Cityscape: An Augmented Reality Application for Displaying 3D Maps on a Mobile Device

Kameron W. Kincade
Colorado School of Mines
kkincade@mines.edu

I. Introduction

Augmented reality (AR) is becoming a primary area of focus in the field of computer vision. The word *augment* originates from the Latin *augmentum*, meaning "to increase" or "to make greater or more intense". Within computer vision, augmented reality generally utilizes a camera and a display to augment the user's perception of reality. This is achieved by overlaying digital artifacts onto the frames processed by the camera, giving these artifacts the impression that they are a tangible part of the physical world. By modifying the user's perception of the real world, augmented reality intends to enhance the user's experience and offer certain abilities that are not necessarily achievable in the physical world.

Augmented reality poses a wide variety of applications, many of which are seen on a daily basis. The primary benefit of augmented reality lies in its ability to render three-dimensional objects that do not exist or are difficult to create in the real world. National broadcasting companies have been using a form of augmented reality since 1998 to display the yellow first down marker on National Football League games (*Figure 1*). The yellow digital artifact indicates the yardage marker the offense much reach to obtain a new set of downs. Rather than creating a physical yellow marker on the football field, the computer generated marker saves resources, as well as eliminates the need to manually move the physical marker. One of Google's newest products, Google Glass, uses augmented reality for a variety of applications including navigation and sports (*Figure 1*). Beyond these specific examples, augmented reality has other important applications within fields such as the military, gaming industries, the medical field, and educational institutes.

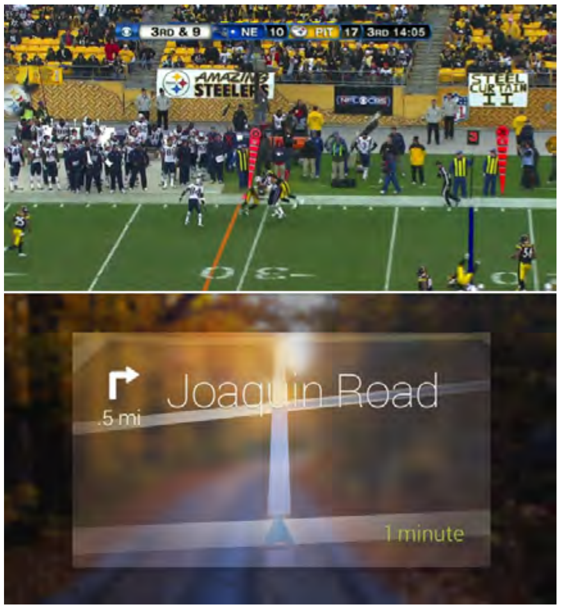

*Figure 1: The top image shows both the blue line of scrimmage marker and the yellow first down marker in a broadcasted NFL game [**?**]. The bottom image shows Google Glass' use of maps in an augmented reality setting [**?**].*

The focus of this paper is to explain *Cityscape*, an augmented reality application for displaying three-dimensional (3D) city maps on Android mobile devices (*Figure 2*). Cityscape was developed as a proof-of-concept to reveal the potential uses for 3D city maps on a mobile device, specifically using Virtual Reality Modeling Language (VRML, *.wrl files) files. The Cityscape application utilizes Vuforia, a software development kit for iOS, Android, and Unity, specifically designed for augmented reality applications.

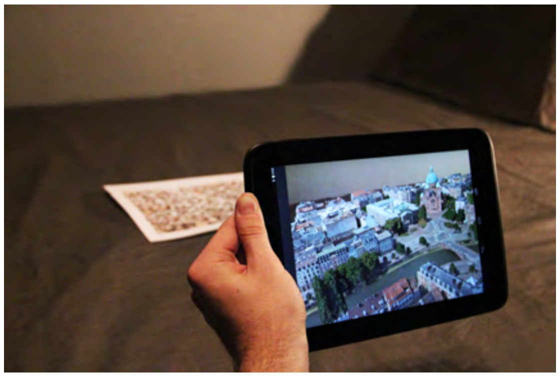

Figure 2: An image showing the end result of the Cityscape proof-of-concept application. The 3D city model is displayed on the screen of the mobile device while using the image target to track the pose of the camera in relation to the rest of the room.

