOf(vertSplit[0]);
95      ArrayList<Integer> vertices = new ArrayList<Integer>();
96      for(int j = 0; j < numPts; j++){
97        double x = Double.valueOf(vertSplit[j*2+1]);
98        Log.d("X", String.valueOf(x));
99        double y = Double.valueOf(vertSplit[j*2+2]);
100       Log.d("Y", String.valueOf(y));
101       Log.d(APP,"HALVES: " + String.valueOf(halfx) + ":" +
              String.valueOf(halfy));
102       int xPos = (int)((x / halfx) * Polygon.VALUE_ONE);
103       Log.d(APP,"XRatio: " + String.valueOf(x/halfx));
```

```
104         Log.d("xPos",String.valueOf(xPos));
105         int yPos = (int)((y / halfy) * Polygon.VALUE_ONE);
106         Log.d(APP,"YRatio: " + String.valueOf(y/halfy));
107         Log.d("yPos", String.valueOf(yPos));
108         vertices.add(xPos);
109         vertices.add(yPos);
110
111       }
112
113       ArrayList<Integer> colors = new ArrayList<Integer>();
114       Log.d(APP,"COLOR: " + c);
115       String[] colorSplit = c.split(" ");
116       for(int j = 0; j < numPts; j++){
117         for(int k = 0; k < colorSplit.length; k++){
118
119           if(colorSplit[k].equals("0x10000")){
120             colors.add(Polygon.VALUE_ONE);
121           }else{
122             colors.add(Integer.valueOf(colorSplit[k]));
123           }
124         }
```

```
125         }
126         ArrayList<Byte> indices = new ArrayList<Byte>();
127         Log.d(APP,"INDICES: " + i);
128         String[] indSplit = i.split(" ");
129         for(int j = 0; j < indSplit.length; j++){
130           Log.d(APP,"Index: " + indSplit[j]);
131           indices.add(Byte.valueOf(indSplit[j]));
132         }
133         Polygon p = new Polygon(intListToArray(vertices), 2,
              intListToArray(colors), 4, byteListToArray(indices), type
              , data);
134         return p;
135       }
136       default:
137         break;
138       }
139       return null;
140   }
141
142   private class showPicture implements Runnable{
143
```

```
144     private Bitmap bmp_ = null;
145
146     public showPicture(Bitmap bmp){
147       bmp_ = bmp;
148     }
149
150     public void run() {
151       if(pictureView_ == null){
152         Log.e(APP,"View is NULL");
153       }
154       if(bmp_ != null) {
155         pictureView_.setImageBitmap(bmp_);
156       } else {
157         Log.e(APP,"NULL BITMAP");
158       }
159     }
160
161   }
162
163   @Override
164   public boolean onCreateOptionsMenu(Menu menu) {
```

```
165         MenuInflater inflater = getMenuInflater();
166         inflater.inflate(R.menu.polygonmenu, menu);
167         return true;
168     }
169 }
```

B.6 surfaceviewlayout.xml

```
1<?xml version="1.0" encoding="utf-8"?>
2<FrameLayout
3    xmlns:android="http://schemas.android.com/apk/res/android"
4    android:layout_width="match_parent"
5    android:layout_height="match_parent">
6    <ImageView android:src="@drawable/icon" android:layout_width="
         fill_parent" android:layout_height="fill_parent" android:id="
         @+id/otherImage" android:scaleType="fitXY"></ImageView>
7    <org.miranda.hd4ar.client.PolygonSurfaceView
8        android:id="@+id/mysurfaceview"
9        android:orientation="horizontal"
10       android:layout_width="fill_parent"
11       android:layout_height="fill_parent">
```

```
12   </org.miranda.hd4ar.client.PolygonSurfaceView>
13</FrameLayout>
```

B.7 PolygonSurfaceView.java

```
 1 package org.miranda.hd4ar.client;
 2
 3 import android.content.Context;
 4 import android.graphics.PixelFormat;
 5 import android.opengl.GLSurfaceView;
 6 import android.util.AttributeSet;
 7 import android.util.Log;
 8 import android.view.MotionEvent;
 9 import android.widget.Toast;
10
11 public class PolygonSurfaceView extends GLSurfaceView {
12
13   private static final String LOG_TAG = PolygonSurfaceView.class.
         getSimpleName();
14
15   private HD4ARRenderer mRenderer;
```

