AR UCSC

JONATHAN CHUANG FOR CMPS161*

CONTENTS

1 Introduction 1
2 Background 2
2.1 Cartography . . . 2
2.2 Augmented Reality . . . 3
2.3 Public Data and Open-Source . . . 4
2.4 Caveat . . . 4
3 Methodology 5
3.1 Data . . . 5
3.2 Building Conversion . . . 8
3.3 Visualization . . . 10
3.4 Conclusions . . . 13

ABSTRACT

The applications of augmented reality technology have only recently began to arise in popular acclaim. Somewhat overshadowed by its more glamorous relative, virtual reality, many of the advancements in augmented reality technology have came and gone with little publicity. As discovered in this project, rapid innovations over the last five years have made the technology extremely accessible. However, given the ease of use, the same questions that are asked of both virtual reality and augmented reality technology remains the same: what benefits can these new mediums offer the general public? Are there any notable benefits in the applications of this technology beyond being a hobbyist's affair?

1 INTRODUCTION

This paper presents the steps taken in order to produce a three-dimensional augmented reality application on mobile platforms of the general elevation, landscape, and cityscape around the University of California, Santa Cruz. Using an iOS device and tracking one of the most commonplace items on a student campus - the student ID card, the application allows the user to view the territory around the campus from all angles in three dimensions.

The application is particularly relevant to the University of California, Santa Cruz campus. Located adjacent to the Santa Cruz Mountains as part

* *Introduction to Data Visualization, University of California, Santa Cruz*

of the Pacific Coast Ranges, the campus is brimming with forestry, the region being well-known for coastal forest ecoregions. This means that often times the notable landmarks on campus are obfuscated by the local flora, making navigation often difficult for new students. By having essentially an eye in the sky, students can better grasp the layout and design of the campus itself.

Both the limitations and possible applications of similar technology are discussed in a later section of this paper.

The project relies heavily on the liberal use of node.js for it's ease of use and massive community of open-source technology and XCode, as the requisite platform for developing an iOS application. Many libraries and technologies developed by foundations, individual developers, and development teams across the world played in the part of how the app comes together. These will be discussed in the development section of this paper.

The principles behind the paper, as it turns out, are not particularly rigorous or innovative. It is an amalgamation of processing large quantities of data efficiently, productively, and in a way that comes together correctly that was core to the development. However, that is not to say that formalized area of study like those covered in the scope of this project do not exist. Many of these will be discussed in the next section of this paper: the background.

2 BACKGROUND

2.1 Cartography

The act of cartography and mapping has a long history that goes as far back into history as far as we can knowably understand it. The ability and knowledge of navigation are understandably requisite in a world that is relativistically Cartesian. Early maps were often designed around major landmarks and notable designations, often referencing visuals that evoked a sense of familiarity or uniqueness to them.

When more precise, expedient travel became necessary, more detailed and accurate maps were then needed to be created. Maps were then designed around navigation based upon the position of stars and notable celestial beings; also, many maps were made entirely by eye in the hopes of taking the visual and depth understanding of the cartographer and how they were ultimately portrayed onto paper. It was only with more advanced technology like the printing press, the compass, the clock, and others that allowed cartographers to fully pursue maps with a high degree of accuracy.

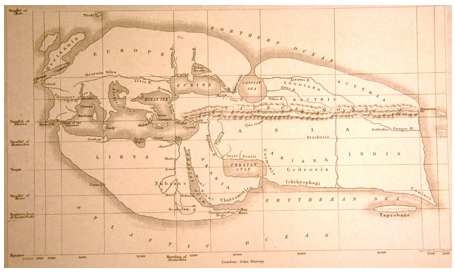


Figure 1: An 1883 recreation of Eratosthenes' map (est. 200 BCE) of the known world

However, many of these maps had a major problem. Due to the nature of printing and writing, as well as limitations in technology at the time, mathematicians and artists had to come together and come to a relative agreement over how a near spherically shaped object - in this case, the Earth - could be projected in a way that could be easily understood by the ultimate consumers of such a projection.

It is these projections that have stirred significant controversy over the last century and even beyond - often times these projections can influence and betray our understanding of the shape of the world as we know it. Certain projections can trivialize entire nations and make others the monolithic titan over its adjacent peers. However, one method of map production managed by its own natured to avoid these faults. The globe, for its inherently Earth-shaped system, projects the Earth in a way that is consistent to what the Earth actually is and is often heralded as the most technically correct map one could strive for.