II. Background

The concept of augmented reality has been around since before the birth of computers and their displays, however, only with modern technology and computing power has AR become a reality. The underlying concepts used in augmented reality are still relatively new to the field of computer vision. The most challenging aspect of augmented reality is determining where the camera is relative to its environment. The camera's position reveals the position of other objects in its field of view so graphics can be rendered in realistic positions on screen. The most widely used method to determine the camera's position is known as feature detection, which involves scanning the input images for known features.

David Lowe published the fundamental algorithm for feature detection, known as SIFT (Scale Invariant Feature Transform), in 1999 [?]. SIFT uses a set of reference images and extracts keypoints that are then stored within a database. SIFT recognizes objects in a new image by comparing the image's individual features to this database and finding matches based on the Euclidean distance of their feature vectors. The discovered matches are narrowed down to subsets of keypoints that coincide with the object's location, scale, and orientation within the scene. Using the keypoints that agree on the object's position in space eliminates a large amount of false detections and filters the correct matches from the initial set. The detection of these keypoint clusters is performed quickly due to its efficient hash table data structure implementation. This is particularly important due to the large number of features found within a single image. Lowe explains, "a typical image of size 500x500 pixels will give rise to about 2000 stable features (although this number depends on both image content and choices for various parameters)" [?].

In addition to SIFT, other algorithms have been developed to efficiently detect feature points within images. SURF (Speeded Up Robust Features) was introduced in 2006 by Herbert Bay and claims to be several times faster than its SIFT competitor [?]. SURF utilizes integral images and a Hessian based blob detector to find its points of interest, which allows for increased computational efficiency. Another feature detector algorithm, FAST (Features from Accelerated Segment Tests), uses corner detection principles to determine keypoints within an image. For a given pixel, FAST analyzes the 16 surrounding pixels' intensities and determines if a certain number (e.g. n = 12) of them fall above or below a certain threshold [?]. If the criterion is met, a corner has likely been detected at that particular pixel value. Yet another feature detection algorithm, known as ORB (Oriented FAST and Rotated BRIEF), builds on previously used algorithms to extract feature points within images. It combines the SIFT and SURF approach of calculating the object's orientation with the FAST algorithm to detect keypoints within the image. It also expands on the BRIEF algorithm that "uses simple binary tests between pixels in a smoothed image patch" to extract features from the input image [?].

II.1 Vuforia and Extended Tracking

Vuforia is a software development kit primarily developed for mobile device applications. At its core, Vuforia uses OpenGL to render graphics on top of the frames received from the mobile device's camera, while providing a number of abstractions to make the process both flexible and easy to use. It utilizes its own feature detection method, comprised of sophisticated algorithms to detect and track features within the camera's frames. The majority of Vuforia applications use an image target (*Figure 3*). The image target is recognized by comparing the camera's detected features against those from the image target database. The target will be tracked as long as the image stays within the camera's field of view. However, once the image target leaves the camera's field of view, the camera's position relative to its environment is lost. This reveals the need for extended tracking.

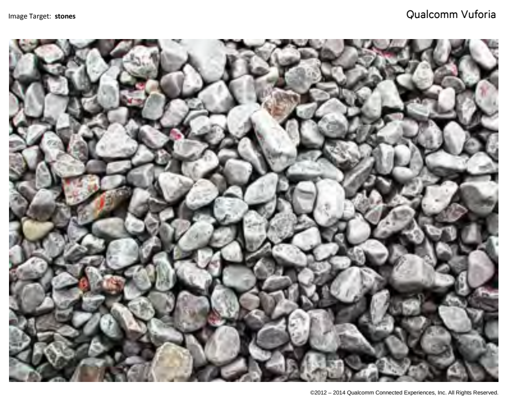
Image Target: stones
Qualcomm Vuforia
©2012 – 2014 Qualcomm Connected Experiences, Inc. All Rights Reserved.

Figure 3: This is a sample image target for a Vuforia application. The image contains subject matter with high contrast that is easily recognizable in its surrounding environment.