```
16
17  private Polygon mSelectedPolygon = null;
18
19  public PolygonSurfaceView(Context context){
20    super(context);
21    init();
22  }
23
24  public PolygonSurfaceView(Context context, AttributeSet as) {
25    super(context, as);
26    int count = as.getAttributeCount();
27    for(int i = 0; i < count; i++){
28      Log.d("BROKEN",as.getAttributeValue(i));
29    }
30
31    init();
32  }
33
34  private void init(){
35    setEGLConfigChooser(8, 8, 8, 8, 16, 0);
36    getHolder().setFormat(PixelFormat.TRANSPARENT);
```

```
37      mRenderer = new HD4ARRenderer();
38      setRenderer(mRenderer);
39      setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
40
41      this.setZOrderOnTop(true);
42    }
43
44    public void addPolygon(Polygon p) {
45      mRenderer.addPolygon(p);
46    }
47
48    public void addPolygon(String type, String points, String color,
49        String indices) {
50      // mRenderer.addPolygon(p);
51    }
52
53    @Override public boolean onTouchEvent(MotionEvent e) {
54      float x = e.getX();
55          float y = e.getY();
56          //Log.d(LOG_TAG,"Screen touch at: " + x + " " + y);
57          switch (e.getAction()) {
```

```
58          case MotionEvent.ACTION_UP:
59            Log.d(LOG_TAG,"Screen touch at: " + x + " " + y);
60            if(mSelectedPolygon != null){
61              mSelectedPolygon.setLineWidth(Polygon.LINE_WIDTH);
62            }
63            Polygon p = mRenderer.findNextPolygon();
64              //Polygon p = mRenderer.findNearestPolygon(x,y);
65              if(p != null){
66                p.setLineWidth(Polygon.SELECTED_LINE_WIDTH);
67              }
68              requestRender();
69              Toast.makeText(this.getContext(), p.getText(), Toast.
                    LENGTH_LONG).show();
70            break;
71          default:
72              break;
73          }
74
75          return true;
76      }
77 }
```

B.8 Polygon.java

```
1 package org.miranda.hd4ar.client;
2 import java.nio.ByteBuffer;
3 import java.nio.ByteOrder;
4 import java.nio.IntBuffer;
5
6 import javax.microedition.khronos.opengles.GL10;
7
8 import android.util.Log;
9
10 public class Polygon {
11
12   public final static int VALUE_ONE = 0x10000;
13
14   public final static int TYPE_BOX_FLAT = 0;
15   public final static int TYPE_BOX_3D = 1;
16   public final static int TYPE_SPHERE = 2;
17
18   private final static String LOG_TAG = Polygon.class.
       getSimpleName();
19
```

```
20  public final static  int LINE_WIDTH = 2;
21  public final static int SELECTED_LINE_WIDTH = 4;
22
23  private int lineWidth = LINE_WIDTH;
24
25  public Polygon(int[] vertices, int vS, int[] colors, int cS,
       byte[] indices, int t, String d){
26    String verts ="";
27    for(int i = 0; i < vertices.length;i++){
28      verts += (String.valueOf(vertices[i]) + " ");
29    }
30    Log.d(LOG_TAG,"VERTS: " + verts);
31
32    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
33        vbb.order(ByteOrder.nativeOrder());
34        vertexBuffer = vbb.asIntBuffer();
35        vertexBuffer.put(vertices);
36        vertexBuffer.position(0);
37
38        String cols ="";
39    for(int i = 0; i < colors.length;i++){
```

```
40        cols += (String.valueOf(colors[i]) + " ");
41      }
42      Log.d(LOG_TAG,"COLS: " + cols);
43
44
45          ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length
                *4);
46          cbb.order(ByteOrder.nativeOrder());
47          colorBuffer = cbb.asIntBuffer();
48          colorBuffer.put(colors);
49          colorBuffer.position(0);
50
51          String ind ="";
52      for(int i = 0; i < indices.length;i++){
53        ind += (String.valueOf(indices[i]) + " ");
54      }
55      Log.d(LOG_TAG,"IND: " + ind);
56
57
58          indexBuffer = ByteBuffer.allocateDirect(indices.length);
59          indexBuffer.put(indices);
```