Figure 2: The Erdapfel of Martin Beheim (est. 1492 CE)

It is in fact behind this that one can find the background to an augmented reality, there-dimensional variation of a section of the Earth. The act of being mapped to more than two-dimensional Cartesian coordinate system offers the same sense of reality that a globe can have without essentially being a globe. Of course, the three-dimensional model is core here, all the augmented system offers is a method of control.

2.2 Augmented Reality

Augmented reality has a surprisingly long history in the history of computer graphics, finding traces and bits and pieces going back to the middle parts of the twentieth century. However, with the indomitable widespread success of the smart phone, augmented reality technology has found a platform where many of its previous skeptics are seeing viability. I would normally avoid anecdote in this form of writing - but I happened to luck myself a pass into a major augmented reality/virtual reality conference a few years back, and the developments within augmented reality have skyrocketed.

However, what people do not realize is that many of the success stories within augmented reality have been entirely put to silence by the industry that dominated both the conference I attended and many of the platforms available - manufacturing. It is certainly no well-hidden fact that manufacturing has been keeping the majority of augmented reality technology entirely in a black box, and with good reason. Many of the implementations currently being tested of augmented reality technology have applications that highly suit the purposes of manufacturing itself - most notably employee training and reducing the potential costs of maintenance.

Many different technologies play a part in how augmented reality functions. The functional gist of augmented reality lies in computer vision. Now with speed advancements in graphics chips, analyzing and identifying key characteristics of a tracking target - in the case of Vuforia, the technology developed by Qualcomm and now owned by PTC Inc., the video stream of the device is streamed and processed in real-time, using feature detection based on the intersection of edges within the tracked image to find the relative transformation matrix representative of the tracked targets location. Only a single matrix is realistically necessary to track a single target - as from a standardized coordinate system to a transformed one is only a single set of rotations, translations, etc. Provided this matrix, the actual implementation through OpenGL is not actually particularly difficult. Any previous render outside of augmented reality can easily be mapped to augmented reality by just transforming - effectively a single matrix multiply - the system by that matrix.

2.3 Public Data and Open-Source

As the internet rapidly continues its hostile, aggressive takeover the entirety of human culture, the rapidly growing scale and size has allowed access to an almost incredible, astonishing quantity of data. Over the span of this project, hundreds of gigabytes of data were acquired, almost all of which was publicly licensed. Not only do many major scientific departments for countries both singular and plural offer many of their more notable datasets entirely for free, but individual experts in their respective fields and even businesses ofter an almost astonishing quantity of data.

Similarly, the rise of the internet has also contributed heavily to the culture of open-source code - which allow interaction and engagement with the above data without much difficulty and in contexts appropriate for each project out of context. Most of the technologies used in this project were found through npm, the packaging manager and community centered around node.js. Almost the entirety of data analysis in this project was completed via node.js. While not as performant or even consistent as other languages, node.js allows for rapid prototyping and enough flexibility for sloppy code to make the time from idea to production extremely short.

2.4 Caveat

The unfortunate issue with the scope of this project is that the background to all of the subjects necessary to download, analyze, and render the augmented reality application is large enough to cover many more sections. In particular, a significant amount of the project requires a certain degree of understanding of computer graphics and three-dimensional computer rendering. While the next few sections will focus on methodology, many more

common techniques to rendering will be discussed without much context, however, the best is done to explain or at least contextualize them as the come along.

3 METHODOLOGY

3.1 Data

3.1.1 *Dataset Preparation*

TERRAIN ELEVATION The USGS National Elevation Dataset (NED) 3D Elevation Program (3DEP) 1/9 arcsecond (approximately 3.4 meters) digital elevation model (DEM)[1] was downloaded from the USGS EarthExplorer portal. A set of four coordinates were selected by hand to represent a rectangular outline of the campus and the two data tiles that make up the outline in the terrain data were downloaded. Formatted in .IMG (the Erdas Imagine file format, a proprietary geospatial data format by now Hexagon Geospatial, previously ERDAS, Inc.), the files are read in node.js using node-gdal, a node.js binding package by Natural Atlas for the Geospatial Data Abstraction Library (GDAL) by the Open Source Geospatial Foundation (OSGeo).