Extended tracking is the concept of having the program gradually learn its environment. It requires the AR application to continually learn its surroundings by detecting other features in the environment beyond that of the image target. By learning its environment, the AR application can continue to track the camera's position even after the image target has left the camera's field of view. This is a powerful concept that allows a fluid and realistic user experience. In order for extended tracking to be possible, features that are extracted from the environment must be added to the image target feature database. The camera's frames are then compared against the new database to determine if the camera's relative position can be determined.

III. Cityscape Application

The Cityscape application closely models the *Image Targets* Android sample provided by Vuforia. It uses an image target to detect and track the camera's position relative to the environment, uses extended tracking to continually track the camera's motion, and renders a three-dimensional model of a city on screen. The three-dimensional city maps are a set of WRL files provided by Microsoft, each with corresponding texture images. The set of provided WRL files are broken down into a number of levels that model a specific region of the world. Specifically, the provided WRL files model a portion of the city of Strasbourg, France. The filename of each WRL file corresponds to its location and, as a result, granularity or resolution. *Figure 4* helps explain the convention for a given WRL filename.

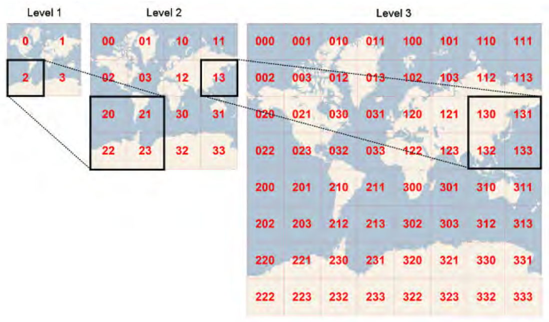

Figure 4: Each number in the WRL filename represents a quadrant of increasing resolution. The more numbers specified, the higher the resolution and the more specific the location is.

Each number, read from left to right, is between zero and three and specifies one of four quadrants, with top left corresponding to zero. Each quadrant can then be subdivided into four more quadrants each specified by another number between zero and three. Therefore, the amount of numbers in a WRL filename corresponds to the granularity or resolution of the 3D model.

III.1 Preprocessing

Outside of the actual application code, Python code was developed to convert each WRL files into the text file format parsed by the Android application. WRL files are represented in their own language, known as VRML. Rather than having the Android application parse each WRL file, some preprocessing code was developed in order to convert WRL files into a simple text format which can be read by the Android application. Although both file formats represent 3D objects, they represent them in different ways. A WRL file first contains a list of vertices with each line containing an x, y, and z coordinate. The vertices are followed by a list of triangle indices which specify the order for the vertices to be listed in. Each line of the triangle indices contains three integers which correspond to the indices for which vertices compose that triangle. Following the triangle indices are a list of texture coordinates, each with an x and y coordinate, which map the provided texture image onto the 3D model. The normals for each triangle are not specified but are rather calculated when the model is rendered.

Because Cityscape is only a proof-of-concept application, it was quicker to reformat the model file rather than change the code that parses the model file. The Android application needs a file containing a list of all vertices, where each x, y, and z coordinate is on its own separate line. Following the vertices are a list of each vertex's normal vector, again with each x, y, and z coordinate on its own separate line. The normals are then followed by the texture coordinates, with each x and y coordinate on its own line.

The Python files WRLtoVuforia.py and WRLFile.py are used to convert a single WRL file or a directory of WRL files into the text file format needed by Cityscape. The WRLFile class is an object-oriented representation of a WRL file. It has a *Parse* method to read in its vertices, triangle indices, and texture coordinates, a method to calculate the normals for each vertex, and a method to write the file to the appropriate Vuforia format. The WRLtoVuforia.py file takes as arguments an input file or directory, an output directory name, and an optional true/false value that specifics if the script should center the model around (0, 0), which is defaulted to true. The script determines if the input argument is a file or a directory and then processes all WRL files. If a directory is specified and the centering flag is set to true, the script uses all files' vertices to center the model.

