Matti Anthony Lahtinen

Cross-platform solution for visualizing Artova "Designer Dog Park"

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Media Technology
Thesis
15/10/2013

Metropolia

Abstract

Author(s) Title Number of Pages Date	Matti Anthony Lahtinen Cross-platform solution for visualizing property 52 Pages 15/10/2013
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor(s)	Janne Kareinen, Artovan Henki Representative Kari Salo, Principle Lecturer

The Goal of this Bachelor Thesis was create a cross-platform interactive solution for visualizing property design with available technology. This project was a co-operative effort between Artovan Henki and Metropolia University of Applied Sciences.

The project itself was to create a three dimensional model of "Designer Koirapuisto" project (Designer Dogpark) and create an interactive solution around it. It was mainly intended to work on the latest Android and iOS platforms.

At first the intended solution for the project was to use the Augmented Reality service Layar. Recently, the service added a new feature called Layar Vision. Vision allows for objects to be rendered at varying accuracy ontop of an image using Natural Feature tracking. This required the development of a simple back end system using PHP with the CodeIgniter framework, and MySQL. The 3D models were developed using Blender.

Unfortunately, undocumented technical limitations were discovered during full scale testing, which made Layar Vision unusable for larger, property size implementations.

The project thereafter changed gears, and with the time remaining, focused on getting a Virtual Reality based solution using the Unity 3.5 Game Engine onto the Android, iOS and PC platforms. Fortunately due the 3D models being developed with real time rendering in mind, they were simple to import into Unity. Afterwards multiple layers were added onto the scene, which allowed the user to see what the park may have looked alternatively with different objects.

The prototype created in unity was then launched onto Flash, and Android 2.2+. iOS unfortunately had deployment issues due to restriction of development only on Macs.

Keywords	Mobile mixed reality, reality-virtuality continuum, 3d environment, unity game engine, Layar, web, PHP, CMS, Augmented Reality (AR), Virtual Reality (VR)

Contents

Abbreviations and Terms **3**

1 Introduction **5**

2 Virtuality Continuum **7**

2.1 The Scale of Mixed Realities 7
2.2 Virtual Reality 8
2.3 Augmented Reality 9

3 Project: Designer Dog Park **12**

4 3D Modeling Phase **14**

4.1 Brief to 3D modeling 14
4.2 Modeling the park equipment 15
4.3 Modeling the ground 18
4.4 Optimizing for Mobile Devices 20

5 Layar Development Phase **21**

5.1 Layar Architecture 21
5.2 Content Management System (CMS) 25
5.3 ARCMS Database Plan 26
5.4 ARCMS Software Architecture Plan 29
5.5 Findings of Model Conversion 31
5.6 Issues with Layar and Scaling 36

6 Unity3D Development Stage **38**

6.1 Considerations and Importing Models 38
6.2 Creating the Cube map 40
6.3 Interactions 41
6.4 Deployment 42

7 Conclusion **43**

8 References **45**

Appendices

Appendix 1. Topographical Map with Extras of Dogpark
Appendix 2. Layar Hotspot Object Map
Appendix 3. Database plan for ARCMS

Abbreviations and Terms

3D Model	3 Dimensional Model that is rendered
Android	Google's Operating System for Mobile Devices
AR	Augmented Reality
ARCMS	Augmented Reality Content Management System
Artova	Arabianranta-Toukola-Vanhakaupunki neighbourhood association
CAD	Computer Assisted Design
CI	CodeIgniter Framework
CMS	Content Management System
CodeIgniter	PHP Web Application Framework
Controller	Programming: User Input
CSS	Cascading Style Sheets
Flash	Adobe Flash multimedia-platform
GPS	Global Positioning System, Geolocation Service
Hackaton	Event where something is programmed in a limited time
HMD	Human Mounted Displays
HTML	Hypertext Markup Language
iOS	Apple's Operating System for Mobile Devices
JSON	JavaScript Object Notation
Kickstarter	Service to crowdfund projects
l3d	Layar propriety 3D Object file format
Layar	Mobile Augmented Reality Service
Layar Player	Layar Browser which uses the Layar Service

Layar Vision	Natural Feature tracking on the Layar service
Layer	Layer in Layar Software, or Photoshop Layer
Mesh	Geometry for a 3D Model
MVC	Model-View-Controller architectural pattern for applications
MySQL	Relational Database management System based on SQL
obj	Wavefront 3D Object file Format
OOP	Object Oriented Programming
PHP	Hypertext Preprocessor Programming Language
Polycount	Amount of Polygons used by a 3D model or a scene
Render	Generating an image from a model
Retopo	Remaking topology
RV continuum	Reality-Virtuality Continuum
SQL	Structured Query Language
Unity	Commercial 3D Platform used to create rapid environment prototypes and Games
URL	Uniform Resource Locator
UV Mapping	Creating a 2D representation of a 3D object
VR	Virtual Reality
Windows	Windows Phone Operating System
WoW	Window-on-the-World

1 Introduction

Virtual Reality has achieved leaps in development within the last decade and has evolved over time into a viable source of entertainment to the average consumer. In the past this format has focused on expensive computer assisted design, simulation and archeology. While this still relevant today, its expansion to entertainment has made the technology cheaper. As the technology got cheaper and smaller, [1] new applications, and variations for virtual reality have risen.

Arguably, Virtual Reality, and its subsequent variations are still fairly new topics. It is one of the hardest things to write about, as the terminology of Virtual Reality varies from a definition of a modern videogame environment to a reality simulation where the user is fully immersed in a Head mounted display and is able to interact with the environment using their body. This environment would react, and there after return nearly full sensory feedback to the user. Due to the discord in the definition, this work will use only use a single definition for it, which will be as broad as possible. Virtual Reality is "[…] a three-dimensional interactive computer-generated environment that incorporates a first-person perspective" [2; 3].

One of the new applications of Virtual Reality (VR) would to apply it over our perceived world. This is known as Augmented Reality (AR). With the advancements in the field of mobile technology; evolving phones from their roots of a single-purpose communication tool on the go [4], to the modern day smartphones allowing input and output of real-time multi-media along with the availability of a camera, access to the internet and geo-location services: mobile devices have become versatile enough to allow Augmented Reality become viable in field advertisement, industry, and entertainment, as a cognitive representation of information [5].

The goal of this project was to explore and create an interactive cross-platform solution to visualize property design with representative 3D models using available technology. It is a co-operative effort between Artovan Henki, Gammelin Koirat and the Metropolia University of Applied Sciences. This Bachelor thesis will follow the project's progress as it explores various solutions.

The main topic of thesis is to explore the methodologies behind the 3D modeling of the Park, taking budgeted modeling concepts for mobile videogames into consideration. The final model was then used in the various potential solutions for this project.

As a secondary goal, this project explores the features from Layar, along with a creation of a Content Management System which would allow for proper control of various API level requirements in the Layar Service without having to create custom code for every situation. However, due to software limitations found with Layar, the thesis documents the issues discovered and migrates to a new solution using Unity 3D; creating a simple interactive virtual environment for mobile devices.

2 Virtuality Continuum

2.1 The Scale of Mixed Realities

By a dictionary definition, *real* is something which exists, while *virtual* is something that is imaginary [6]. In computers science, this has been expanded to the definition of concepts which are simulated by a computer. For example, a simulation within a modern video game is an environment simulated by a computer and does not exist in the real world.

Reality is our perceived world, and does not have to be simulated. It can however have unexpected input and has variables which cannot always be taken to account. Augmented Reality occurs where virtual environment is augmented on top of perceived reality, [5] while the opposite happens when reality is augmented on top of the virtual environment is called Augmented Virtuality. [7]

Both of these are younger subgenres of Virtual Reality (VR), therefore, like with VR, there is about the correct definition for it. The general consensus is that Augmented Reality is the combination of the real and the virtual. Due to this, it is categorized as a Mixed Reality as shown in Figure 1; but by itself it is only one of the possible forms of it. [5; 7]

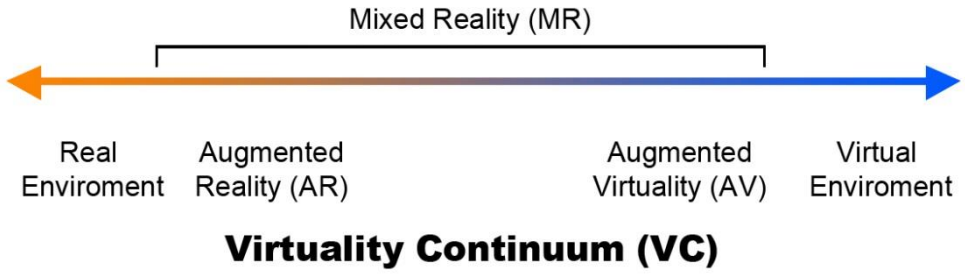


Figure 1 – Milgram's Virtuality Continuum [7]

By the definition above, Mixed Reality is a form of virtual reality merged with the real, in which on effects the other. It is a taxonomic scale (Figure 1) which defines the varying degrees on how an environment which is created by an application deviates between the Virtual Environment and the Real, and how one affects the other. [5; 7]

2.2 Virtual Reality

Originally conceptualized in late 1960s, [8] VR was also known as Cyberspace and Synthetic Reality. Its definition has varied greatly to date. Originally these terms were coined for simulations where the user would be completely immersed into a Virtual World using a Head Mounted Display (HMD). The user would interact with the world through a data suit or another interface that would allow for tactile interaction. [3; 5; 8]

After 1990's, Virtual Reality as a term began to more liberally used to non-immersive environments, or was limited to fairly expensive research and military projects. Since then, commercial development into virtual reality technology development has mainly focused on non-immersive technology such as in video games, computer assisted design and imagery. However, as of late 2012 there have been a demand and push for immersed VR; with the recent crowd funding of a HMD product called Oculus Rift. Its Kickstarter, a crowdsourcing source earned it over 2,4 million dollars, nearly 10 times its original crowd funding goal. [9; 10] This combined with already available gesture driven controllers, such as the Microsoft Kinect and Leap Motion [11] have driven new possible ways to interact with virtual objects without the use of a physical controller. This has shown how technology has become cheaper and available over the years. In the near future we might see a resurgence of immersed VR. [9; 10; 11]

As mentioned in the introduction, this work will refer to Virtual Reality as "[…] a three-dimensional interactive computer-generated environment that incorporates a first-person perspective" [2; 3]. Situations where the application might not fully implement a first-person perspective might also fall into this definition; however those will not be explored in this work.

2.3 Augmented Reality

While VR has dwindled in the past before the late 2012, its main application began to migrate into its subgenre of Augmented Reality, in which live video was combined with computer graphics [7,4]. It is most commonly used in newscasts, and sports in the form of real footage can be combined with artificial, creating a mixed medium for both real footage and virtually visualized data. While this could be done manually by the broadcaster, instead of automatically through software, it still fits the general description of Augmented Reality as it's done live as the show is broadcasting. For example, a swimming competition may show during a live what the current timer is at, on which lanes the first 3 swimmers are on, and who they are; as shown during London Olympics 2012 (Figure 2). [5,4; 7,4; 12]

Figure 2 – Screen cap of London Olympics 2012, Mens 100m Butterfly Final [13]

As mobile devices have become lighter and more powerful, some equipped with a camera and geolocation services; Augment Reality has become viable on a more portable platform. Due to the devices being portable, users now have an extra layer of discoverability in the perceived world [5]. There are signs of this also being commercialized, with Google spear heading a product, known as Google Glass. Google Glass is a portable head-worn overlay which displays superficial data that we could today retrieve with our phones [14]. Yet there are a myriad of other ways on how Augmented Reality can be implemented, as programs have to track in order to augment virtual world on top of our real world [5; 7].

One of the oldest methods for Augmented Reality has been marker based recognition. 1-Dimensional Barcodes have been the most common (Figure 3), allowing for the quick recognition by a computer to identify what a specific tag refers to, cross referencing with an existing file system or database. Usually these are system specific, as they are mere ids for a larger system. This has been extended upon by Matrix or 2 Dimensional Barcode which store more much information than the 1D Barcodes, but work in a similar method. Best known Matrix Barcode is Quick Response or QR code [5] For example, a QR Code can contain a Website URL instead of just an identification code (Figure 4).