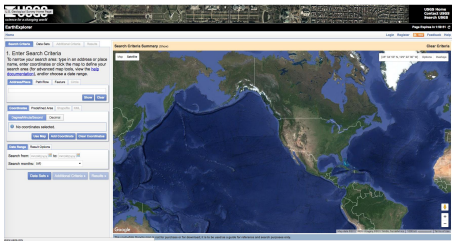


Figure 3: The USGS EarthExplorer Website

Each dataset is made of several bands, 8112x8112 2D arrays storing single values representing elevation height and other relevant information, like latitude and longitude. Using the four coordinates request, the terrain height across the desired rectangular outline was extracted from the two datasets and merged into a single 2D array. All of the values in the array (see: longitude and latitude) are converted to meters (the elevation unit) for the sake of simplicity.

SATELLITE IMAGERY There are two major sources of imagery considered - satellite imagery and aerial orthography (images taken from various airplanes for the purposes of orthographic rasterization/projection). Initially, satellite imagery was selected due to the limitations caused by skew in aerial orthography. There are two free major sources of satellite imagery - the data from Landsat 7/8 series, a satellite operation jointly ran by the USGS and NASA and downloadable through the aforementioned EarthExplorer[2], and the data from Sentinel 2, a project operated by the ESA and downloadable through EarthExplorer, the ESA Sentinel Scientific Data Hub, and as an Amazon (AWS) Public Dataset. For this project, the AWS dataset is utilized, however, they should all be roughly the same.

Due to recency and higher resolution, the data from Sentinel 2[3] was selected and downloaded. The most recent tileset with minimal cloud cover (<1%) was downloaded in the format of 12 10980x10980 JPEG2000 images,

each representing one of the twelve spatial resolution channel bands provided by Sentinel 2. Each band represents a segment of the frequency range of light - four total within the visible spectrum, eight external.

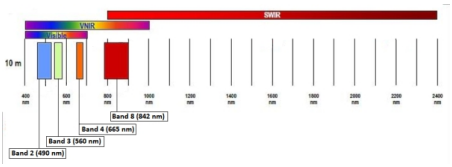
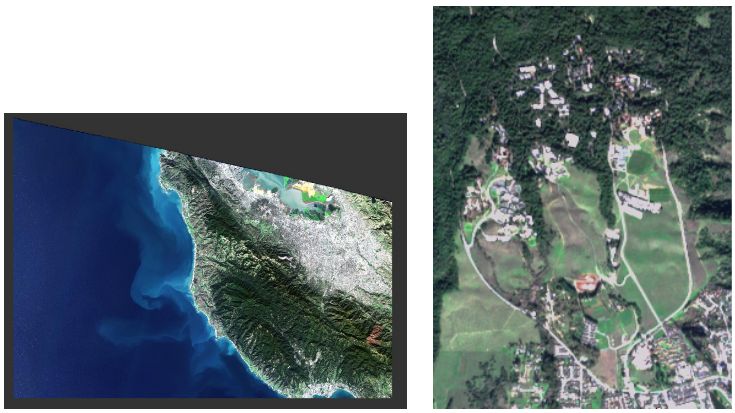


Figure 4: An image showcasing the four bands (blue, green, red, infrared) Sentinel 2 offers at 10m resolution.

In order to process these images, the red, green, and blue channels (Bands 4, 3, and 2 respectively) have to be merged and processed. This can technically be done in a tool like Photoshop, however, given the extraordinary size of the images, it can be difficult and slow to process (namely, cut out the segments relevant to the coordinates of the elevation date) and save. Instead, a tool called the Sentinel Application Platform (SNAP) was utilized to merge the three channels and extract the desired region (via latitudinal and longitudinal coordinates).

White balancing, curve adjustments, and other adjustments were made to improve the visual quality of the resulting imagery. Seeing as how human vision does not perceive the visible spectrum linearly, many of the images we commonly associate with imagery of terrain is not representative of what the output light looks like. Adjustments must be made to feel similar to how humans naturally perceive the visible spectrum[4].