III.2 Android Application

Upon startup of the application, all Vuforia text files contained within Android's "assets/ModelFiles" folder are loaded into memory. The vertices, normals, and texture coordinates are all stored in separate Android ByteBuffers which are later used when rendering the three-dimensional models on the device screen. After loading all Vuforia text files, the corresponding texture images are loaded into an array. It is important to note that the names of each text file and their corresponding image texture should match exactly in order for the textures to be correctly mapped to the correct portion of the map.

The Android application contains a menu that allows for different application settings to be configured by the user (*Figure 5*). The menu can be revealed by swiping from left to right on the screen or by double-tapping the screen. The *Settings* section offers options to turn on extended tracking, activate autofocus, initiate the camera's flash, and change the scale used for displaying the city maps. The *Camera* section provides an option to switch between the front-facing camera and the rear-facing camera. Lastly, the *Datasets* section specifies which image target the application will use.

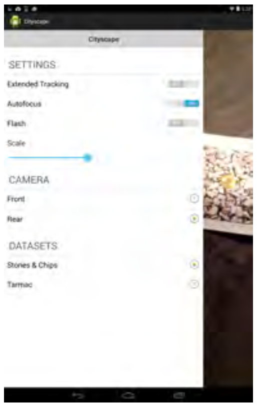

Figure 5: The Cityscape application menu.

The two image targets are stored in application's "assets" folder as an XML and corresponding DAT file. These files are loaded into Vuforia *DataSet* objects and are continually compared to the input frames provided by the device camera. The *Extended Tracking* option toggles the use of Vuforia's sophisticated extended tracking algorithms and determines whether or not the city maps are displayed. If extended tracking is off, a teapot is used in place of the city maps. If extended tracking is enabled, the DataSet objects are modified with new features pulled from new distinguishable objects within the camera's field of view. After the DataSet objects are discovered within the camera frames, the *ImageTargets.java* and *ImageTargetRenderer.java* files help render the camera frame along with the over-layed digital objects.

IV. Results

Figure 6 and *Figure 7* show the results of non-extended tracking and extended tracking, respectively. The *Scale* slider in the application menu enables the user to change the size of the city model when it is displayed on screen. The larger the scale, the larger the model will appear. A larger scale highlights the intricacies of the buildings but reveals some of the resolution issues that arise in trying to model a complex, three-dimensional city. *Figure 8* shows a larger scale value and a close up of the Eglise Catholique Saint Pierre Le Jeune building in Strasbourg, France.

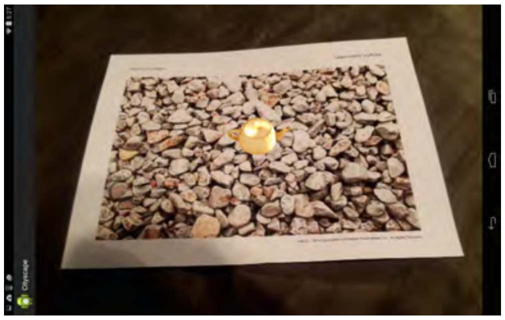

Figure 6: The teapot that is displayed when extended tracking is disabled.

```
int numTrackables = mCurrentDataset.getNumTrackables();
for (int count = 0; count < numTrackables; count++) {
    Trackable trackable = mCurrentDataset.getTrackable(count);
    if (isExtendedTrackingActive()) {
        trackable.startExtendedTracking();
    }

    String name = "Current Dataset : " + trackable.getName();
    trackable.setUserData(name);
}
```

Figure 7: An overview of the city maps that are displayed when extended tracking is active.

In order to obtain a better resolution on the city maps shown in the images in this paper, a number of smaller, higher resolution WRL files were stitched together. Specifically, the city model shown in this paper was comprised of sixteen different WRL files stitched together into a square grid. The application is designed to read all Vuforia model files stored within the "assets/ModelFiles" folder. This allows for a very large city to be displayed. However, the amount of files being read into memory directly correlates to a longer loading time when the application is started.