Figure 3 (Left) – EAN-13 Barcode for number 1234567890128 [15]
Figure 4 (Middle) - QR Marker pointing http://en.m.wikipedia.org [16]
Figure 5 (Right) Example of a Fiduciary AR Marker [5]

The idea of barcodes has been extended with Fiduciary markers within AR. While these markers do not contain any data, they have identifiable geometric shapes (Figure 5) which an application can recognize through a camera. The application would be programmed to support these specific markers, allowing to individual tracking. Virtual objects or virtual environments could then be simply overlaid onto the position and orientation of these markers, superseding them. For example, a marker could be placed on an advertisement of a product in a magazine which, when visualized through a Window-on-the-World, such as a camera with a viewfinder would show a miniaturized model of a given product. It would then be possible to view a virtual product from most viewing angles. [5]

There are methods of AR which do not require a precise control over the augmentation. These methods do not require for markers, instead using other sensory methods to detection what a user might be viewing. Geolocation based AR uses the wireless networks and / or Global Positioning System's data to locate an approximate location of the user, while using the orientation of their devices through magnetic and gravimetric sensors to deduce which direction the user is looking at, and what they would see.

Figure 6 – Augment Marker and sample Sofa rendered through its Software. Notice how Scaling is dependent on the size of the marker [19]

This can be unconventional as these markers tend to be software specific. By default they do not seem fit for a print medium [17, 16-17]. Some software, such as Augment by Augment (Figure 6) and services such as Layar have attempted to solve this issue by recognizing patterns in already existing images, so that they can be converted into markers with varying success. These applications simply break down the images, to simpler components to create a 'fingerprint' which the application will look for [18; 19]. This allows for support of large posters and even art works to behave as Fiduciary markers [5]. This is known as Natural Feature Tracking, which allows the tracking of images, instead of the custom markers. This could technically be extended with gravimetric, magnetic and geolocation data to create a flowing experience [5; 17, 16-17].

This project work will explore if it is possible to use Natural Feature tracking in a practical approach to demonstrate what a designed environment may look like; and also find out if it is not successful, find a new solution on how to visualize the property. The project was done during first trimester of the year 2012. There were only few cross-platform services that could considered for the thesis. Platforms such as 'String' were considered, However, at the time of the project, the choice was only limited to iOS operating system. As there was no development kit for iOS software available on Windows and Linux machines, it was not used in the scope of this project. [20] Layar was trialed as a solution instead. Other services were considered as well, but only Layar Service had an available Natural Feature tracking through its new feature Vision. Due to the time frame, and nature of the project; a Custom AR solution was thought not to be an option.

3 Project: Designer Dog Park

The Artova Dog Park is joint pilot project between Artovan Henki and Gammelin Koirat in the World Design Capital Helsinki promotion of the year 2012. Its goal is to create a park with fencing where dogs could frolic around, and where their owners may rest without disturbing the developing area of Arabianranta or the natural park which is in the neighborhood. [21; 22; 23] The design would come from the use of functional statues. Metropolia's goal in the project was to provide a means of visualizing the property.

The original description of the project was to visualize a semi-interactive environment in the form of Augmented Reality as a Window-on-the-world which would be revealed when the user would be able to look at a marker through a mobile device. The user could then slightly change area to visualize what the area might look with and without some features. Its main topic was to explore the possibilities using Layar Vision, a new Natural Feature tracking feature for Layar which was introduced along the same time when Metropolia's involvement in the project begun. [24] I did the initial background research for it, and at the time things seemed that they might work. Figure 7 demonstrates some potential Points of Interest where the markers could have behaved as a Windows on the World (WoW).

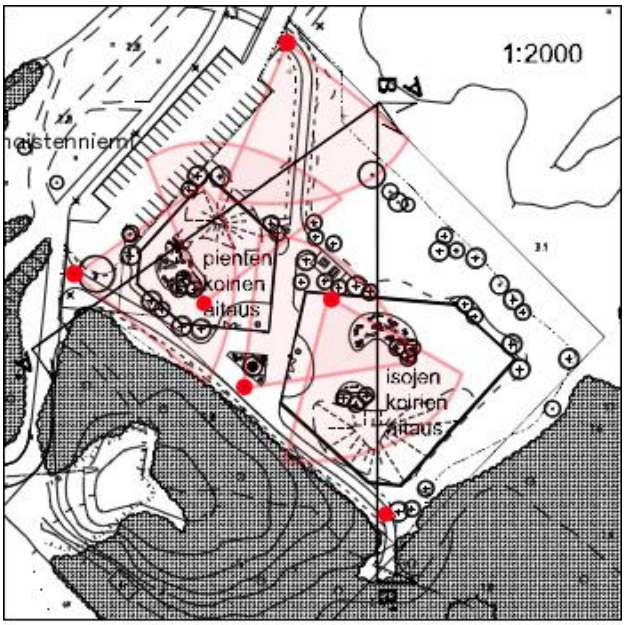


Figure 7 – Overview of the Park Location and Example of a potential Layout of WoW marker points with Field of Views. *Appendix 1 shows full color map.*

At the start of the project we were also provided with a list of potential equipment and standard colors which would possibly be used in the park. When Metropolia got involved in the project most of the planning was still on the board for the park, but to use the time efficiently modeling work already was in progress before official confirmations were created. This was so that they could have been readily placed in the Virtual Environment. The models were created accordance to the Helsinki Construction Ministry's park equipment guidelines [25].

The initial list of the park's equipment was the following:

- 2 Variations of Fences type A7, 1.2m and 1.6m
- Notice Board, code C2 of height 0.7m x 1m
- Various types of Benches D3, D8 and D12. It was not fully clear which were going to be used at the time.
- Trash-can, H2 Dark Grey

Figure 8 – Render of the Initial Models

Alongside modeling all of known candidates for the park equipment (Figure 8) I also conducted researching how the environment would run on Layar. The work aside the above list eventually changed, and Benches were dropped out in favor of tables and leaning posts.

4 3D Modeling Phase

4.1 Brief to 3D modeling

3D modeling is the art of constructing a mathematical representation of an object's surface, to create recognizable shapes. It can then be used in a myriad of applications, such as assisted design of industrial components, to creating virtual characters onto screen, or everything seen in a video game. If the object has to have textured, especially in a polygonal medium such as an OpenGL rendering engine, it should be first Unwrapped, creating a 2D image representation of the model. [26, 29-40]

This project required for elements of the dog park to be modeled in 3D, in order for them to be displayed by Layar application. It was also important to remember that this application would mainly run on a mobile device, and would be downloaded, thus optimization of polygons would be important.

Blender was the tool of choice for this. Originally developed as an in-house animation suite by Not a Number (NaN), it is now a free open-source 3D graphics software suite. It has been used to create various shorts, such as Big Buck Bunny, Sintel, and Elephant's Dream. [27; 28] It has robust polygon modeling tools, which I am familiar with. This helps with the creation of low polygon models, and quickens the pace as the tool doesn't have to be learned in the process.

4.2 Modeling the park equipment

To model the equipment in Figure 8, Schematics of the original Park equipment was used as reference. This was applied as a scaled background onto blender, and there after modeled using vertices placement and box modeling. This same technique was used on all models. Most of these did not require any texturing, thus simple coloring of components was sufficient. For this documentation, the modeling process for Park Bench D3 (Figure 9) was documented as it shows most of the modeling methods used.

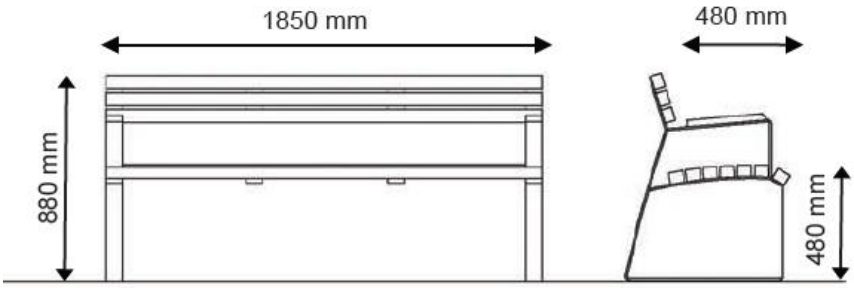


Figure 9 – Specifications for D8 Park Bench [25, 63]

The schematics were scaled to match to an agreed unit form within Blender. It is first considered that a distance unit (u) in Blender is equivalent to meters in the metric system. This doesn't really matter much as the object in question could technically be scaled to any size, but it helps to have all the objects scaled to a standard unit. A bounding Cube was first created with the scale of 0.88u x 1.85u x 0.480u as specified. The background image was the scaled to match this bounding box as close as possible, for both left and front perspectives. Once this was done, a missing 0.22m was added to depth of the bounding box to allow for rest of the bench to fit. As these are both sharing the same image, the scale can be applied properly on both.

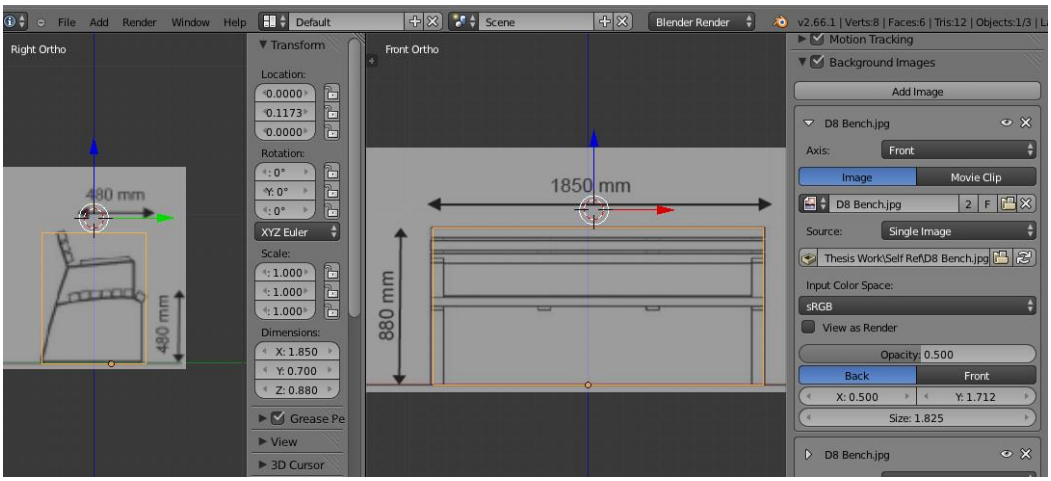
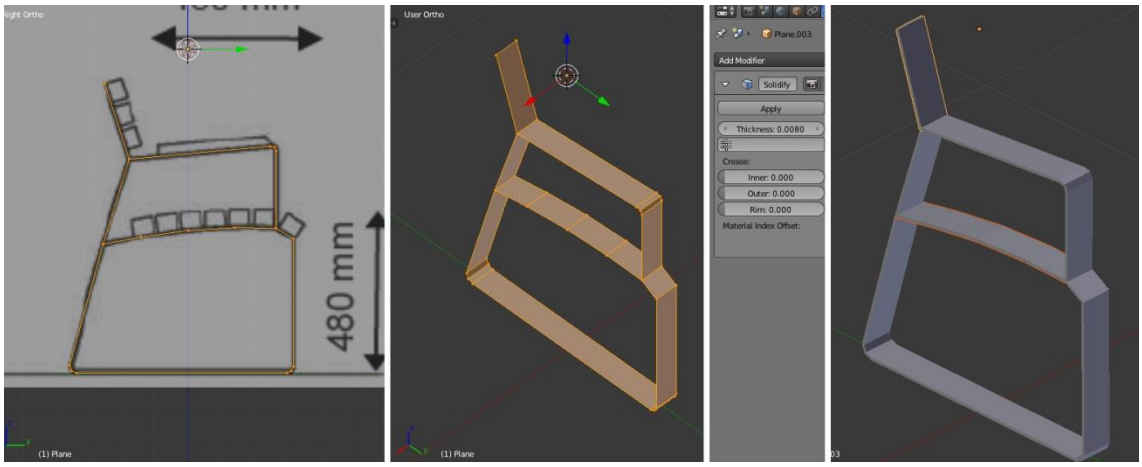


Figure 10 - Matching the background with the bounding cube using background images

As we may notice, the bench itself is quite simple, and can be mainly deconstructed into shapes of boxes. The most complex part of the model is actually the 'legs' where the bench keeps contact on the ground. According the specifications, these are made of 70mm x 8mm sheets of steel. Once the scaling of the background image was complete, the leg of the bench was the goal: it was more complex shape, so it was to be modeled manually.