The resulting image, while satisfying the fundamental criterion of the project, did not satisfy the resolution demands. The resulting image from the Sentinel 2 public dataset was far too low resolution for the purposes of augmented reality and was consequently rejected. While other satellite imagery sources (for a cost) were considered, such as the WorldView satellites covered by DigitalGlobe[5], the GeoEye satellites, satellite imagery owned by the Airbus Defense and Space Department, etc. they were far to cost prohibitive for the size of the University of California, Santa Cruz campus and were similarly rejected.



(a) A Sentinel 2 tile. **(b)** After cropping

Figure 5: Working with Sentinel 2 Data.

Figure 6: The final imagery created, using aerial orthography.

AERIAL ORTHOGRAPHY In the end, aerial orthography was selected as the primary source for imagery. Through EarthExplorer, a total of twenty four images were extracted from the regions representing the campus, stored as 12000x8000 TIF images. Due to the extreme size of all images, these tiles converted in batch to a much smaller resolution PNG files using sharps, a node.js binding set for libvips and PhotoScape was utilized to merge all twenty four images into vertical strips, then larger ones. Given the range of coordinates of the corner images of the dataset allowed extraction of the relevant region of terrain. The resulting image was then formatted to fit a power of two image (in this case, 2048x2048) for usage as a texture in OpenGL ES 2.0.

BUILDING DATA All of the buildings utilized in the project were obtained via the OpenStreetMaps (OSM) Buildings API. Their web API allows for downloading based on a range of latitudinal and longitudinal coordinates[6], so the relevant data was extracted as a singular XML file. This file would then be processed using xml2js, a node.js library purposed for XML to JSON conversion.

The dataset itself is a wealth of information provided by many OpenStreetMap contributors, including Peter McMillan, a director at UCSC IT. It is formatted in three majors segments - a list of nodes representing a latitudinal and longitudinal coordinate with information about the date of creation and creator, a list of major features with references to single nodes, highlighting wheelchair limited access points and other map-relevant details, and finally, a list of buildings stored as a series of nodes representing the outer edge of the building and other children nodes storing relevant information about the buildings (building types, like "University", "Apartments", "Greenhouse", "Library", etc., the address, and on occasion, the number of levels the building is).

3.1.2 *Dataset Conversion*

TERRAIN ELEVATION Given the elevation data, a simple mesh can be formed by sampling the dataset, forming effectively a heightmap of the relevant terrain. Each arrangement of four points can then be utilized as the four vertices (or two triangles) of a mesh, mapped across the entire dataset. Normals are calculated by taking the cross-product of the intersecting vectors

(or the average of the two vectors making up each axis - north and south averaged and west and east averaged). The resulting mesh data was then converted to the Wavefront OBJ format for easy parsing, with indices used to dramatically reduce the number of requisite vertices.

AERIAL ORTHOGRAPHY The texture coordinates of the aforementioned mesh were generated relatively simply - mapped evenly across the sampled dataset.

The resulting OBJ file from two above two subsections is 66,638 lines long.

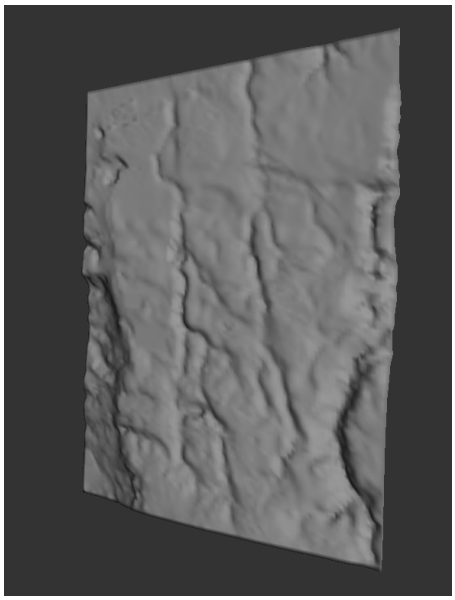


Figure 7: The terrain mesh generated and viewed with OBJ Viewer.

BUILDING DATA Buildings are a difficult problem made up of several steps, so the next section will be dedicate towards the difficulties and challenges presented.[7]

3.2 Building Conversion

BUILDING ROOFS All of the buildings in the project currently use a flat roof. However, given the complexity of the shape of roofs of the buildings on campus (the outline of a building is not necessarily convex), a more complex system was required to output the building roofs.