Overall, the Cityscape application is relatively stable. Once the image target is identified by the camera, the extended tracking reliably tracks the camera's motion and continues to render the model appropriately. Vuforia's provided extended tracking feature was tested using the Cityscape application on a number of different surfaces. Textured tables and floors of different patterns were tested with encouraging results. Even on plain, poorly textured surfaces, the application successfully tracked the camera's motion and rendered the city map appropriately.

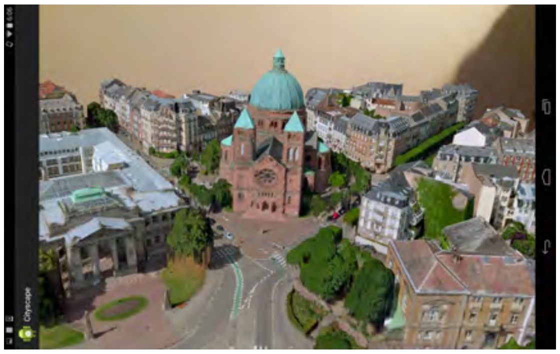
Figure 8: Close up of the Eglise Catholique Saint Pierre Le Jeune building in Strasbourg, France, as rendered by the Cityscape application.

The largest downfall of the Cityscape application is the time required to read the Vuforia model files into memory. Loading the sixteen text files needed to render the city model took roughly 25-30 seconds. In order for Cityscape to be useful, this initialization time must be decreased. Beyond the loading time, Cityscape's tracking abilities are highly dependent on Vuforia's integrated tracking methods. As a result, improving the extended tracking capabilities would be difficult and would require modifying Vuforia's source code. Due to the time constraints on the project, Cityscape did not modify any of Vuforia's source code.

V. Conclusions

Cityscape is a successful proof-of-concept application showing the power of augmented reality on a mobile device, specifically through the use of maps. The application reliably tracks the camera's motion while utilizing extended tracking capabilities to view specific portions of larger models when the image target is unable to remain in the camera's field of view. Models can be rendered with moderate resolution, while larger, higher-resolution models require more memory and a longer loading time upon application initialization. An application like Cityscape has a number of different potential applications. Architects and civil engineers could better convey specifications and designs of future projects. Military officials can better describe mission outlines and plans of attack. Smart phones can become more powerful devices, capable of a whole new idea of maps and directional aids. Teachers can better engage students in educational settings. The concept of planar augmented reality will only increase in potential as modern technology continues to evolve. Larger amounts of memory and faster processors will continue to lend themselves to augmented reality applications.

V.1 Future Work

In order to build upon Cityscape, the concept of marker-less tracking could be adapted. Instead of using a predefined image target, a marker-less tracking system creates an image target using an input frame from the camera. In such applications, it is often an option for the user to take a picture with the device to use as the image target. Cityscape could adopt the marker-less tracking strategy, allowing the user to take a photo of the environment to use for the image target, rather than needing to carry around the sample image target shown in *Figure 3*.

References

[1] Coolong, Neal. *Two 3rd-and-9 Plays, Two Years, Two Completely Different Results* `http://www.behindthesteelcurtain.com/2011/10/31/2527133/Steelers-patriots-game-analysis-week-7` October 2011

[2] Google. *Google Glass Navigation*. `https://www.google.com/glass/start/what-it-does/` December 2013

[3] Lowe, David. *Distinctive Image Features from Scale-Invariant Keypoints* University of British Columbia. Vancouver, B.C., Canada. January 5, 2004

[4] Bay, Herbert. Ess, Andreas. Tuytelaars, Tinne. Van Gool, Luc. *Speeded-Up Robust Features (SURF)*. Zurich, Switzerland. September 2008

[5] Viswanathan, Deepak Geetha. *Features from Accelerated Segment Test (FAST)*.

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/AV1FeaturefromAcceleratedSegmentTest.pdf. 16 March 2011

[6] Rublee, Ethan. Rabaud, Vincent. Konolige, Kurt. Bradski, Gary. *ORB: an efficient alternative to SIFT or SURF*. Menlo Park, California. 2008

[7] Hoff, William. *Derivation of Homograpy*. Colorado School of Mines. http://inside.mines.edu/~whoff/courses/EENG512/lectures/05a-3Dto2DTransforms-additional.pdf. 2014