This was achieved first by creating an edge map (Figure 11a), which was extruded by 0.070u to create 3D shape out of an infinitely thin planar surfaces (Figure 11b), Edges which had more than 2 quads connected were separated strategically. These would later be cleaned up to create a more solid mesh for rendering engines. All of these components would then have a Solidify modifier was applied, with a thickness of 0.008u or 8 mm (Figure 11c-d).



Figures 11 – a. Edge modeling
b. Extrusion
c. Solidify Modifier
d. Separation

After the Solidify modifier was assigned to the separated components, the overall mesh geometry was cleaned up to a condition in which all of the polygons would be convex, and had no polygons overlapping each other (See Figure 12). This is done to lessen the visual glitches caused be faces constantly competing on which is being used causing a flickering effect.

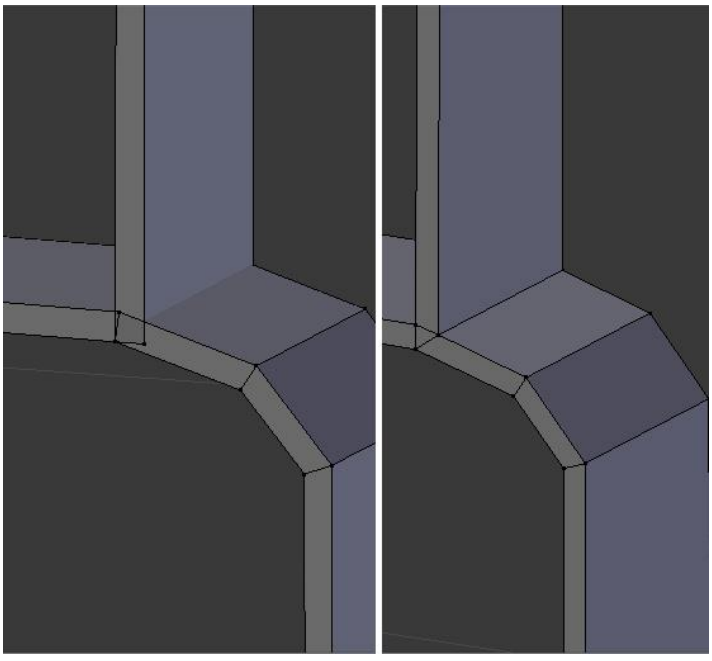


Figure 12 – Cleaning the Geometry - Manually fixing overlaps.

The rest of the model consisted of simple shapes. This required creating long rectangles, which followed the blueprint schematics on their placements. Sections of the arms/legs were used them to create the supports on the other part of the seat. Other simple shapes, such as the fences, also were created using long two sided planes, which had a transparent texture created in Photoshop applied. Once the shapes were created, simple materials applied to these shapes, and according to reference. Other equipment were created using the same above techniques.

Color	Color Code[25]	Hex Color	Guessed Reflectivity
Black Grey	RAL 7021	**233d4c**	Low
Black Green	RAL 6012	**003730**	Low
Steel	-	cccccc	High

Figure 13 - Table of Materials and their colors and reflectivity as set.
The Hex values were taken straight from the brochure using photoshop, but are not officially stated [23; 25,15]

Figure 13 shows the list of colors which were specified, along with converted hex values which were picked up from the documentation.

4.3 Modeling the ground

The ground was a more complex affair compared to the modeling of the park equipment. The topographic plan was used as reference to create a height map using the values given in the blueprints. The size of the cut out topographic map covers approximately ~145m^2. The lowest value, +2.40m is represented by black while the highest value, +5.35m is represented by white. The difference between these two heights is 2.95m. Everything in between were various percentage based shades of grey between these two values.

Figure 14 a. - Topographic Map [22]
b. Calculated Contour Photoshop Layers with Topographic Map imposed on top
c. Resulting Smoothed out Displacement Map

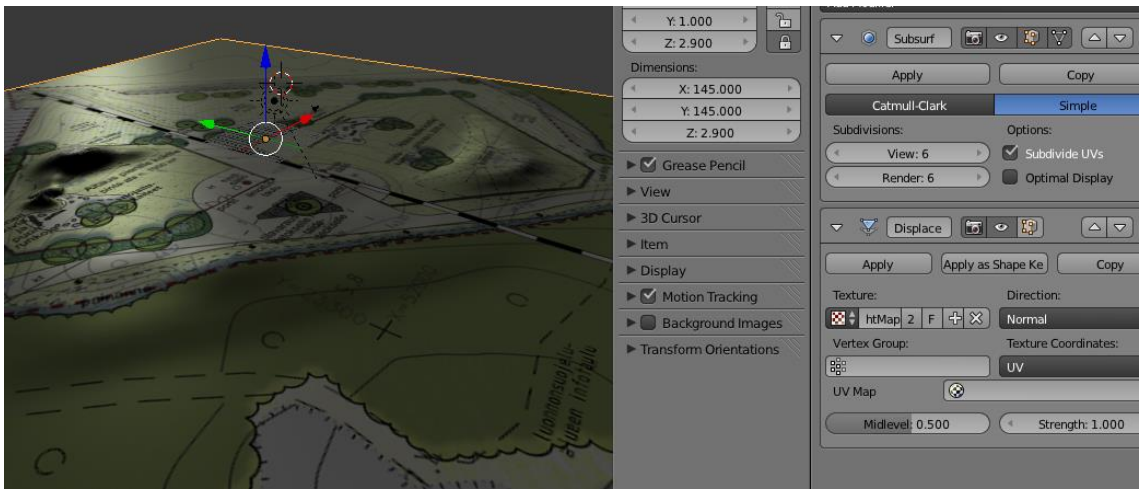


Figure 15 Applying Displacement Map to Subsurface Planar in Blender

First the different heights of the Topographic Map (Figure 14a) were referenced to create various height layers (Figure 14b). These layers were then blended together using a slight blur using Photoshop (Figure 14c) resulting in a usable displacement map. This displacement map was then applied to a high-polygon reference plane, which then was scaled according to the approximate 145u x 145u x 2.95u (Figure 15). Once it was scaled done, the existing topographic map was used as reference during a remake of the topology or retopo of the mesh to lessen the amount of polygons used (Figure 16) and to bring out some of the main features, such as where the fences and bushes are. This was done manually by snapping vertices on the surface of the reference plane.

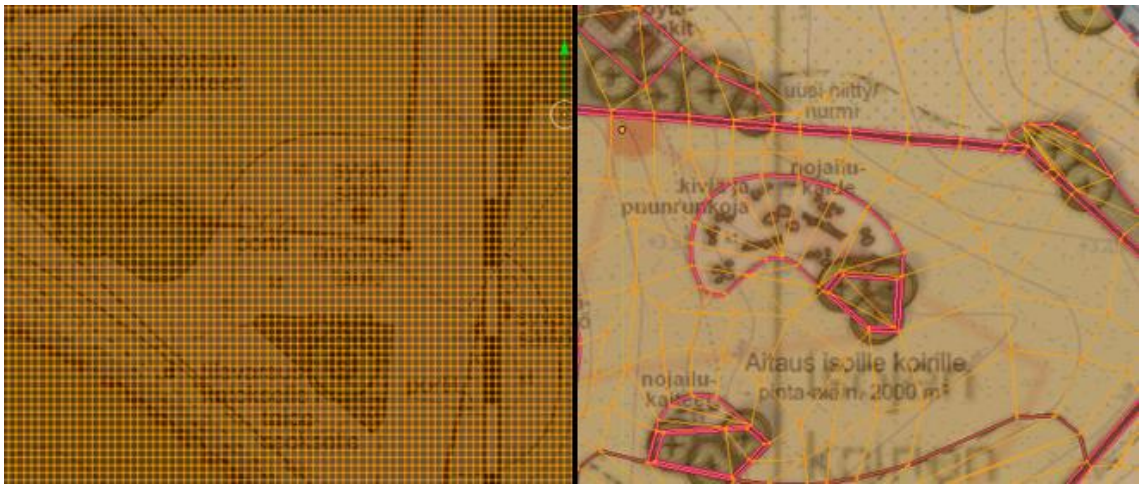


Figure 16- Comparison between High Resolution Planar and Retopo Mesh

Texturing the ground was quite straight forward. It was planar unwrapped, and there after a simple tiled texturing found from CGI Society's texture library [29] were used liberally. The Topographical map defined all the types of ground used, and this was used as reference, along with color directions of the materials that would be used.

4.4 Optimizing for Mobile Devices

The optimization and reduction of polygons for rendering engines is important, especially when considering that the models are to be displayed on mobile phones, which, while powerful have their limitations. This also applies to the textures as well. They should be of low resolution, with a suggested resolution of 256 pixels squared, especially as these images maybe downloaded through data connections.

Most of the accuracy from the ground plane (Figure 16) would be taken out when considering the processing power mobile device. The 65 thousand polygons would have to be reduced to approximately a thousand polygons for most optimal viewing. For Layar, this model would have been optimized even further as it would be downloaded using a mobile data connection alongside of being rendered on top of a feed from the camera.

Once the ground and textures were optimized, the park equipment was applied to the scene along with similar optimizations. Trees unfortunately had to be simplified to a 'ball on a stick' as they would otherwise have taken the majority of all the polygons on the scene, and would have taken a while to load up. Other various elements were also added to scene, including rocks and logs.

The plan is to have the models load up and offset by settings we set through Layar, similar to how Game engines handle their models.

5 Layar Development Phase

5.1 Layar Architecture

Layar is an AR service for iOS and Android mobile devices. Layar Vision is an extension of Layar, which allows for interactive virtual objects to be augmented on top of real world objects through Natural feature tracking, instead of Layar's usual gravimetric and geolocation based augmentation. By default, Layar is a service; which is accessed through a cross-platform application which requires an active data connection. This is due to everything is loaded and compared to information available from the external servers, where are Layar's service provide the information what servers are used in a layer. [30;31]

Layar's Point of Interests (POI) are handled through layers which can be toggled on and off by the user. These layers are created by other users. Layar requires layer creators to host their information on their own servers, while there are already commercial services available which would do the hosting for one. [31]

As seen in Figure 17, Layar works by first by having its Client, the Layar Player query information about the active Layer from Layar's Services. The Service returns general information about the Layer, and provides the address from which behaves as an API endpoint to provide further information on the Layer. This interface responds by returning JSON data (Figure 17), with links to assets used by the Layer.

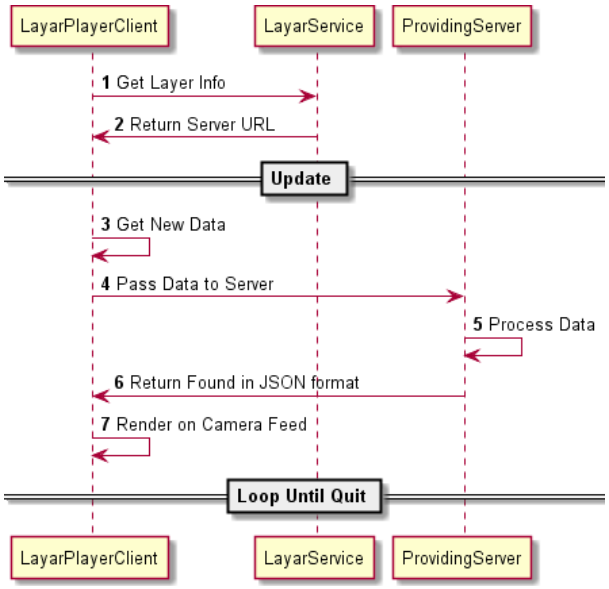


Figure 17 Simplified Sequence Diagram of Layar

In this case, we will create our own service (Providing Server in Figure 17) which hosts the information, but first the Layer has to be created in the Layar Service. To do this an account must be first created at http://layar.com. After this is completed, The creator has to provide the following information to create the layer: [32]

Layer name: unique id for the layer; it appears to having to be universally unique for the entire Layar service.

Title: Name for the layer which will appear to the end user when they are using the Layar player client to search for a Layer.

Short description: Short description of what the layer is, this is visible to the end user, when they are searching for potential layers through the Layar Service.

Publisher Name: Name of the publisher of the Layer. Visible to the end user and is searchable through the Layar service.

API endpoint URL: URL to the database API endpoint. This is important to link to the interface used by the Layar.

Layer Type: Selection of either Generic 2D or Mix of 2D and 3D inside a 3D space.

Vision Enabled: Disabled Layer Type to Mix of 2D and 3D inside a 3D space, and uses a slightly altered structure to define what reference image is being used. This reference image is uploaded to the Layar services for analysis, which then returns the "fingerprint" which Layar services can use to identify the image, linking it to a specific POI hotspot.