Mapbox's earcut library[8], which uses Fast Industrial-Strength Triangulation of Polygons (FIST) and Z-order curves, to triangulate a set of convex vertices. FIST is an optimized implementation of the more fundamental ear-clipping technique, which is dependent on the two ears theorem, that any four vertices can be represented by two triangles, and involves the procedural extraction of triangles from a surface. Z-order curves are a computationally quick technique of projecting multidimensional data to a single dimension and are utilized to optimize and guarantee the results of the ear-clipping process.

For building height, the highest point on the elevation data across all of the nodes was used as a baseline and added to the relevant number of floors (if provided by the OSM XML data) with each floor being approximately three to four meters, or approximately a little above ten feet. In the

case where the floor data is not available, the height of approximated as being directly proportional to the area of the building. The shoelace formula (also known as Gauss's area formula or the surveyor's formula), a relatively straightforward formula which takes a rolling aggregate of multiplicand vector values and a rather elegant calculus proof that works similarly to the two ears theorem, was utilized to calculate the area. This formula is also used in the next section.

BUILDING WALLS Given the coordinates of every building vertices, a wall can be generated by wrapping around the building and constructing each edge of the building as a pair of triangles. However, the dataset provided OSM can list the vertices representing the buildings in clockwise or counter-clockwise order, which results in problems in rendering due to cross-product calculations (see: right hand rule). The Shoelace formula can be modified to find the orientation of the list of vertices, as the sum of the determinants (as calculated by the multiplication of the vectors representing the edges as you traverse the list of vertices) represents the direction of the nodes be it positive or negative (representing counter-clockwise and clockwise respectively). The walls extend to the height of the elevation data and do not extend below the terrain. While this detail isn't particularly noticeable, it is noteworthy.

BUILDING NORMALS Two attempts at computing building normals (the walls, not the roof, which simply points up) were made. The first was done by simply finding the cross product of two of the edges of each wall, giving effectively flat shading. The second was done by taking the half-angle at the vertex between the two vectors that extend to its adjacent vertices. This output results in smoother shading. However, a large problem with the ultimate visualization is that the buildings have a difficulty in visibility due to blending into the terrain texture and smooth (Gouraud or Phong) shading only conflagrated these issues. Flat shading (technically, but purposelessly, a Gouraud shader was utilized) was the ultimate selection.

BUILDING INFORMATION Building information requires two outputs. The first step is to store the information of the building. A total of 531 buildings are present in the dataset for buildings on the UCSC campus, broken up into nine major categories (minor categories do also exist, but often were the product of bad data and handpicked off). The buildings were categorized into two values - the type and a unique ID. The purpose here was to limit the number of building IDs such that no ID could be above 255, or the maximum possible visual color renderable for identification.

Table 1: Table of Building Types

Building Type	Quantity
University	236
Residential	14
House	58
Dormitory	53
Greenhouse	9
Farm Auxillary	25
Apartments	105
Trailer	32
Industrial	21

The first output is taking the data from the OSM XML and outputting a JS file with all of the buildings and their relevant info organized by type. This will be the reference dataset that will be loaded in the web-based user interface.

The second output is storing the two values mentioned above (the type and unique ID). Due to not using the particular portion of the OBJ file, the type and unique ID were mapped to the texture coordinates and stored there for easy processing and binding in OpenGL.

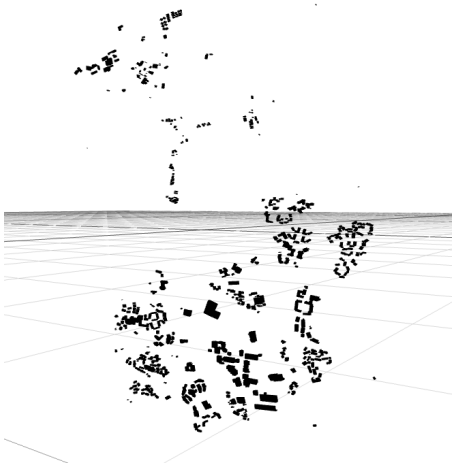


Figure 8: The building OBJ file previewed in XCode.

OUTPUT The resulting building model was output via OBJ format as well, broken up into the two major sections (roofs and walls) outlined above. All of the roofs utilize the same normal, which is stored as the first normal in the OBJ file.

The resulting OBJ file is 29,595 lines long.