```
60          indexBuffer.position(0);
61
62          data = d;
63          createText(data);
64      }
65
66      public void createText(String text){
67
68      }
69
70      public String getText(){
71        return data;
72      }
73
74      public void draw(GL10 gl){
75        gl.glLineWidth(lineWidth);
76        gl.glFrontFace(GL10.GL_CW);
77        gl.glVertexPointer(2, GL10.GL_FIXED, 0, vertexBuffer);
78
79        gl.glColorPointer(4, GL10.GL_FIXED, 0, colorBuffer);
```

```
80      gl.glDrawElements(GL10.GL_LINES, indexBuffer.capacity(), GL10.
            GL_UNSIGNED_BYTE, indexBuffer);
81      /*gl.glEnable(GL10.GL_BLEND);
82      gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA
            +3);
83      gl.glDrawElements(GL10.GL_TRIANGLES, indexBuffer.capacity(),
            GL10.GL_UNSIGNED_BYTE, indexBuffer);
84      gl.glDisable(GL10.GL_BLEND);*/
85  }
86
87  public void setLineWidth(int lw){
88    lineWidth = lw;
89  }
90
91  public boolean selected(){
92    return false;
93  }
94
95  //private DoubleBuffer vertexBuffer;
96  private IntBuffer vertexBuffer;
97  private IntBuffer colorBuffer;
```

```
98    private ByteBuffer indexBuffer;
99    private String data;
100
101 }
```

B.9 HD4ARRenderer.java

```
1 package org.miranda.hd4ar.client;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5
6 import javax.microedition.khronos.egl.EGLConfig;
7 import javax.microedition.khronos.opengles.GL10;
8
9 import android.opengl.GLSurfaceView;
10 import android.util.Log;
11
12 class HD4ARRenderer implements GLSurfaceView.Renderer {
13
```

```
14  private final static String APP = HD4ARClient.class.
        getSimpleName();
15
16  private int mSelectedIndex = -1;
17
18  public HD4ARRenderer() {
19    polygons = new ArrayList<Polygon>();
20    }
21
22    public void onDrawFrame(GL10 gl) {
23      Log.d(APP,"onDrawFrame");
24      Log.d(APP,"0x10000 = " + String.valueOf(0x10000));
25      Log.d(APP,"VALUE_ONE = " + String.valueOf(Polygon.VALUE_ONE)
            );
26        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.
            GL_DEPTH_BUFFER_BIT);
27
28        gl.glMatrixMode(GL10.GL_MODELVIEW);
29        gl.glLoadIdentity();
30
31        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
```

```
32          gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
33
34          Iterator<Polygon> it = polygons.iterator();
35          Log.d(APP, String.valueOf(polygons.size()));
36          while(it.hasNext()){
37            Log.d(APP, "Drawing polygon");
38            it.next().draw(gl);
39          }
40      }
41
42      public void onSurfaceChanged(GL10 gl, int width, int height) {
43           gl.glViewport(0, 0, width, height);
44
45           Log.i(APP,"w/h: " + String.valueOf(width) + "/" + String.
                 valueOf(height));
46           gl.glMatrixMode(GL10.GL_PROJECTION);
47           gl.glLoadIdentity();
48           gl.glOrthof(-width, width, -height, height, 0, 0);
49
50      }
51
```

```
52      public void onSurfaceCreated(GL10 gl, EGLConfig config) {
53          gl.glDisable(GL10.GL_DITHER);
54
55          gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
56                  GL10.GL_FASTEST);
57
58          gl.glClearColor(0,0,0,0);
59          gl.glEnable(GL10.GL_CULL_FACE);
60          gl.glShadeModel(GL10.GL_SMOOTH);
61          gl.glEnable(GL10.GL_DEPTH_TEST);
62      }
63
64      public void addPolygon(Polygon p){
65        polygons.add(p);
66      }
67
68      public Polygon findNextPolygon(){
69        if(polygons.size() != 0){
70          mSelectedIndex = (mSelectedIndex + 1) % polygons.size();
71          return polygons.get(mSelectedIndex);
72        }
```

```
73        return null;
74      }
75
76      public Polygon findNearestPolygon(float x, float y){
77        if(polygons.size()!= 0){
78          return polygons.get(0);
79        }
80        return null;
81      }
82
83      private ArrayList<Polygon> polygons;
84
85 }
```