When a Layer is first created it is unpublished, and one has to request for approval, which after it will have to be published for public use. [32]

Layar version 6.0 Documentation [31] contained a tutorial which ended up with the following database structure as demonstrated by Figure 18:

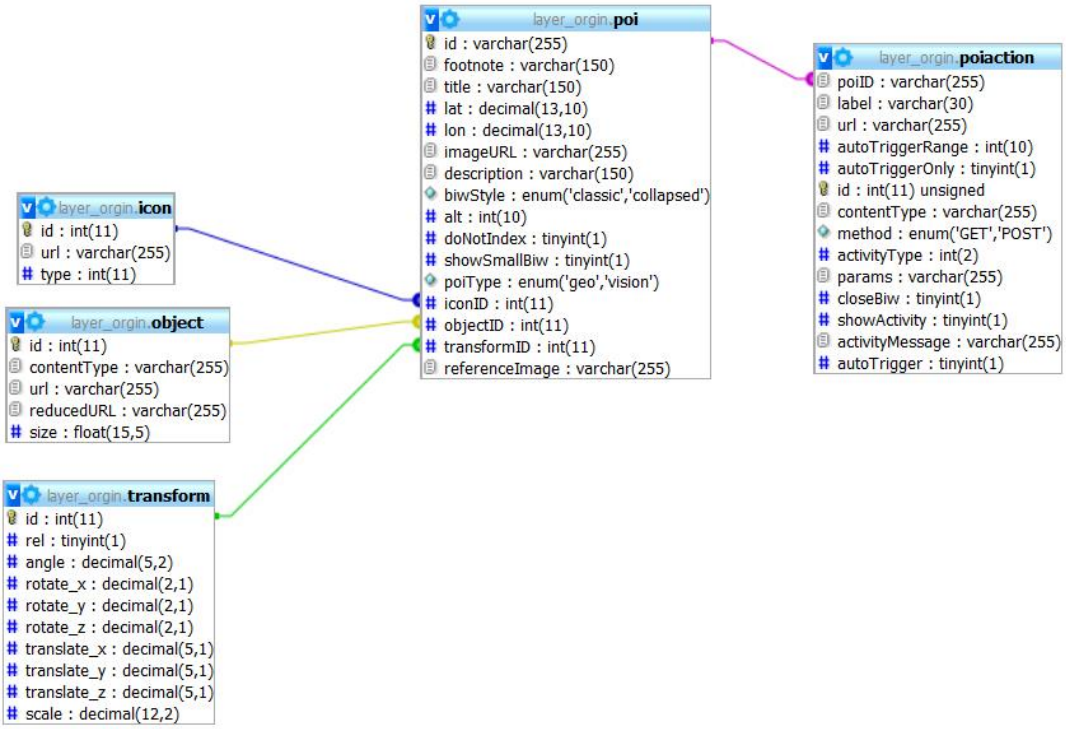


Figure 18 Layar 6.0 Tutorial Database Model [31]

The main focus of this database tutorial schema provided was to allow for 3 different types of Point of Interests with the possible actions for them:

- 1D Icons, represented by the *Icon* table
- 2D images, represented by *Point of Interest* (POI) table
- 3D Objects, expanding upon 2D images with *Object*, and *Transform tables*

When the a Point of Interest (POI) request is sent out by the client to the database interface, The POI Response must adhere to a structure, as defined by Layar [33] or the data will not be interpreted by the Layar player client. This structure has not received any major changes since the Layar 6.2 API, and should be compatible with the latest versions of Layar. [32] In short, there are only a handful of required components in POI response:

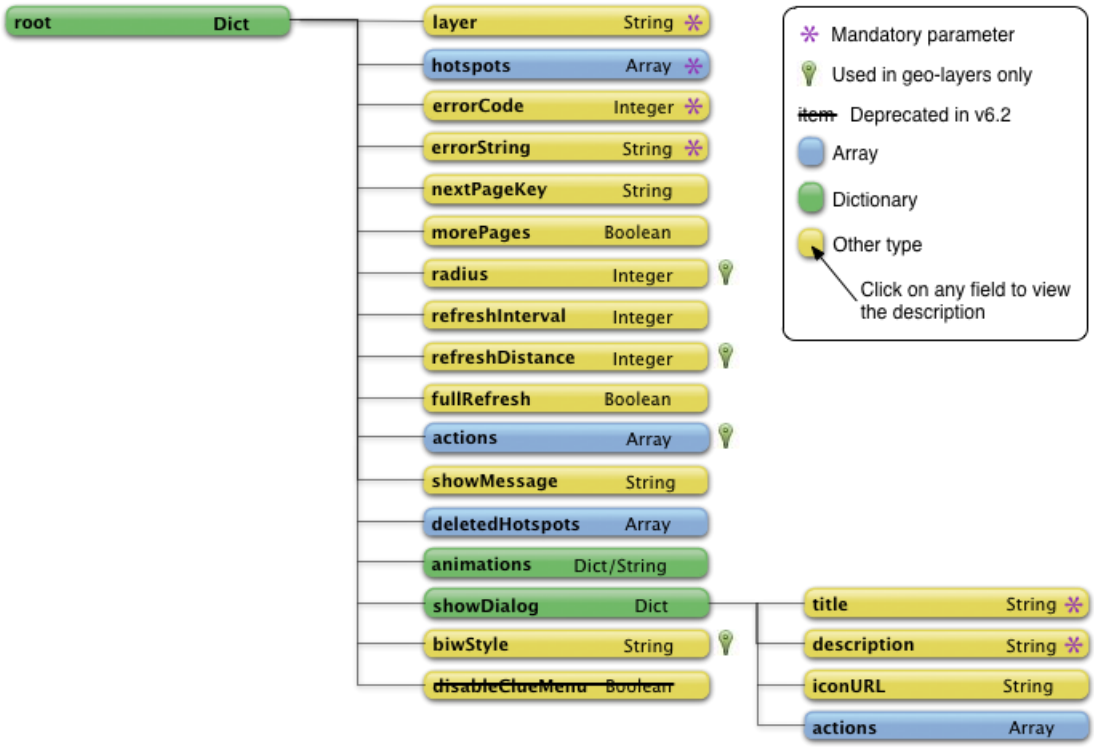


Figure 19 Root JSON getPOIs Response from Layar Documentation [33]

Figure 19 demonstrates that the properties, *layer*, *hotspots*, *errorCode* and *errorString* are required when defining a POI Response. *Radius*, and *refreshDistance*, and *biwStyle* on the other hand are required by layers using *Geolocation* instead of vision. The project will be focusing on those.

Hotspots parameter is an array of hotspot objects. *Appendix 2* shows the full extent of its structure [34]. To enable Layar Vision with 3D models for the layer, the POI Response must contain an *object*, which has an URI reference to a converted model; containing a *transform* parameter, that defines the *rotations* and *translation* of the object in relation to the *anchor* parameter, or image that is behaving as the tag which id is defined as a String in the parameter *referenceImage*.

While there are other features in Layar, such as Layar animations, the new 7.1 HTML widgets, more advanced Interactions and other various features, these features are not going to be further explored in the project as they are not required by the client. The service should be designed for core use only, while keeping possible expansion in mind for other projects that may use the service.

5.2 Content Management System (CMS)

At the time exploring Layar, the initial 3D modeling was complete, but no official word was available of the Park layout and what models would actually be used. During this time, a Content Management System (CMS) solution was also explored with Layar. CMS was planned to allow for multiple Layar layers and scenes to be handle through an interface, minimizing the risk of creating an incorrect JSON structure. This would allow changing models quickly, along with their size and orientation, and how one would interact with these models. That way, the work done in this research into Layar could be explored further in the future.

I had to take account the code already provided by Layar, and expand upon it. First a definition of initial requirements was drafted:

- The CMS would have Support multiple Layer maintenance without the need to write further code.
- Relational Updating, and automatic clean up on Layer deletion
    - If a Layer is updated, every entry which refers to it is updated with a single query
    - If a Point of Interest is updated, every entry which refers to it is updated with a single query

- Support for multiple types of Layar layers, from Vision to simple geolocation

Future features were also kept in mind, in case the project would be expanded upon into the future:

- Ability to adjust position of model through an interface
- Multilingual support with relational Updating
    - If a language is updated, every entry for that language is updated by a single query
- There can be multiple Actions per Point of Interest.
- User Groups, support for multiple user and user management

A PHP Model-view-controller (MVC) framework called CodeIgniter was selected as scaffolding to create the web application; ARCMS or Augmented Reality Content Management System.

5.3 ARCMS Database Plan

One of the first things which were noted is the fact that the project would attempt to recycle models but assign different transforms to the object. This means that POIs should be able to share existing objects, icons and transformations without having to create a separate entry for each one. Potentially, if there would be more point of interests, they could share assets, but have different parameters for them. This was solved by creating index tables; *poi-transform*, and *poi-object* which would point the 'point of interests' (*poi*) towards the respective assets; *transform*, object.

See Appendix 3 for Full Final Diagram of Database Structure which was planned

POIs would also belong to specific layers, thus POI would belong to a layer. This means POI should be identified by its *layerName* and *poiId*, making both of these values the index keys. If a Layer would be removed, all entries referencing to it would be removed from the database, saving maintenance. In MySQL this is done by adding cascading constraints on delete, as seen in Figure 20. The index tables such as *poi-*

transform, and *poi-object* would have similar constraints pointing towards the *layer* table, and the *poi* table.

```
ALTER TABLE `poi`
  ADD CONSTRAINT `poi_ibfk_1` FOREIGN KEY (`layerName`) REFERENCES `layer` (`name`) ON
DELETE CASCADE ON UPDATE CASCADE;
```

Figure 20 – Adding Cascading constraint to table *poi* onto the key *layerName* which points to table *layer* index key *name*, which if updated, or removed applies the same to this poi *layerName.* As generated by *phpMyAdmin*

As a side preparation for Languages, a language table was also created, which contained the different language ids, which would be referenced by different text tables, such as *title*, *description*, *footnote, activity* and *label*.

Most of the MySQL queries could easily be handled in PHP using CodeIgniter's database helper 'db', in which the class is extended by CodeIgniter's Model Class, *CI_Model*: (Figure 21)

```
Class objects extends CI_Model{
[…]
function getList($layername){
  $query = $this->db->get_where("object" , array( "layerName" => $layerName ));
  $ar = array();
   if($query->num_rows() > 0){
      foreach( $query->result() as $row){
        $ar[] = objects::toArray($row);
      }
    return $ar;
  }
  return FALSE;
}
[…]
```

Figure 21: PHP snippet from function *getList* from class *objects* Extending CodeIgniter's Model (*CI_Model*) class

However, the most important queries had to be written by hand. The first query for retrieving geolocation Point of interest was to determine which points of interests were in range, and combining all the data into a single table using a single query. The query also had to take account the distance between two geographical locations, one

provided, and the point of interest. This distance calculation can be achieved using *haversine formula*, shown in Equation 22.

Whereas:

$$haversin(\theta) = sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

$$distance = 2r \arcsin\left(\sqrt{haversin(\Delta\phi) + \cos(\phi_1)\cos(\phi_2)\, haversin(\Delta\lambda)}\right)$$

$\phi_1, \phi_2 =$ latitude of point 1 and latitude of point 2 in radians $\quad \Delta\phi = \phi_2 - \phi_1$

$\lambda_1, \lambda_2 =$ longitude of point 1 and longitude of point 2 in radians $\quad \Delta\lambda = \lambda_2 - \lambda_1$

$r = radius\ of\ sphere.\ In\ this\ case, The\ Sphere\ is\ mean\ radii\ of\ Earth\ = \sim 6371km$

The result is returned as the units used for the sphere itself.