3.3 Visualization

3.3.1 *Tracking*

Vuforia[9] simplifies the process of target tracking significantly, taking care of most of the computer vision (CV) steps involved. The only major work required was feeding it a strong base dataset to use to track as a target. While the student ID card has plenty of features that can be used to track, a difficulty can arise when the base dataset is to opinionated and consequently has issues tracking.

Figure 9: The edited student ID card used.

This is solved by taking an image of the student ID and masking any unique characteristics. The profile picture was swapped with the aggregate image of some 32 faces of male German students at the University of Regensburg in Regensburg, Bavaraia, Germany. The barcode and quarter identifier, as well as the student name and affiliated college, were masked out utilizing the stamp tool in Photoshop, with careful attention to not generate any more distinguishing features.

3.3.2 *Processing*

Nick Lockwood's GLView toolkit found in GitHub was utilized to process the OBJ files. Due to limitations on iOS and in XCode, arrays of sufficiently large enough size do not function fully properly. GLView circumvents these issues by spreading out the load over several arrays and with several other tricks to improve load times.

3.3.3 *Rendering*

iOS uses OpenGL ES 2.0, built on their own Embedded Apple Graphics Library (EAGL). It's implementation is very similar to what one would encounter in OpenGL ES 2.0. I created three GLSL shaders - one for the terrain with a texture, one for the buildings with Gouraud shading, and one for the secondary buffer used for building identification.

The actual act of rendering is not particularly complex, however, took a few redesigns due to limitations of Apple's implementation and the core limitations OpenGL ES 2.0. For example, multiple render buffers outside of one of every color/depth/stencil buffer is not permitted. As a result, a secondary frame buffer is required to draw to the offscreen buffer. Apple also demands substituting several significant OpenGL commands in replacement for their own implementations and insuring the completeness of buffers before entry can be difficult if not meticulously planned.

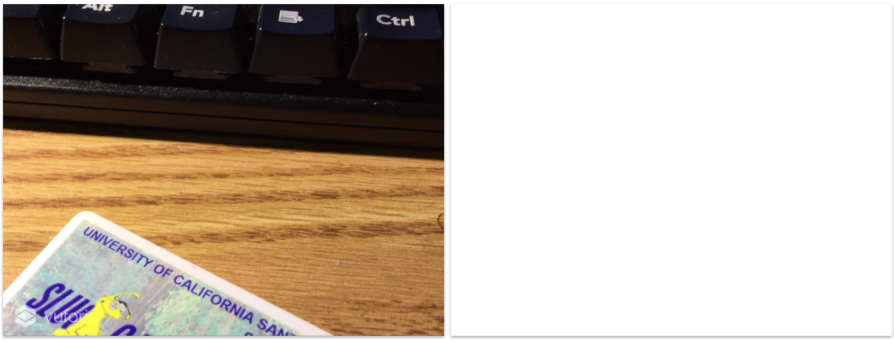
The rendering loop is straightforward. Vuforia renders the output of the camera on its own buffer and then to the screen. Then the application reenables several GL settings (eg. GL_DEPTH_TEST) calls Vuforia for the transformation matrix (given that it has been processed and a target has been found), which is then used to render the terrain and then the buildings from the loaded OBJ files, switching shaders in-between. Then the offscreen buffer is bound and rendered to utilizing the bound texture coordinates of the building data. The primary draw buffer is loaded again and rendered to at the end of the rendering loop.

3.3.4 *Interface*

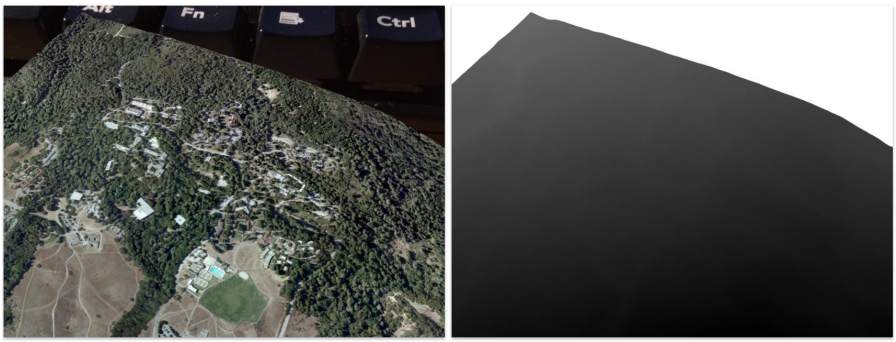
The application interface was created by adding Webkit-based view layer on top of the OpenGL layer. iOS provides the ability to communicate to the Javascript through the execution of javascript at a window level. Basic HTML/CSS/JS was utilized, notably flexbox for UI due to ease of use and the "evergreen" (always up to date) nature of iOS web implementations. The interface gives basic instructions on the use of the application as well as provides the output for building selection.