Equation 22: Haversine Function and Finding Distances between two Coordinates on a Sphere [**35; 36**]

Usually, the geographic coordinates are provided by the devices themselves. Today, most geographic coordinates are presented in the World Geodetic System format, or WGS84. While developing the point of interests, we have to manually input the geographic coordinates. Before we can input the coordinates however, we may have to convert them from *Degree Minute format* to Decimal format (Equation 23).

$$D + \frac{m}{60} + \frac{s}{3600}$$

$$D = Degrees$$

$$m = Minutes$$

$$s = Seconds$$

Equation 23: Converting Degree Minute Format (DMF) to Decimal Format (DD) [**35**]

When the degrees of latitude or longitude are of East/North the degree value is positive. If it is West/South, the Degree value is negative. This especially comes important when using the haversine formula (Equation 22) to find the distance between two geographic coordinates. Geographic Coordinates have to be converted into decimal for the query. The final MySQL query adapted from the haversine formula to find to distance the two geographic coordinates is seen in Figure 24.

```
SELECT ASIN(
   SQRT(
    POWER(
     SIN( (RADIANS(lat2 - lat1)) /2 )
    ,2) +
    COS( RADIANS(lat2) ) *
    COS( RADIANS(lat1) ) *
    POWER(
     SIN( (RADIANS(lon2 - lon2)) /2 )
    ,2)
   )
 ) * 2 * 6371
AS distance
```

Figure 24: Simplified Snippet of Haversine SQL Query in *RetrivePOI* function from *poi* object;
*See **/application/model/poi.php** for the Full Source code for POI filtering and retrieving Queries.*

Figure 24 was used in favor over the tutorial calculation provided by Layar [5;37], as there were unnecessary mile->kilometer conversions involved in the math. The queries become simpler when querying for a Vision Layer, as the geolocation is not really needed for them. However, the image 'fingerprint' from the Layer service, identifies which image id is being used, which has to exists in the database.

5.4 ARCMS Software Architecture Plan

As ARCM's was to use a Model-View-Controller framework, CodeIgniter. Model-View-Controller Architecture pattern (MVC) comes from its use of 3 different component types, the Model, View and Controller. *Models* control data, enforcing various rules for it, while maintaining a state for the application being either temporary or permanent. *Views* Generally generate the user interface based on the model, while *Controllers* receive input, usually an external one from a user, which then interacts with a model deciding by logic on which view is used. [38, 29-35]

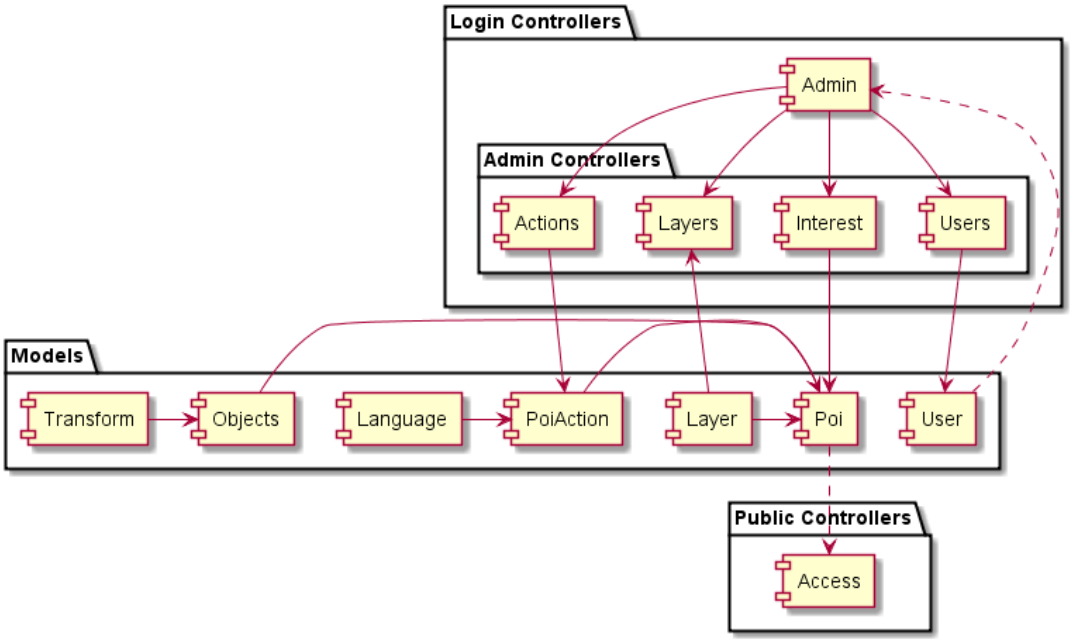


Figure 25 - Planned Architecture between the Models and Controllers

This architecture was expanded upon to create two different types of controllers: Restricted and Public. It consisted of a Login Controller, Admin; along with Admin Controllers, Actions, Layers, Interest, and Users. A Login Helper was designed to be used as an interface to all Admin Controllers, using a Login Helper which would check if the user is logged in or not. Only after logging through the Admin Controller would a user access Admin Controllers. Otherwise the user would be redirected back to the Admin Controller (Figure 24). The Models would be the only direct interfaces with the database.

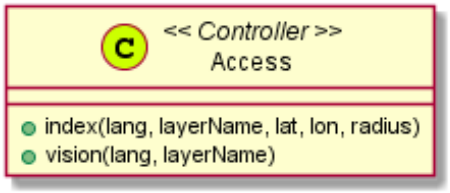


Figure 26 - Access Controller UML

The Access controller would be the only controller which could be accessed externally without login. It had two actions, index, and vision. (See Figure 25)

Index action receives the parameters to identify a geolocation point of interest. The Parameters latitude (*lat*) and longitude (*lon*) contains the current location of the client, while *layerName* would be used to select the currently active layer. Along with this, the Layar Player Client provides a user defined *radius,* which is used to filter out Point of interests far away. This is equivalent to the Layar Documentation getPOI Request.

Visio action receives only two parameters the language (*lang*) being used, and *layerName* which is used to select the Layer in use. It would return JSON, which would list all data available for the selected Layer. Technically both of these actions could be combined, However the Layar Tutorial suggested keep the actions separate as later versions may not support this [39].

As this software architecture plan was made, rough prototype scaffolding was created, to test out how well Layar works. The prototype did not have any of major required features from the architecture, other than simple access to creating and removing point of interests to and from the database. The index action works as expected, but vision had to be tested still to assess if the application should be continued further. This required us to test using a test 3D model as at the time, the park was not ready. However as we were going to deal with 3D models, Layar requires the conversion of the 3D models into Layar's proprietary l3d file format.

5.5 Findings of Model Conversion

I found the Layar documentation of the model conversion process lacking, as it did not contain any detailed instructions, other than how to use the application; most of the details are scattered around the Layar documentation site thus most of the results here are through experimentation. Due to this, many of the results found during this project may be subject of change in the future. However, as a look back of one year since the project was finished; there has not been any update to the conversion process. The Version of the Converter has been 3.0.1 in March 2012, and only updated by a few minor patches to 3.0.3 in August 2012 [40].

To use any 3D models in any of the Layar layers, one has to first convert the 3D models to proper format. First, the 3D model must be exported from 3D suite into

Wavefront Obj format. If one wants to use textures, the model must be properly UV unwrapped. Thereafter it has to be imported into Layar provided model conversion tool [41] from which it is converted into Layar's proprietary l3d file format. Unfortunately the specifications of the l3d file format are not public information thus could not be deconstructed or documented. [32]

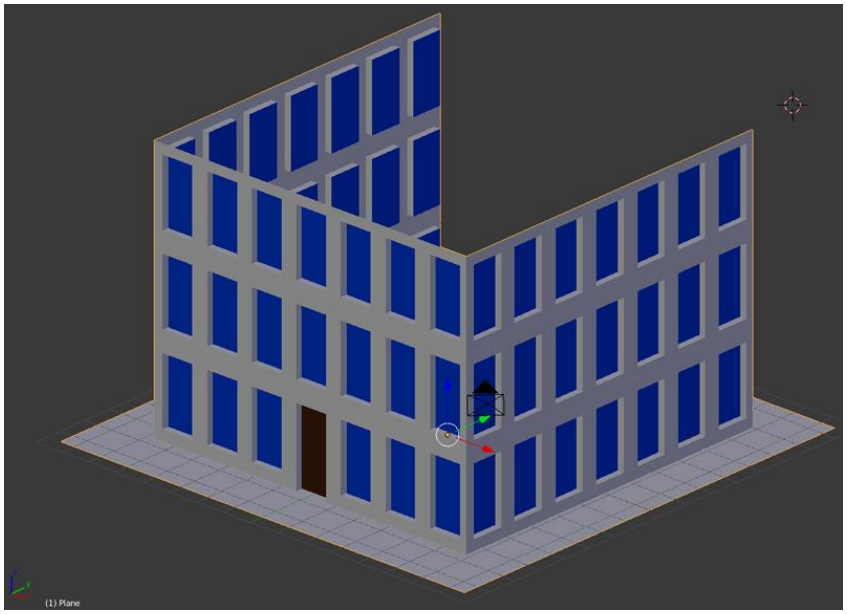


Figure 27 - Test Model before exportation from Blender

Figure 27 shows a quick test model which was created in Blender; it was later exported with textures and materials to Obj format. In this case, note the axis reference of the building, as the front of the building is facing towards *negative* Y axis. This is due to the fact that the perspective facing the *positive* Y axis is considered as front by Blender. Using apply rotation and scaling tool, using blender hotkey: Ctrl + A + 4, the model is normalized into a scale of 1 x 1 x 1, which is converted 15u x 15u x 9u. This can then be scaled in Layar without distorting the model. The default Obj export settings for Blender version 2.66 are *–Z Forward*, and *Y Up.*

The Layar 3D Model Conversion program is then started. It is a simple java application, which loads the program, and can do simple manipulations to it. At first glance, everything in the application is disabled on first load, except for the import through "File Menu" > "Import Wavefront (.obj/.mtl)" menu option (Figure 28).

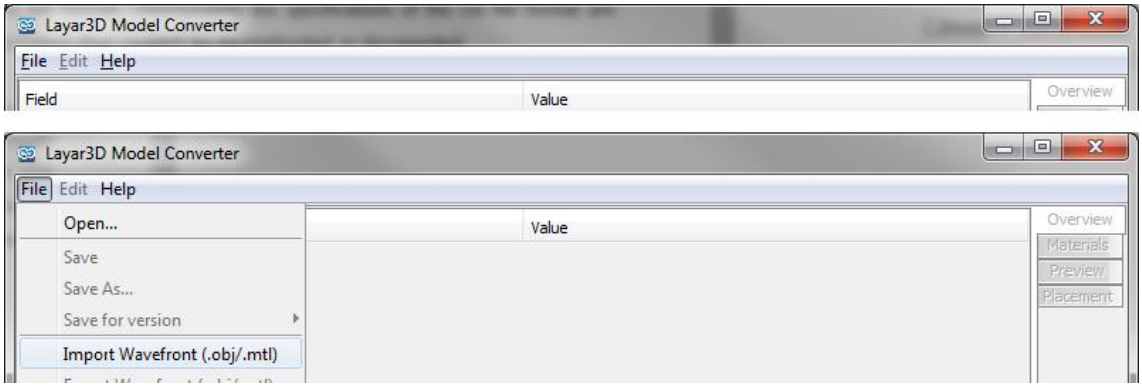
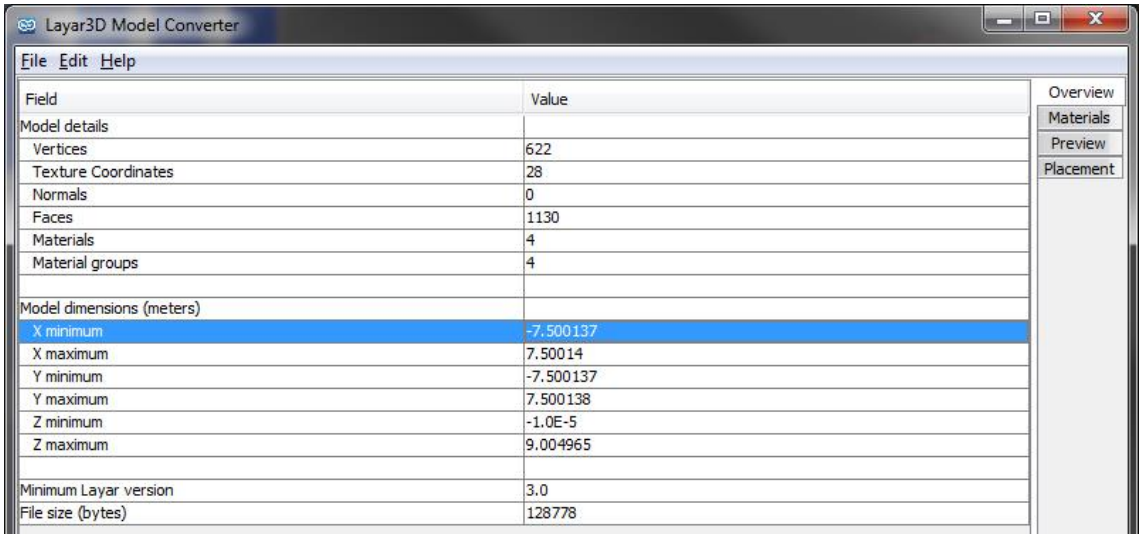


Figure 28 - The Overall Window and Opening the File Menu

Once the Obj file is selected and loaded, the next screen (Figure 29) shows some general statistics out of the model itself, and is generally good when considering optimizing the model or working on cross compatibility with older versions of Layar. Typically, the most important values one has to pay attention when working on mobile models is file size, and the amount of vertices used. Mostly these go hand in hand. But this example it is only a test model so it is not necessary to optimize these.