3.3.5 *Interaction*

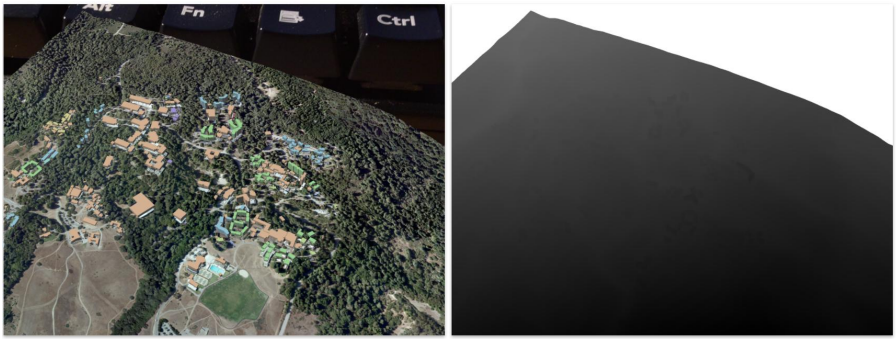
When the view controller containing both the OpenGL and web views is tapped, it sends the coordinates of the touched location to the class holding



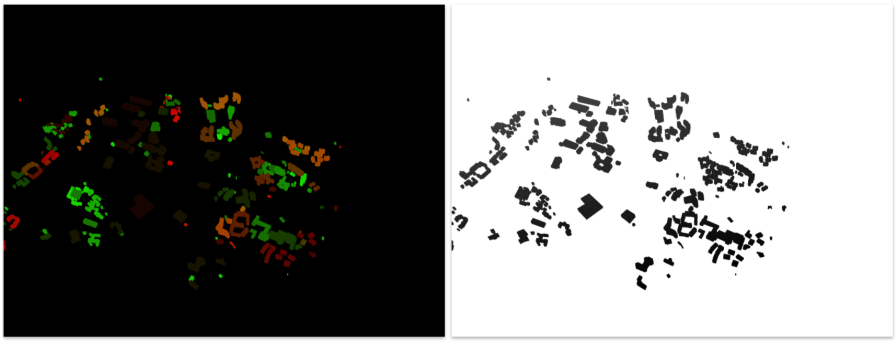
(a) Input image (color and depth).



(b) Terrain rendered.



(c) Buildings rendered.



(d) The building ID render.

Figure 10: The full rendering pass.

the OpenGL view. On each render loop, the class checks for changes to the touched position and if it detects one, it utilizes glReadPixels to extract the unsigned bytes of the offscreen buffer storing building information. This

is then sent via an executed javascript command to the web view, where a function is called which reads the aforementioned building information and updates the DOM.

3.4 Conclusions

3.4.1 *Limitations*

Projection is the largest issue given the current dataset. Almost every component of this project is affected by some sort of limitation created by projection or unit conversion. The terrain data is stored in latitude and longitude, meaning it doesn't map evenly to a rectangular shape. The imagery is taken from a plane and adjusted to a degree for orthogonality, but doing so creates bias due to projection. There are several places where the data does not align together as well - buildings are often not well mapped to the imagery. Building information is often sporadic and highly variable in quailty (quite a few buildings were originally labeled for their building type as "Yes"). While these issues are not significant and do not significantly hamper the final product, they are enough of an annoyance to be holding back the efficacy and purpose of the project.

Figure 11: The final product.

3.4.2 *Incompletions*

There are several goals that simply did not fit within the time requirements. Positioning the location of the device on the map itself using integrated GPU functionality within the devices was an obvious goal. While the latitude and longitude of the devices is easy to acquire, converting that into a position in OpenGL was a step that did not fit time requirements.