Field	Value
Model details	
Vertices	622
Texture Coordinates	28
Normals	0
Faces	1130
Materials	4
Material groups	4
Model dimensions (meters)	
X minimum	-7.500137
X maximum	7.50014
Y minimum	-7.500137
Y maximum	7.500138
Z minimum	-1.0E-5
Z maximum	9.004965
Minimum Layar version	3.0
File size (bytes)	128778

Figure 29 - Model Information Screen

When the model was previewed using the preview tab, the model was facing the camera while having some of the color textures (Figure 30a). If the model happens to be oriented oddly, the settings on export are different. Occasionally the textures might also be missing, however you can still fix this manually for each material through the tool (Figure 30b) by manipulating the drop down menus to set the textures per materials.

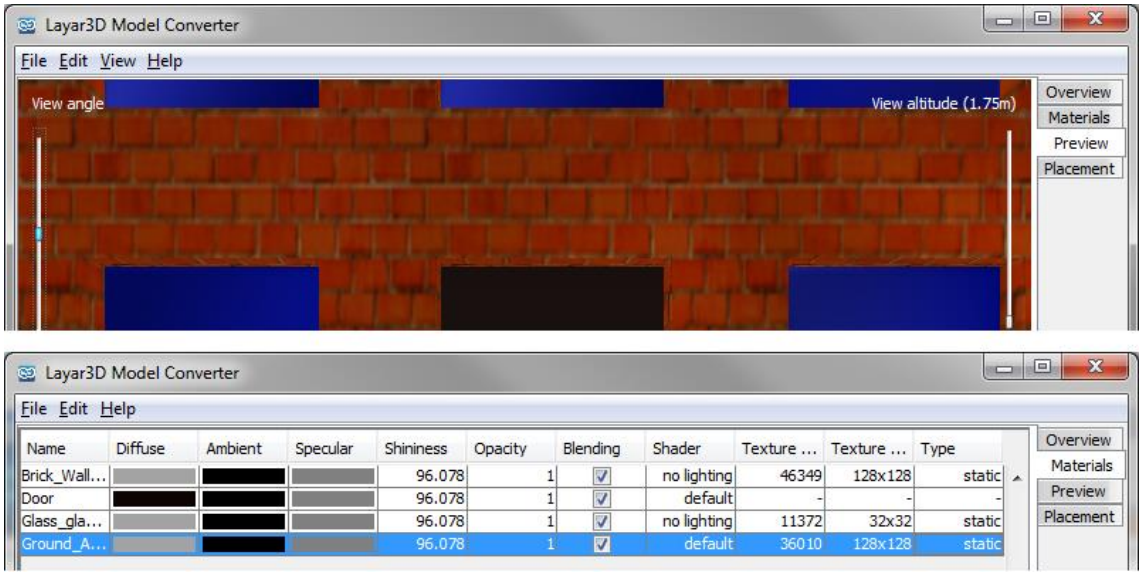


Figure 30 - a. Preview Tab.
b. Materials Tab

The Preview window doesn't seem to have a manipulation option, other previewing the model from different angles using an OpenGL renderer which behaves similarly to the renderer in the Layar Player client. Once the Model was been exported (saved as) into l3d format it can be uploaded to a remote service, with the address to the model added to the database using the rough prototype that was created in Section 5.4. To test the rendering capabilities, the model was reduced to a scale of 1:25, converting the 15u x 15u x 9u model into 0.5u x 0.5u x 0.36u, was applied to the Z axis of the model to view it from a theoretical distance of 2.5m, instead of straight up close.

A Test Marker added in Section 5.1 was put on a display and there after scanned and viewed through the Layar Player client on an Android 2.3.3 *Galaxy S1* and an Android 4.0.4 *Galaxy Note*. This was also repeated using an Android 4.2.2 *Nexus 10*, a year later with the results observed in Figure 30a-30b. Unfortunately iPhones have not been available.

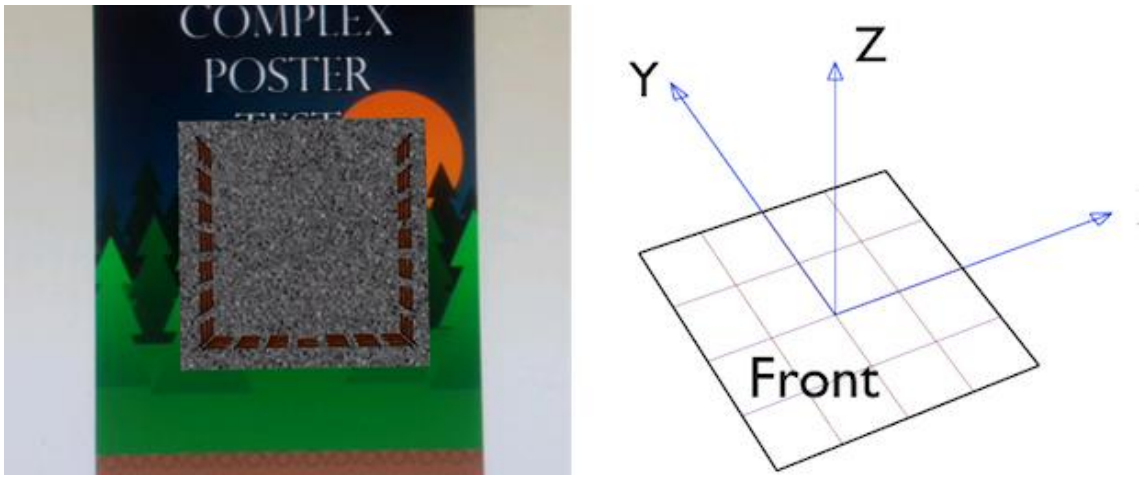


Figure 31a. Poster observed through Layar interface on a Nexus 10 tablet.
31b. Layar documented Axis's, with the Front representing the Poster/Marker, with the Y Axis being Up [32]

Taking the results into account, as we want to achieve a "window-on-world" type of effect, the following is observed:

- The Negative Y on the model axis must become the positive Z Layar Vision camera axis, or the Forward.
- The Negative Z on the model must become the positive Y Layar Vision camera axis, considering it as Up.

Figure 32 – Model with Converted Axis's facing the viewer

From this, it can be deduced that the model must be exported from Blender by adjusting the Obj exporter to export -Y as Forward, while -Z as up, flipping the Default values while Z is negative instead of positive. See Figure 31 for fixed rotation. This was demonstrated to the client on a status report, and they seemed content with it.

5.6 Issues with Layar and Scaling

A tracking issue may arise with images that do not have enough contrast or good lighting, causing jitter moving the model around incorrectly. However, it was observed that even with the best possible case, where and image was coming from a lit source, straight from the screen this jitter was not reduced. This indicated that there might be an issue with scale. With a theoretical distance of 2.5m, of a model nearly half meter cubed, there was very noticeable movement involved in the position of the model compared to the marker. This can be seen in the positional difference of the square between Figures 31a and 32. This was confirmed when the model size was increased, along with the distance the issue become more noticeable. Reverse was true as well, thus the smaller the model was, the less it would jitter and move about.

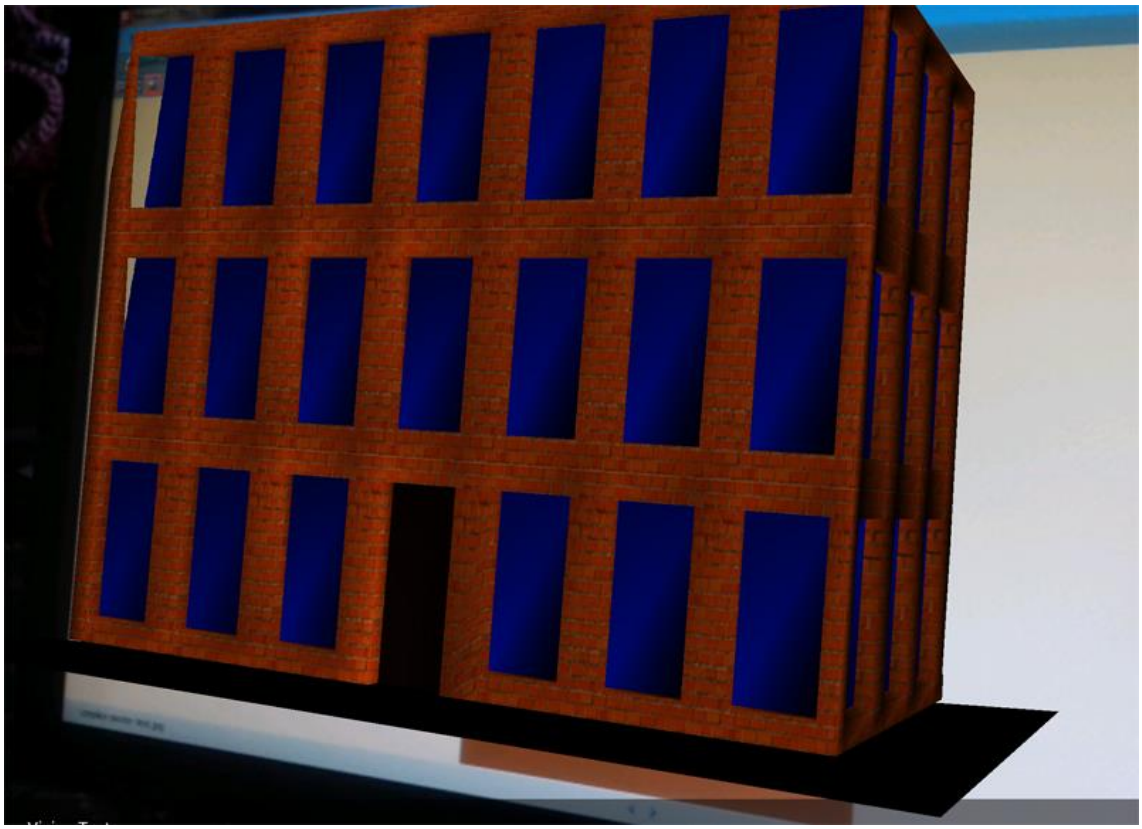


Figure 33 – Layar's Rendering Distance Restriction demonstrated

On further tests however, multiple showstopper were discovered. When the model's scale is made larger, the Jitter noticeability increased as there was a much larger difference in depth. The other was the observation that Layar Player's renderer is restricted to rendering anything within a range of 20 m. This was tested by making the scale of the model 1:1; with the model offset by -10m x, -3m y, -20m z. Figure 33 demonstrates this effect: notice how part of the left corner of the test building is not rendered, while 4 columns of windows are missing from the right side, cutting the model right after the 20 m mark.

This was an *undocumented and unexpected restriction*. Time was limited, and with this discovery Layar was found to be unfeasible to be a solution as the project area in question is around 145m^2. A Backup plan was quickly created and initiated. As the project goal was to create a working solution for visualizing property, another choice was to make it using Unity 3D Engine. This change was then completed within a week before prototype deadline.

6 Unity3D Development Stage

6.1 Considerations and Importing Models

The switch to Unity3D was chosen due to timing. During March-April of 2012, A limited promotion was available in which free basic mobile and flash publishing licenses were provided by Unity Technologies [42]. As of Unity 4 in 2013, they have been providing basic mobile licenses with the basic version of the platform.

I've also had previous experience with Unity 3D engine from a previous experience participating in the 48 hour Ludum Dare Hackaton. As I had limited time available when starting the new prototype, it had to be developed quickly. Fortunately, the Unity provides nearly native tools for model importation from blender, thus I had to mostly learn how to make the world interactive.

The Model importation process is simple. One first requires either Blender version 2.45-2.49 or versions 2.58 or later. The native file created by Blender, can be directly saved into the Assets folder of the Unity project at the same time while Unity is open. After the files are saved, and the project is loaded up, or if the Unity program is already open, the files are automatically converted by Unity. This means modifications done to the model in Blender show nearly in real time in Unity. Some of the optimization that was required from minimizing the file size of the models was reverted, increasing the polycount of the models. However other optimizations were now be done regarding assets management.

First, any model that duplicated in the scene would now simply be an instance to a single mesh, so that only 1 model asset would be used per each unique model. This could then be scaled, and transformed uniquely per each object, creating more variance in the scene. This instancing would be done in Unity, through creating 'prefab' objects into a project. A prefab would be identifiable by a blue color coding of the name of the object in the hierarchy window in Unity (Figure 34), referencing to an object in the project window.

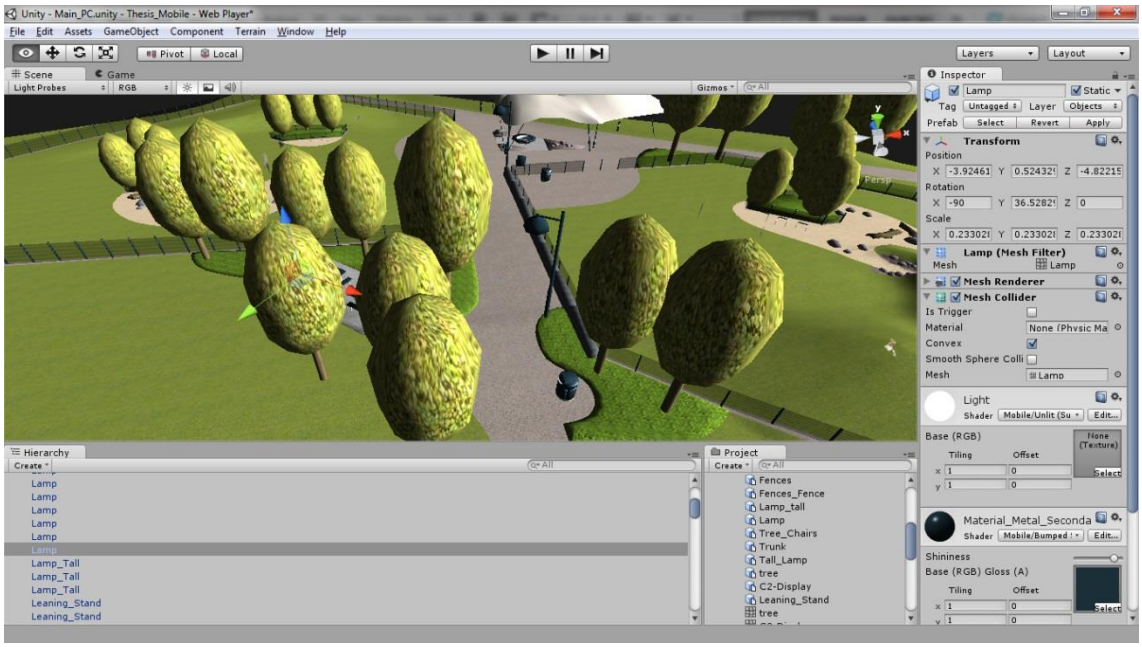


Figure 34 - Unity Interface and Example of Instancing of object *Lamp_tall*

Another change that had to be done to the model was an addition of a collision box to the environment. Because the user would now be completely detached from the real world, only being virtually present, they could easily move off edge of the map and fall infinitely. To avoid this, the border of the map was extruded up, and made invisible to behave a block to avoid the user from dropping off as seen by the wireframe in Figure 35.

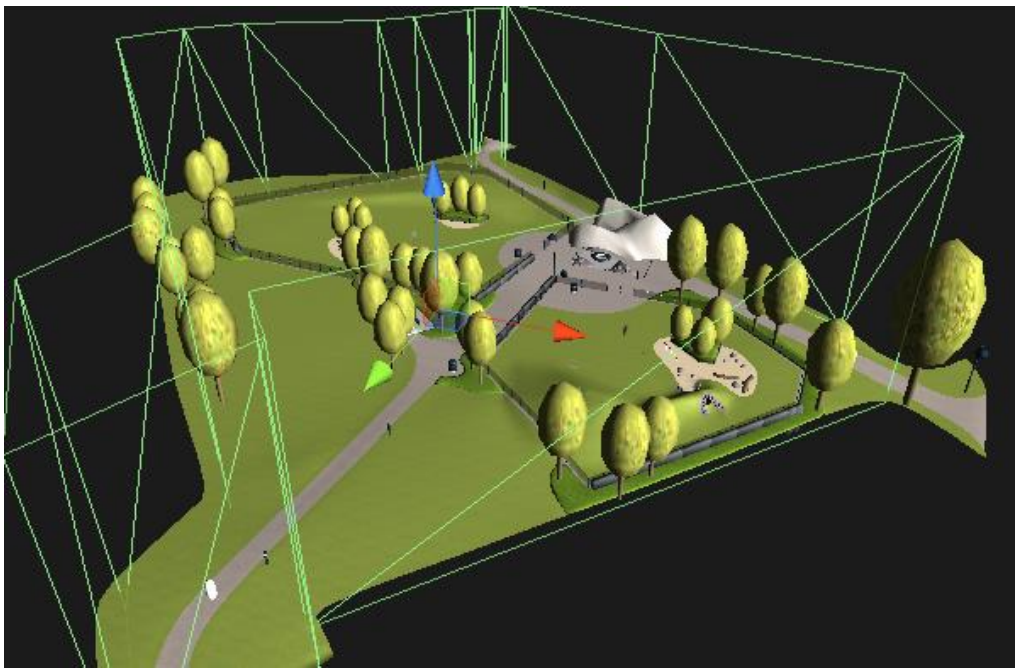


Figure 35 - Park replicated in Unity with Borders defined by wireframe.

6.2 Creating the Cube map

The final aspect that had to be done was to create the sky dome, which would mimic the real world environment around the site. This was done by going on scene where the park would be constructed at, and creating a panorama by taking up to 18 photos of the environment from different directions using a 3-way pan tilt tripod and a Canon Powershot G12. While this is not the most optimal way of doing this, as a proper mapped panorama would to take account the lens distortion, however these tools were not available. The set of images would then be patched up into a panoramic view using Photoshop, making sure there were minimal amounts of edges at best. The most difficult part of this was to actually the grass as it was too noisy to achieve a reliable point of reference.

Figure 36 –Photoshop manipulated Panorama of the Environment

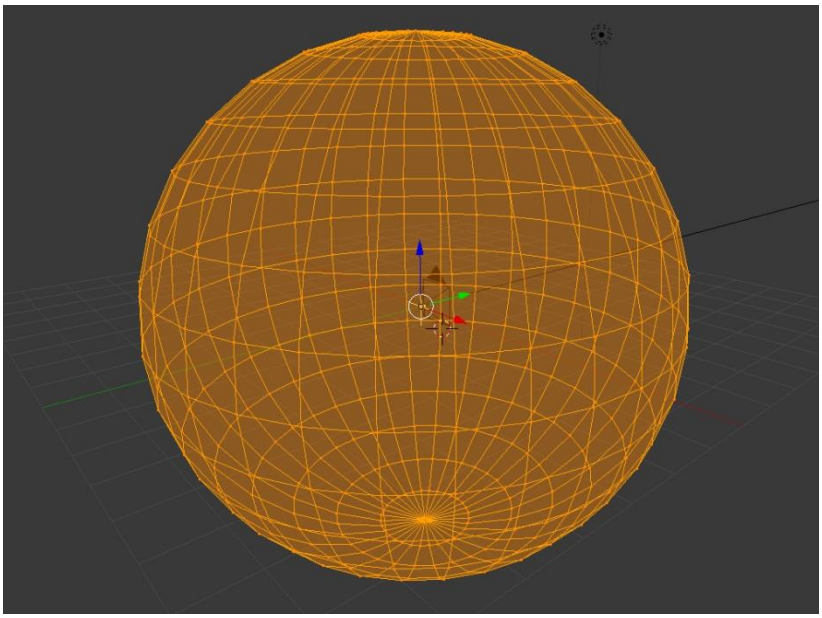


Figure 37 - A 90 degree camera inside a sphere is animated to axis, only pointing towards them when rendering

This is not however enough when creating the skybox, as the panorama has to be converted into a cubic format, so that It can successfully be applied to a cube, known as a cube map which would behave as the sky dome. This can be done by creating a sphere, mapping the panoramic view onto it, and then moving the camera with a 90 degree field of view inside a sphere (See Figure 37), and rotating it in each axis, in each direction, creating 6 images. [43]

6.3 Interactions

Unity provides ready set scripts for to enable for quick prototyping. One of these is the first person controller or *FirstPersonController*, which maps keyboard and mouse controls to an object, and applying physics to it. An extension of this allowed for touch interfaces, along with a 'virtual' controller on the touch interface, allowing for movement using such devices. These controller scripts were modified so that the user would only move around the environment, only having a restricted ability to look around in the horizontal axis as vertical viewing was not required.

The Client had requested for another feature after demonstrating the first prototype of the Unity version, as the view was to show what an area would look like with different types of objects:

- *A 'designer' tent in the middle of the park*: What would the park look like with a tent, or a tree at the crossroads? Unfortunately no ready-made design was available, so one was quickly conceptualized using existing styles.

- *Fenced stones*: What would the park look like if the line of sight to the dogs was disrupted by fenced stones? This would theoretically keep the noise down from the dogs as people would pass by the park.

This was done by creating buttons onto the interface, which would toggle the visibility of these objects, as defined.

6.4 Deployment

Deploying to the various devices is fairly straightforward from Unity. Once the project was complete, one only had to build the project accordance to the devices selected. Unfortunately it was found out too late that building a Unity application into iOS required access to an Apple Developer account along with a Macintosh Operating system running Unity with the same license. As I did not have access to one, I was restricted to deploy only to Android devices and Desktop platforms through Flash. A separate file was created for either mobile or Web release, due to the difference in controls.

As the final software was a rough prototype, it was not released over Google play, but was provided through 3rd party means, and is now available through a flash player at http://users.metropolia.fi/~mattila/artova/ or through the android client available at http://users.metropolia.fi/~mattila/artova/artovakoirapuisto.apk.

Once the binaries were completed, there were downloaded onto 3 separate Android devices, Samsung Galaxy S, Samsung Galaxy Note, and eventually on Nexus 10 a year later its completion. The older devices had a smaller frame rate, but this improved with each software update as well as newer hardware.

7 Conclusion

It has been over a year since the software side of this project was completed. While the project was partially successful in achieving in its goal of visualizing property design, it was disappointing to discover the restrictions in the Layar Service. For a period of a year I tried to see if any update would happen to fix the issue regarding rendering distance, to no avail. But technology moves on and develops further, and more uses are created for the existing ones.

The experiment of using Layar as a platform for visualization property design was a failure, and it is now easy to conclude that Layar Vision is designed to be used with print media augmentation, allowing for interfaces or small objects to be displayed on top of the prints providing secondary information or media. I realize that the way do the 3D visualization could have for-example been done from a birds eye perspective, by super-imposing a scaled down model reference tag. This would have required more accurate camera tracking of tags.

The primary idea, explored in this work also failed. If Layar did not have the restriction of the rendering distance, it would require more accurate cameras and better tracking software to achieve less jittery result in rotation and transformation of large objects during natural movement of a held device. In the future, this might be completely possible but today a true marker based tracking would be a more accurate for visualizing property. However this is expensive in terms of development and was not feasible when this project was started.

It is disheartening to see how much of my preparation and planning had to be abandoned on the discovery of the limitations with Layar Vision to meet my deadlines. Although this work is documented in this report, the CMS was not continued any further and was left in a rough prototypal stage. While I did quickly adapt to the change, the quick pace affected my work morale. Fortunately I did not have to do many changes to the 3D visualization of the park, which allowed me to speed things along. However, after completing the software project to a releasable state, I ended up having a hard time finishing this paper due to my morale. In stark contrast, the client was still pleased with the solution even though it was not as expected. The

visualization was still used by Artova to display what the surroundings might look like to its members, and to possibly get backers to the construction project.

It was surprising to note that this report might be the first documentation of the 20 m limitation of the OpenGL renderer in Layar. While it was the correct course of action to try out the Layar service, I find that I should have ran more tests before assuming that all the functions would work. The problem with the view distance in Vision is still present with the newest versions of Layar as of May 2013. As a side note, the markers for Layar Geolocation appear to behave differently as they are scaled according to distance, instead of actually being placed at the point where they are relatively speaking staying close.