The ability to point the device in a direction and learn about buildings in that direction was suggested and very heavily considered but did not meet

time requirements. The ability to search for buildings in the interface as well as highlight them was very heavily desired. More mapping data like roads and walking paths seems like a useful feature, as well gridlines and other common map details.

Improvements to the rendering were also desired - there was a want for an improvement to the shading model used for buildings and other techniques were thought of for rendering the terrain with respect to the time of day (even using different - say, night time satellite - imagery, alternatively). Transparency and other more spirited user interface elements could have been implemented to improve the aesthetic quality of the final product.

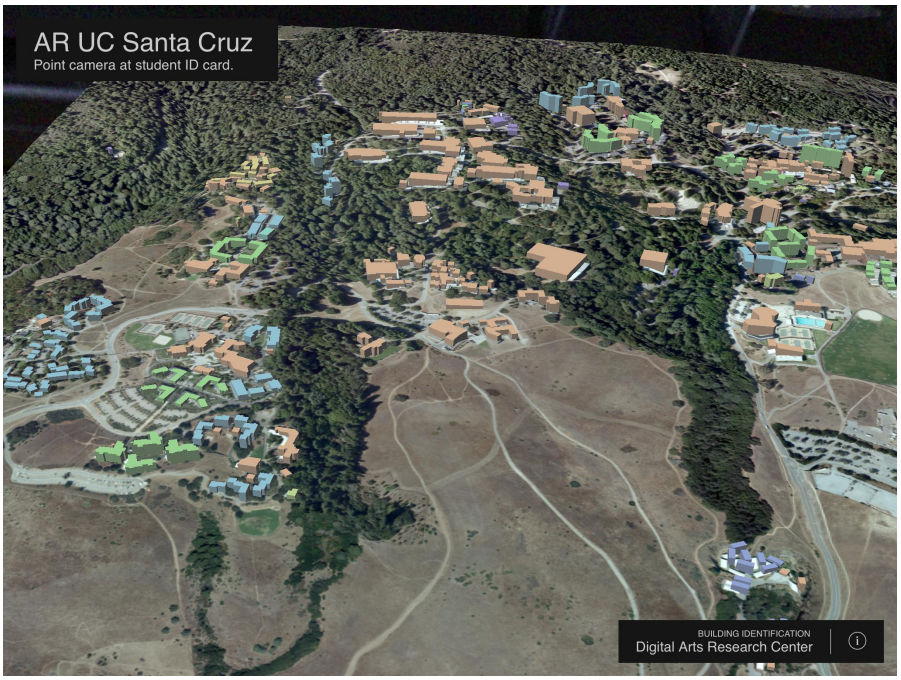


Figure 12: A close-up.

3.4.3 *Epilogue*

While the scope of the project itself covered a vast array of different datasets, processing them, and their implementations, many of the techniques utilized were not particularly difficult. Even the ones that offered a degree of difficult were circumvented through the liberal use of open-source libraries available. Tools like Vuforia bring new attention to the incredible possibilities of augmented reality and the ease of use that complements it.

What one can realize here is the astonishing limitless nature of technology and its innovations, and how often the steps required to get to something that on a particular level could be seen as never ben done before is often not particularly large or big. All that is required is the little thought and the act of taking the first step, because from then on, these things almost map themselves.

REFERENCES

[1] National Elevation Dataset (NED). https://lta.cr.usgs.gov/NED, Jan 2015.

[2] EarthExplorer. https://earthexplorer.usgs.gov/.

[3] Sentinel-2 on AWS. http://sentinel-pds.s3-website.eu-central-1.amazonaws.com/.

[4] Robert Simmon. How To Make a True-Color Landsat 8 Image. https://earthobservatory.nasa.gov/blogs/elegantfigures/2013/10/22/how-to-make-a-true-color-landsat-8-image/, Oct.

[5] DigitalGlobe. https://www.digitalglobe.com/.

[6] Buildings. http://wiki.openstreetmap.org/wiki/Buildings, Dec 2016.

[7] A J Ashton. Mapping 3d building features in OpenStreetMap. https://www.mapbox.com/blog/mapping-3d-buildings/, Oct 2016.

[8] mapbox/earcut. https://github.com/mapbox/earcut.

[9] Vuforia Developer Portal. https://developer.vuforia.com/.