This project has been a learning experience, mainly through failure. I realize that the topic should have been a bit more focused to reduce possible variables that might cause the structure of project to fail. I suggest future aspiring students to break down their projects into smaller, more focused and testable components.

Fortunately with my previous knowledge on the Unity tool kit, I was able to quickly create a very rough prototype existing work I had done for Layar. Much of the work for the prototype was already created, as ready-made packages for controllers and the 'player characters' could be used only with minor modifications; the time I spent doing this was kept to a minimum. I do find a continued interest in Virtual Reality. I suspect that if the project had been kept more focused to creating a virtual environment instead of developing a web service around an existing augmented reality service; I would have expected the final prototype to be of much better quality, and perhaps focused on creating experience using Human mountain displays. It will be interesting to see the resurgence of HMDs through an upcoming commercialized product called Oculus Rift in the future [9;10].

8 References

1. Moore GE. Cramming more components onto integrated circuits [online]. Electronics 1965;38(8):114-117.
   URL: http://web.eng.fiu.edu/npala/EEE5425/Gordon_Moore_1965_Article.pdf
   Accessed September 11 2013.

2. Brey P. The ethics of representation and action in virtual reality [online]. Ethics and Information Technology. Netherlands: Kluwer Academic Publishers; 1999.
   URL: http://realities.id.tue.nl/wp-content/uploads/2010/03/brey-1999.pdf
   Accessed May 29 2012.

3. Brey P. Virtual Reality and Computer Simulation [online preprint]. Netherlands: University of Twente; 2008.
   URL:
   http://www.utwente.nl/gw/wijsb/organization/brey/Publicaties_Brey/Brey_2008_VR-CS.pdf
   Accessed May 29 2012.

4. Wuebbling M. Mobile graphics moving towards console level [online]. Nvidia Blog. April 20 2012.
   URL: http://blogs.nvidia.com/blog/2012/04/20/mobile-graphics-moving-toward-console-level/
   Accessed May 29 2012.

5. Madden L. Professional Augmented Reality Browsers for Smartphones: Programming for Junaio, Layar and Wikitude. 1st ed. United Kingdom: John Wiley & Sons Ltd; 2011.

6. Virtual. In: Oxford Dictionary [Online]. Oxford, UK: Oxford University Press.
   URL: http://oxforddictionaries.com/definition/english/virtual
   Accessed May 29 2012.

7. Milgram P, Kishino F. A Taxonomy of Mixed Reality visual displays [online]. IEICE Transactions of Information Systems 1994; E77-D(12).
   URL:
   http://www.eecs.ucf.edu/~cwingrav/teaching/ids6713_sprg2010/assets/Milgram_IEICE_1994.pdf.
   Accessed May 29 2012.

8. Onyesolu MO, Eze FU. Understanding Virtual Reality Technology: Advances and Applications [online]. In: Schmidt M, editor. Advances in Computer Science and Engineering. InTech; March 2011.
URL: http://www.intechopen.com/books/advances-in-computerscience-and-engineering/understanding-virtual-reality-technology-advances-and-applications.
Accessed May 29 2012.

9. Oculus Rift virtual reality headset gets Kickstarter cash [online]. BBC News. August 1 2012.
URL: http://www.bbc.co.uk/news/technology-19085967.
Accessed August 3 2012.

10. Oculus VR. Oculus Rift: Step Into the Game [online]. Kickstarter. Sept 1 2012.
URL: http://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game
Accessed August 3 2012.

11. Leap Motion unveils world's most accurate 3-D motion control technology for computing [online]. Marketwired. May 21 2012.
URL: http://www.marketwire.com/press-release/leap-motion-unveils-worlds-most-accurate-3-d-motion-control-technology-for-computing-1659460.htm
Accessed January 8 2012.

12. Kipper G, Rampolla J. Augmented Reality: An emerging technologies Guide to AR. 1st ed. Kindle e-book. Syngress. December 3 2012.

13. Michael Phelps Wins Gold – Men's 100m Butterfly Full Event [online]. Youtube.com. August 3 2012.
URL: http://www.youtube.com/watch?v=X7bj_LUIY7Y
Accessed May 21 2013.

14. Braiker B. Google Glass News: a new way to see the world [online]. The Guardian. April 5 2012.
URL: http://www.theguardian.com/world/us-news-blog/2012/apr/05/google
Accessed May 18 2013.

15. Online EAN barcode generator [online].
URL: http://marin.jb.free.fr/barcode/
Accessed September 17 2013.

16. QR Code [online]. Wikipedia. 15 September 2013.
URL: http://en.wikipedia.org/wiki/QR_code
Accessed September 17 2013.

17. Cawood S, Fiala M. Augmented Reality: A Practical Guide [online excerpt]. 1st ed. Pragmatic Bookshelf.
URL: http://media.pragprog.com/titles/cfar/intro.pdf
Accessed May 18 2013.

18. Layar – Layar Vision Explained [online]. Layar AR. Youtube.com. August 1 2011.
URL: http://www.youtube.com/watch?v=8KFjlpnMhmw
Accessed January 8 2012.

19. Augment – 3D Augmented Reality [online]. Google Play.
URL: https://play.google.com/store/apps/details?id=com.ar.augment
Accessed September 17 2013.

20. StringAR FAQ [online]. String Labs.
URL: http://www.poweredbystring.com/faq
Accessed September 17 2013.

21. Artova Helsinki [online].
URL: http://www.artova.fi/
Accessed September 17 2013.

22. Koirapuistohanke [online]. Gammelin Koirat RY.
URL: https://sites.google.com/site/gammelinkoiratyhdistys/koirapuistohanke
Accessed September 17 2013.

23. Client Communication.

24. Layar – Introducing : Layar Vision [online]. Youtube.com.
URL: http://www.youtube.com/watch?v=AsD0DuPT1GI
Accessed February 15 2013.

25. Helsingin Kaupunkikalusteohje [online]. 1st ed. Helsingin kaupungin rakennusvirasto. 2010.
URL: http://www.hel.fi/hel2/hkr/julkaisut/kaluste/koko_ohje.pdf
Accessed February 14 2012

26. Wartmann C, Kauppi M. The Blender Gamekit. 2nd ed. Netherlands: Blender Foundation; 2008.

27. Hess, R. The Essential Blender: Guide to 3D Creation with the Open Source Suite Blender. Netherlands: Blender Foundation; 2007.

28. Blender FAQ – Why is Blender Free [online]. Blender Foundation.
URL: http://www.blender.org/education-help/faq/general/#c487
Accessed February 14 2012.

29. CGTextures [online]. CGTextures.
URL: http://cgtextures.com
Accessed March 2012.

30. Layar [online]. Layar.
URL: https://www.layar.com/
Accessed March 2012.

31. Layar 6.0 Developer Documentation [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/layar-platform-overview/
Accessed March 2012.

32. Layar 6.2 Developer Documentation [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/layar-platform-overview/
Accessed May 2013.

33. GetPOIs Response [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/api/getpois-response/
Accessed May 2013.

34. Hotspots [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/api/getpois-response/hotspots/
Accessed May 2013.

35. Calculate distance, bearing and more between Latitude/Longitude points [online]. Movable Type Scripts.
URL: http://www.movable-type.co.uk/scripts/latlong.html
Accessed May 2013.

36. Wikipedia Earth [online]. Wikipedia.
URL: http://en.wikipedia.org/wiki/Earth
Accessed May 2013.

37. Create a simple Geo-location layer [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/tutorials-tools/create-simple-geo-location-layer/
Accessed May 2013.

38. Ruby S, Thomas D, Hansson D. Agile Web Development with Rails. 4th ed. Pragmatic Programmers. March 2011.

39. Create a simple vision layer [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/tutorials-tools/create-simple-vision-layer/
Accessed May 2013.

40. Layar Converter History [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/3d-model-converter/model-converter-release-history/
Accessed May 2013.

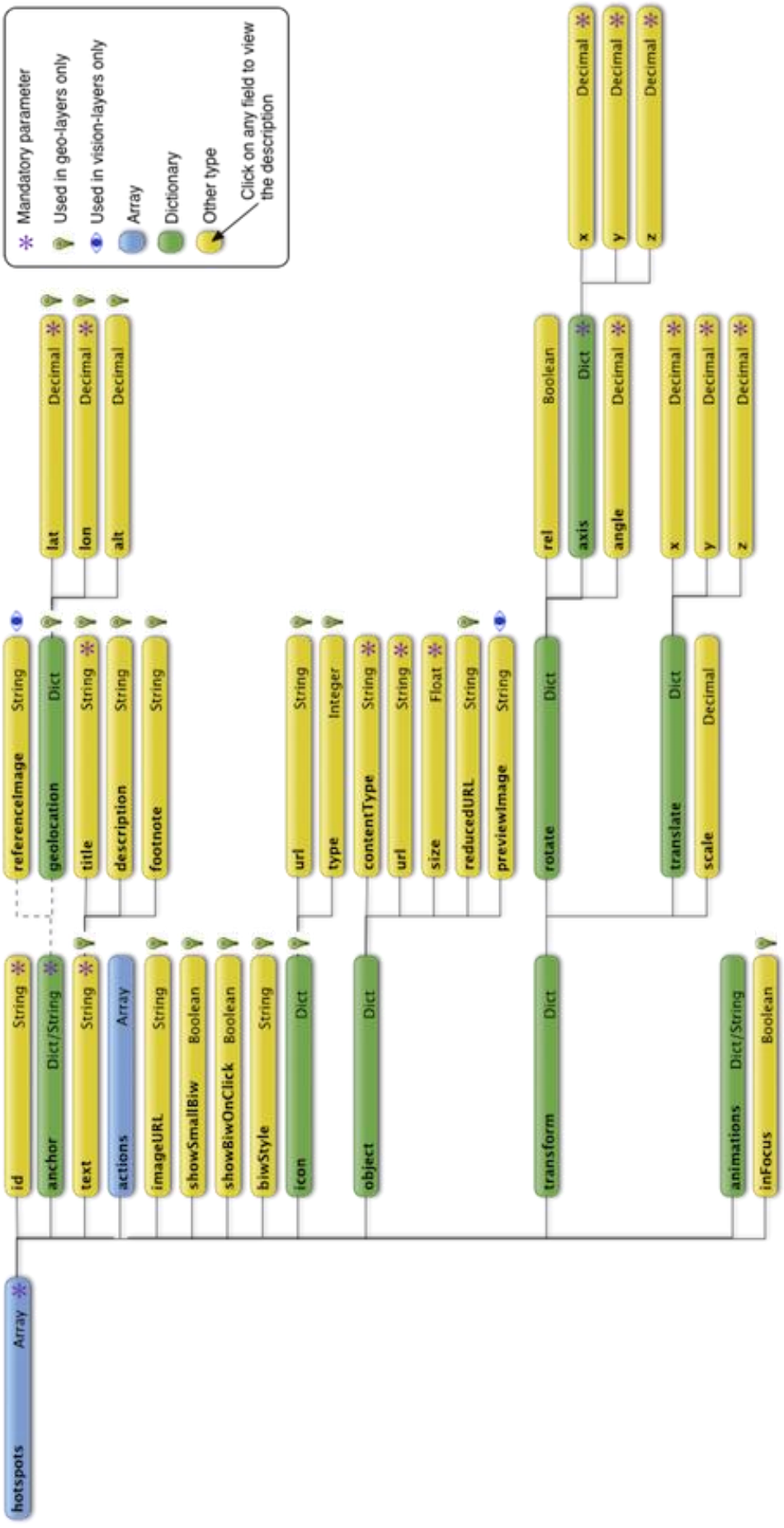
41. 3D Model Converter [online]. Layar Developer Documentation.
URL: https://www.layar.com/documentation/browser/3d-model-converter/
Accessed May 2013.

42. Wee J. Unity Technologies makes Unity 3D Mobile Basic free until 8 April, yours Forever [online]. E27. March 15 2012.
URL: http://e27.co/?p=20845
Accessed March 2012.

43. Lewis P. Create your own environment Maps.
URL: http://www.aerotwist.com/tutorials/create-your-own-environment-maps/
Accessed April 2012.

Appendix 1 – Map of Artova Koirapuisto [22;23]

Appendix 2 – Layar Hotspot Structure [34]



Appendix 3 – Full ARCMS Database Structure